VNiVERSiDAD D SALAMANCA

TRABAJO DE FIN DE GRADO

GRADO EN ESTADÍSTICA

Facultad de Ciencias

Curso 2023/2024

INTRODUCCIÓN AL APRENDIZAJE POR REFUERZO

REINFORCEMENT LEARNING INTRODUCTION

Autor: Adrián Hernández Pérez

Tutoras: Ana Belén Nieto Librero y Nerea González García

VNiVERSiDAD D SALAMANCA

TRABAJO DE FIN DE GRADO

GRADO EN ESTADÍSTICA

Facultad de Ciencias

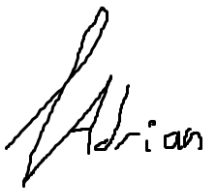
Curso 2023/2024

INTRODUCCIÓN AL APRENDIZAJE POR REFUERZO

Autor: Adrián Hernández Pérez

Tutoras: Ana Belén Nieto Librero y Nerea González García

Firma autor:



VNiVERSiDAD D SALAMANCA
CAMPUS DE EXCELENCIA INTERNACIONAL

Facvltad D Ciencias
VNiVERSiDAD D SALAMANCA

Certificado de los tutores TFG

Dña. Ana Belén Nieto Librero y Dña. Nerea González García, profesoras del departamento de Estadística de la Universidad de Salamanca,

HACEN CONSTAR:

Que el trabajo titulado "Introducción al aprendizaje por refuerzo", que se presenta, ha sido realizado por Adrián Hernández Pérez, con DNI ****9059A y constituye la memoria del trabajo realizado para la superación de la asignatura Trabajo de Fin de Grado en Estadística en esta Universidad.

Salamanca, 3 de julio de 2024

Fdo.: NIETO LIBRERO ANA BELEN - 11966438K
Firmado digitalmente por NIETO LIBRERO ANA BELEN - 11966438K
Fecha: 2024.07.03 23:10:20 +02'00'

Fdo.: GONZALEZ GARCIA NEREA - 72828390V
Firmado digitalmente por GONZALEZ GARCIA NEREA - 72828390V
Fecha: 2024.07.03 23:01:51 +02'00'

ÍNDICE

Resumen

Este trabajo explora el campo del aprendizaje por refuerzo, una forma de aprendizaje automático que permite a los agentes aprender y optimizar su comportamiento a través de la interacción con el entorno, sin necesidad de supervisión directa. El documento aborda los fundamentos del aprendizaje por refuerzo, incluyendo sus principales técnicas y algoritmos. También se discuten las aplicaciones prácticas del aprendizaje por refuerzo en áreas como juegos, robótica, y optimización de procesos industriales. Finalmente, a través de un caso práctico, se demuestra cómo implementar estos conceptos en un escenario real, enfatizando el potencial del aprendizaje por refuerzo para resolver problemas complejos y dinámicos que requieren decisiones secuenciales.

Abstract

This paper explores the field of reinforcement learning, a form of machine learning that allows agents to learn and optimize their behavior through interaction with the environment, without the need for direct supervision. The document addresses the fundamentals of reinforcement learning, including its main techniques and algorithms. It also discusses the practical applications of reinforcement learning in areas such as gaming, robotics, and industrial process optimization. Finally, through a practical case study, it demonstrates how to implement these concepts in a real-world scenario, emphasizing the potential of reinforcement learning to solve complex and dynamic problems that require sequential decisions.

CAPÍTULO 1: Introducción

1.1. Historia del aprendizaje automático

El aprendizaje automático, una rama de la inteligencia artificial, se enfoca en el desarrollo de algoritmos y técnicas que permiten a los ordenadores aprender y mejorar a partir de la experiencia. Su historia se remonta a varias décadas y ha evolucionado significativamente a lo largo del tiempo.

El término "aprendizaje automático" fue acuñado por Arthur Samuel en 1959. Samuel definió este campo como una rama de la inteligencia artificial que emplea técnicas estadísticas y algoritmos computacionales para permitir que las computadoras "aprendan", es decir, mejoren su desempeño en una tarea específica tras procesar una gran cantidad de datos, sin depender de instrucciones explícitas externas proporcionadas por el programador, las cuales podrían estar sesgadas (Núñez Reiz et al., 2019; Vemuri, 2020).

Otras influencias para el aprendizaje automático provienen de la teoría de la computación propuesta por Alan Turing. Turing planteó en 1950 la pregunta "¿pueden las máquinas pensar?" en su famoso artículo "*Computing Machinery and Intelligence*"(Turing, 1950). Describiendo lo que ahora se conoce como la prueba de Turing, propuso que una máquina puede considerarse inteligente si un evaluador humano tiene dificultad para distinguir entre las respuestas de una máquina y las de un humano, basándose únicamente en interacciones textuales. La idea es que, en un juego, a un interrogador se le asigne la tarea de determinar si un jugador es una computadora o un humano, eligiendo entre dos participantes, A y B, utilizando únicamente respuestas escritas a preguntas planteadas. En la actualidad es un desafío considerable programar un ordenador para que supere esta prueba. (Colliot, 2023; Zhang et al., 2023).

En 1956, durante la histórica conferencia de *Dartmouth*, se planteó la idea de que las máquinas podrían aprender y razonar como los seres humanos. Esta conferencia, organizada por John McCarthy, Marvin Minsky, Nathaniel Rochester y Claude Shannon, es ampliamente considerada como el evento fundacional de la inteligencia artificial. El objetivo del taller era explorar la posibilidad de que cada aspecto del aprendizaje o cualquier otra característica de la inteligencia puede ser descrito tan precisamente que una máquina puede ser construida para simularlo (Colliot, 2023; Crevier, 1993; Nilsson, 2009).

Entre los años 1960 y 1970, se desarrollaron teorías fundamentales y modelos básicos de aprendizaje automático. Marvin Minsky y Seymour Papert publicaron "*Perceptrons*" en 1969, una obra crucial que analizó las capacidades y limitaciones de los perceptrones, un tipo de red neuronal simple. Sin embargo, también señalaron las limitaciones de estos modelos, lo que llevó a un estancamiento en la investigación de redes neuronales (Zhang et al., 2023).

En 1979, estudiantes de ingeniería de la Universidad de Stanford desarrollaron el "*Stanford Cart*", un robot capaz de moverse por una habitación esquivando diferentes objetos. Este avance fue posible gracias al algoritmo del vecino más cercano, que permitió al robot reconocer patrones y navegar de manera autónoma. Este algoritmo de reconocimiento de patrones es fundamental en la inteligencia artificial, ya que dota a las

máquinas de la capacidad de aprender y anticipar respuestas efectivas basadas en los datos recopilados (Crevier, 1993; McCorduck, 1979).

El interés en las redes neuronales resurgió en la década de 1980 con la introducción del algoritmo de retropropagación por (Rumelhart et al. 1986). Este algoritmo permitió entrenar redes neuronales de múltiples capas, superando algunas de las limitaciones anteriores y revitalizando el campo de la inteligencia artificial. La retropropagación es un método de cálculo del gradiente que ajusta iterativamente los pesos de las conexiones en la red para minimizar el error entre la salida deseada y la salida real (Colliot, 2023).

El interés en el aprendizaje por refuerzo comenzó a incrementarse significativamente en la década de 1980. Uno de los momentos clave fue en 1986, cuando Richard Sutton publicó su trabajo sobre el aprendizaje temporal-diferido. Este método combinaba elementos de control óptimo y aprendizaje automático, y jugó un papel fundamental en el desarrollo de algoritmos de aprendizaje por refuerzo (Sutton & Barto, 2014).

La mejora en la capacidad computacional y la disponibilidad de grandes conjuntos de datos durante la década de 1990 llevaron a una adopción más amplia del aprendizaje automático. Se desarrollaron algoritmos de aprendizaje supervisado y no supervisado más avanzados, como las máquinas de soporte vectorial y los métodos de agrupamiento como K-medias (Colliot, 2023).

El aprendizaje profundo comenzó a ganar prominencia en la primera década de los 2000 debido a varios avances significativos en hardware, especialmente las unidades de procesamiento gráfico (*Graphics Processing Unit*, GPU). Estos avances permitieron entrenar modelos complejos y grandes de manera más eficiente y rápida. Las GPU, inicialmente diseñadas para renderizar gráficos, demostraron ser muy efectivas para los algoritmos de aprendizaje profundo debido a su alta capacidad de procesamiento paralelo. Uno de los hitos clave en este período fue el desarrollo de redes neuronales profundas, que demostraron su eficacia en una variedad de tareas, incluyendo el reconocimiento de voz, visión por computadora y procesamiento del lenguaje natural (Bengio et al., 2021).

En la última década, el aprendizaje automático se ha consolidado como una tecnología clave en la industria y la investigación. Nuevos paradigmas, como el aprendizaje por refuerzo, han demostrado su eficacia en problemas complejos y dinámicos.

El aprendizaje por refuerzo es un área del aprendizaje automático en la que un agente aprende a tomar decisiones mediante la interacción con su entorno. El objetivo del agente es aprender una política que maximice la recompensa acumulada a lo largo del tiempo. Este proceso se basa en el principio de ensayo y error, donde el agente recibe retroalimentación en forma de recompensas o penalizaciones, que utiliza para ajustar sus acciones futuras. A diferencia de otras técnicas de aprendizaje automático, el aprendizaje por refuerzo no requiere un conjunto de datos previamente etiquetado. En su lugar, el agente debe explorar el entorno para descubrir cuáles acciones producen las mejores recompensas, aprendiendo de manera autónoma a mejorar su desempeño (Sutton & Barto, 2014).

Para comprender mejor el aprendizaje por refuerzo, es útil contrastarlo con las técnicas tradicionales de aprendizaje automático: el aprendizaje supervisado y el aprendizaje no supervisado. Además, también abordaremos el aprendizaje semisupervisado, que combina aspectos de ambos enfoques.

1.2. Aprendizaje supervisado, no supervisado y semi-supervisado

El aprendizaje supervisado es un tipo de aprendizaje automático donde el modelo se entrena utilizando un conjunto de datos etiquetados. Esto significa que cada entrada de entrenamiento x_i con $i = 1, \dots, n$ está acompañada de una etiqueta o resultado esperado y_i. El objetivo es que el modelo aprenda a correlacionar las entradas con las salidas correctas, y pueda hacer predicciones precisas para información recién adquirida o previamente no examinada. Entre las técnicas comúnmente utilizadas en este enfoque de aprendizaje se encuentran el clasificador Naive Bayes, basado en los principios establecidos por Thomas Bayes (Bayes & Prince, 1763), las Máquinas de soporte vectorial, desarrolladas inicialmente por Vladimir Vapnik acompañado de sus dos amigos Boser y Guyon (Boser et al., 1996) y los árboles de decisión, una metodología popularizada por Ross Quinlan (Quinlan, 1986). El proceso de aprendizaje supervisado se divide en dos categorías principales: clasificación y regresión.

Hablamos de clasificación cuando el objetivo es asignar etiquetas discretas a las observaciones, como por ejemplo clasificar correos electrónicos como 'spam' o 'no spam' o imágenes médicas como ‘cáncer’ o ‘no cáncer’. Por otro lado, en la regresión las etiquetas son valores continuos, como predecir el precio de una casa basado en sus características o predecir el clima en base a variables meteorológicas como la humedad o la velocidad del viento (James et al., 2013; Murphy, 2012).

El aprendizaje no supervisado se utiliza cuando los datos de entrenamiento no están etiquetados, es decir, para cada observación i, donde $i = 1, \dots, n$, observamos un vector de mediciones x_i pero no una respuesta asociada y_i. El objetivo es descubrir patrones ocultos o estructuras en los datos. Existen varias técnicas de aprendizaje no supervisado, entre las cuales se destacan el análisis de conglomerados, en el cual se agrupan los datos en conjuntos basados en la similitud entre ellos, es decir, determina sobre la base de $x_1, \dots, x_n$ si las observaciones se agrupan en grupos relativamente distintos. Un ejemplo común es el algoritmo k-medias, desarrollado inicialmente por Stuart Lloyd en 1957 y publicado en 1982 (Lloyd, 1982), y de forma independiente por James MacQueen en 1967 (MacQueen, 1967). Otro método destacable es la reducción de la dimensionalidad, en la cual se reduce el número de variables en los datos mientras se preserva la mayor cantidad posible de información, es decir, se transforma un conjunto de datos que contiene muchas variables, denominadas dimensiones, en un formato donde las variables son menos en número, pero siguen representando la esencia o las características más importantes de los datos originales. Ejemplos de este método pueden ser técnicas como análisis de componentes principales, introducido por Karl Pearson en 1901 (Pearson, 1901) o el análisis de correspondencias, desarrollado por Jean-Paul Benzécri en la década de 1960 (Benzécri, 1973; James et al., 2013; Murphy, 2012).

El aprendizaje semi-supervisado es una subdisciplina del aprendizaje automático que busca resolver problemas utilizando tanto datos etiquetados como no etiquetados. En muchos escenarios de aplicación, como en tareas típicas de diseño estructural, es común encontrar una gran cantidad de datos no etiquetados y solo una pequeña cantidad de datos etiquetados. Aprovechando estos datos no etiquetados, se espera que el aprendizaje semisupervisado pueda superar el rendimiento obtenido cuando se utilizan únicamente datos etiquetados. Un método clásico de aprendizaje semisupervisado es el autoentrenamiento, utilizado originalmente en tareas de clasificación. Este método, que

implica entrenar inicialmente un clasificador con un conjunto de datos etiquetados de tamaño reducido, luego utilizar este clasificador para asignar etiquetas provisionales a los datos no etiquetados y, finalmente, volver a entrenar el clasificador con el conjunto de datos ampliado que incluye estas nuevas etiquetas, fue desarrollado por (Yarowsky, 1995) en su estudio sobre desambiguación del sentido de las palabras. Este enfoque también puede aplicarse a tareas de regresión más generales (Fei et al., 2023).

1.3. Aprendizaje por refuerzo

El aprendizaje por refuerzo difiere significativamente de estos enfoques tradicionales. Mientras que el aprendizaje supervisado y no supervisado trabajan con conjuntos de datos estáticos, el aprendizaje por refuerzo involucra la interacción continua con un entorno dinámico. El agente de aprendizaje por refuerzo debe aprender a través de la experiencia, tomando acciones que afectan el estado del entorno y recibiendo recompensas que guían su comportamiento. El agente es la entidad que toma decisiones y aprende de las consecuencias de sus acciones, mientras que el entorno es todo aquello con lo que el agente interactúa y que puede cambiar en respuesta a las acciones del agente. El aprendizaje por refuerzo se basa en la formulación de los procesos de decisión de Markov (PDM). Un PDM es una estructura matemática que modela la interacción de un agente con su entorno a través de estados S, que representan las diferentes situaciones en las que puede encontrarse el agente. A cada estado, se asocian acciones posibles A, que el agente puede decidir ejecutar. La transición entre estados se rige por una función de probabilidad $P(s, a, s')$, que determina la probabilidad de transitar al estado s' desde el estado s al tomar la acción a. Cada transición está también asociada con una función de recompensa $R(s, a, s')$, que otorga una recompensa específica basada en la acción tomada y el cambio de estado resultante. El objetivo primordial en el aprendizaje por refuerzo es desarrollar una política π, que es una estrategia que define la acción óptima $\pi(s)$ a tomar en cada estado s para maximizar la suma total de las recompensas futuras esperadas. La eficacia de una política se mide mediante la función de valor $V^{\pi}(s)$, que calcula el valor esperado de las recompensas acumuladas desde el estado s siguiendo la política π. Este marco de trabajo no solo permite al agente aprender de las experiencias acumuladas, ajustando sus decisiones para optimizar las recompensas a lo largo del tiempo, sino que también facilita la exploración sistemática y la explotación de conocimientos para alcanzar decisiones cada vez más efectivas (Sutton & Barto, 2014).

A lo largo de los últimos años, el aprendizaje por refuerzo se ha consolidado como un método ampliamente utilizado debido a su notable capacidad para abordar problemas desafiantes en la toma de decisiones secuenciales, un concepto detalladamente explorado por Richard S. Sutton y Andrew G. Barto (Sutton & Barto, 2014). Esta serie de logros se deben a la combinación del aprendizaje por refuerzo con técnicas de aprendizaje profundo. Esta combinación recibe el nombre de aprendizaje por refuerzo profundo y es especialmente útil en problemas con espacios de estados de alta dimensionalidad. En el pasado, el diseño del aprendizaje por refuerzo presentaba un problema significativo en la selección de características. Esto se debía a que la alta dimensionalidad de los espacios de estado dificultaba la creación de modelos eficaces sin una selección experta y manual de características relevantes. Además, las limitaciones computacionales de la época restringían la capacidad de procesar grandes volúmenes de datos y características complejas, lo que impedía el desarrollo de políticas óptimas. Sin embargo, el aprendizaje

por refuerzo profundo ha superado estos obstáculos al permitir que las redes neuronales aprendan automáticamente representaciones efectivas de los datos, descubriendo niveles de abstracción que facilitan la generalización a nuevas situaciones y entornos. Esta capacidad para aprender directamente de los datos sin procesar, sin la necesidad de una selección de características predefinida, ha permitido el éxito del aprendizaje por refuerzo en tareas complejas con un conocimiento previo limitado (François-Lavet et al., 2018).

Como podemos observar en la *Figura 1*, el aprendizaje automático se divide en cinco categorías: aprendizaje supervisado, no supervisado, semi-supervisado, por refuerzo y por refuerzo profundo.

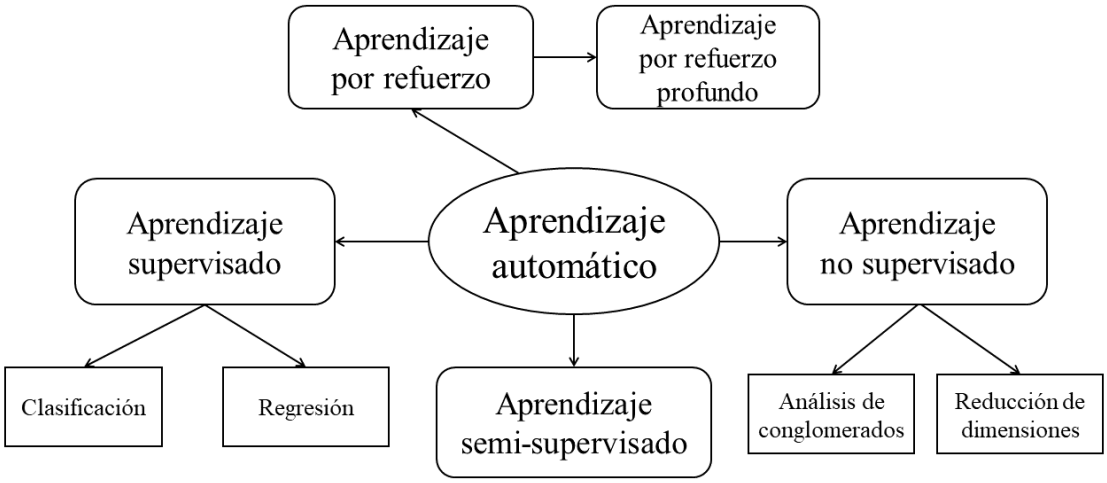


Figura 1. Tipos de aprendizaje automático

La clasificación de los algoritmos de aprendizaje por refuerzo se basa en tres criterios distintos: basados en modelo o libres de modelo, con política o sin política, y, por último, basados en valor o basados en política. Los algoritmos basados en modelo, que construyen y utilizan un modelo explícito del entorno, y los algoritmos libres de modelo, que operan directamente a partir de la experiencia adquirida, fueron explorados en profundidad en el libro *'Reinforcement Learning: An Introduction'* de (Sutton & Barto, 2014). Por otro lado, los algoritmos con política se beneficiaron inicialmente de las contribuciones de Ronald Williams y fueron desarrollados posteriormente por Sutton en varios trabajos (Williams, 1992). Los algoritmos sin política, que incluyen técnicas como el método de diferencia temporales de Sutton (Sutton, 1988), y los métodos basados en valor, detallados por Watkins en su desarrollo del algoritmo *Q-Learning* (Watkins & Dayan, 1992), contrastan con los métodos basados en política, los cuales ajustan directamente la política a seguir por el agente.

Los algoritmos basados en modelo construyen un modelo explícito del entorno, lo que incluye la función de transición y la función de recompensa. La función de transición $P(s' \mid s, a)$ especifica la probabilidad de moverse de un estado s a un nuevo estado s' al tomar una acción a. La función de recompensa $R(s, a)$ proporciona la recompensa esperada al tomar la acción a en el estado s. Un ejemplo de algoritmo basado en modelo podría ser la iteración de valores, método el cual consiste en calcular iterativamente el valor esperado de cada estado bajo una política hasta que los valores convergen. Los algoritmos libres de modelo no construyen un modelo explícito del entorno. En lugar de eso, estos algoritmos aprenden directamente de la interacción con el entorno. Esto significa que el agente ajusta sus estimaciones de valor basadas en las recompensas

observadas y las transiciones de estado reales que experimenta a medida que realiza acciones. Un ejemplo de algoritmo libre de modelo es *Q-Learning*, el cual aprende la función de valor de acción $Q(s, a)$ sin requerir un modelo del entorno (Sutton & Barto, 2014).

Los algoritmos con política aprenden directamente una política $\pi(a \mid s)$, es decir, estos algoritmos determinan directamente qué acción tomar en cada estado, optimizando la política para maximizar la recompensa acumulada. Un algoritmo con política es *Proximal policy optimization*, desarrollado por (Schulman et al., 2017), el cual emplea una metodología que se fundamenta en el cálculo de gradientes para ajustar la política. Este cálculo de gradientes señala la trayectoria por la cual se deben modificar los parámetros para optimizar, ya sea maximizando o minimizando, la función objetivo. Los algoritmos sin política, por otro lado, se dedican a aprender una función de valor, representando el valor esperado de las recompensas futuras. Las decisiones se toman derivando la política a partir de la función de valor aprendida, generalmente eligiendo la acción que maximiza la función de valor en cada estado. Por ejemplo, el algoritmo Q-Network profundo, descrito por (Mnih et al., 2015),combina *Q-Learning* con redes neuronales profundas para aprender la función de valor de acción en entornos complejos y de alta dimensionalidad.

Los algoritmos basados en valor buscan aprender una función de valor, la cual puede ser la función de valor de estado $V(s)$, que representa el valor esperado de las recompensas futuras acumuladas comenzando desde el estado s y siguiendo una política específica, o la función de valor de acción $Q(s, a)$, que representa el valor esperado de tomar la acción a en el estado s después de seguir la política óptima. Estas funciones de valor estiman el valor esperado de las recompensas futuras que el agente puede obtener desde un estado específico o al tomar una acción específica. Dos ejemplos de algoritmos basados en valor pueden ser *Q-Learning*, desarrollado por Watkins (Watkins & Dayan, 1992) o SARSA, detallado por Rummery y Niranjan (Rummery & Niranjan, 1994). Los algoritmos basados en política se enfocan en aprender directamente la política que determina qué acciones debe tomar el agente en cada estado. A diferencia de los algoritmos basados en valor, que primero estiman una función de valor y luego derivan una política a partir de ella, los algoritmos basados en política ajustan la propia política para maximizar la recompensa acumulada esperada. Este enfoque permite una optimización más directa y eficiente de la política de decisión del agente. *Reinforce* es un algoritmo basado en política que ajusta la política en la dirección que maximiza la expectativa de la recompensa, como lo describió Williams en su investigación seminal (Sutton & Barto, 2014; Williams, 1992).

1.4. Estado actual del aprendizaje por refuerzo

En las últimas décadas, el aprendizaje por refuerzo ha experimentado un crecimiento significativo debido a tres factores principales: el aumento de la capacidad computacional, la disponibilidad de grandes cantidades de datos y el desarrollo de algoritmos más eficientes. Estos avances han permitido que el aprendizaje por refuerzo se aplique con éxito en una variedad de campos. (Sutton & Barto, 2014) destacan cómo estos desarrollos han ampliado las aplicaciones del aprendizaje por refuerzo en diversos sectores. Además, (Jordan & Mitchell, 2015) analizan cómo las mejoras en la capacidad computacional y el acceso a extensos conjuntos de datos han sido cruciales para los avances en el campo. Por otro lado, (Li, 2018) proporciona una visión general de cómo

la evolución de algoritmos más eficientes ha impulsado el uso del aprendizaje por refuerzo en aplicaciones más complejas y variadas.

Uno de los hitos más importantes en el campo del aprendizaje por refuerzo fue el desarrollo de *AlphaGo* por *Google DeepMind*. AlphaGo es un programa de inteligencia artificial que combina técnicas de aprendizaje profundo y aprendizaje por refuerzo para jugar al juego de mesa Go. En 2016, AlphaGo venció a Lee Sedol, uno de los mejores jugadores de Go del mundo, en una serie de partidas. Este logro fue extremadamente notable debido a la complejidad del juego de Go, el cual tiene una enorme cantidad de combinaciones posibles de movimientos. Otra contribución significativa en el campo del aprendizaje por refuerzo proviene de OpenAI, una organización de investigación en inteligencia artificial. OpenAI desarrolló "*OpenAI Five*", un equipo de inteligencia artificial que puede jugar al juego de estrategia en tiempo real "*Dota 2*" a nivel competitivo. Este logro demostró que los algoritmos de aprendizaje por refuerzo pueden manejar entornos muy complejos y dinámicos, donde las decisiones deben tomarse en tiempo real y las situaciones cambian constantemente. El aprendizaje por refuerzo también se ha aplicado con éxito en la robótica. En este campo, los robots utilizan el aprendizaje por refuerzo para aprender a realizar tareas complejas de manera autónoma. Un ejemplo notable es el trabajo de (Levine et al., 2016), quienes desarrollaron un sistema que permite a los robots coordinar sus movimientos de mano y ojo para agarrar objetos. Utilizaron técnicas de aprendizaje profundo y recolección de datos a gran escala para entrenar a los robots (Berner et al., 2019; Silver et al., 2017).

A pesar de los avances significativos en el aprendizaje por refuerzo, este campo enfrenta varios desafíos que limitan su desarrollo. Sin embargo, también ofrece grandes oportunidades para la innovación y mejora en diversos sectores. A continuación, se detallan las principales limitaciones y posibles aplicaciones.

1.5. Limitaciones y posibles aplicaciones

El aprendizaje por refuerzo tiene algunas limitaciones, pero también presenta muchas oportunidades de aplicación. Una limitación importante es que muchos métodos de aprendizaje por refuerzo dependen de estimar funciones de valor para resolver problemas, lo que puede ser menos eficiente que otros enfoques, como los algoritmos genéticos. Estos algoritmos genéticos evalúan el comportamiento a lo largo de la vida de muchos agentes no aprendices, cada uno utilizando una política diferente para interactuar con su entorno, y seleccionan aquellos que son capaces de obtener la mayor recompensa. No utilizan funciones de valor y pueden ser efectivos en situaciones donde las políticas son pocas o fáciles de encontrar. Sin embargo, los algoritmos genéticos no aprovechan toda la información disponible sobre cómo los estados y las acciones se relacionan entre sí, lo que puede hacer que el proceso de búsqueda sea menos eficiente. Otro desafío significativo en el aprendizaje por refuerzo es la gestión de espacios de estado grandes, que pueden hacer inviable representar y aprender valores para cada estado individualmente, requiriendo técnicas avanzadas como las redes neuronales profundas para su aproximación. Además, la convergencia lenta es otro problema crítico, ya que el proceso de aprender una política óptima puede ser extremadamente largo debido a la alta dimensionalidad del espacio de estado, la variabilidad en las recompensas y la necesidad de equilibrar exploración y explotación. Encontrar el equilibrio adecuado entre explorar nuevas estrategias y explotar el conocimiento ya adquirido es crucial; si se explora

demasiado, se puede perder tiempo y recursos, mientras que, si se explota demasiado, se pueden pasar por alto mejores soluciones (Sutton & Barto, 2014).

A pesar de estas limitaciones, el aprendizaje por refuerzo tiene un gran potencial en muchas áreas. Por ejemplo, puede ser de gran ayuda en la medicina centrada en el paciente, optimizando tratamientos médicos según las necesidades individuales de los pacientes. También puede mejorar la capacidad de los robots para realizar tareas complejas y adaptarse a cambios en su entorno, lo que sería muy útil en la manufactura y la logística. Además, los algoritmos de aprendizaje por refuerzo pueden optimizar el uso de recursos en la gestión de energía y la agricultura, promoviendo prácticas más sostenibles. Métodos como los gradientes de política, que ajustan directamente las estrategias basándose en interacciones detalladas con el entorno, pueden proporcionar soluciones más eficientes y robustas. Estos métodos permiten aprovechar mejor el potencial del aprendizaje por refuerzo en diversas aplicaciones, maximizando su efectividad y eficiencia (Li, 2018; Sutton & Barto, 2014).

1.6. Objetivos

El objetivo general de mi trabajo es analizar el funcionamiento del aprendizaje por refuerzo, investigando sus algoritmos, limitaciones y soluciones potenciales, y demostrando su aplicabilidad a través de un caso práctico que ilustre su implementación en un entorno específico.

A continuación, se presentan los objetivos específicos que orientarán el análisis y la exploración en este trabajo. Estos objetivos buscan aclarar y profundizar en distintos aspectos del aprendizaje por refuerzo, desde sus principios básicos hasta su implementación práctica:

- Explicar qué es el aprendizaje por refuerzo y cómo se diferencia de otros paradigmas de aprendizaje automático.
- Definir los elementos básicos de un entorno de aprendizaje por refuerzo, incluyendo agentes, estados, acciones y recompensas.
- Describir qué son las políticas en el aprendizaje por refuerzo, cómo se utilizan las funciones de valor para evaluar la calidad de las políticas y los métodos para determinar la política óptima que maximice la recompensa acumulada a largo plazo.
- Analizar los algoritmos basados en modelos, incluyendo la evaluación de la política (predicción) y la iteración de política e iteración de valores (control).
- Analizar los métodos libres de modelos, abarcando algoritmos como Monte-Carlo y diferencias temporales (predicción), como algoritmos como *Q-Learning* y SARSA (control).

- Demostrar cómo la integración del aprendizaje profundo con el aprendizaje por refuerzo permite abordar desafíos en entornos con grandes espacios de estados y acciones.

- Investigar desafíos típicos en el aprendizaje por refuerzo, como el dilema de exploración contra explotación y la alta dimensionalidad.

- Implementar un caso práctico que demuestre la aplicación de un algoritmo de aprendizaje por refuerzo.

CAPÍTULO 2: Aprendizaje por refuerzo

En años recientes, el ámbito del aprendizaje automático ha visto un progreso destacable gracias al surgimiento de técnicas novedosas que superan los enfoques supervisados y no supervisados convencionales. Estos avances son un salto importante hacia el desarrollo de sistemas de inteligencia artificial que sean más versátiles, independientes y efectivos. Entre las metodologías más notables se halla el aprendizaje por refuerzo, que ha transformado la forma en que los ordenadores se relacionan con su ambiente y manejan la información.

En la última década, el aprendizaje por refuerzo ha capturado un creciente interés dentro de las comunidades de aprendizaje automático e inteligencia artificial, gracias a su atractiva propuesta de educar a los agentes mediante incentivos y sanciones. Este tipo de aprendizaje permite a un agente determinar cuál es el mejor curso de acción o qué decisiones debe tomar para optimizar una recompensa. En este proceso, al agente no se le indica directamente qué acciones realizar, en su lugar, debe identificar mediante un enfoque de ensayo y error, las acciones que le generan mayores recompensas. Cuando el agente de aprendizaje por refuerzo ejecuta una acción adecuada, obtiene una recompensa positiva, en cambio, si comete un error, esta consecuencia es negativa (Sivamayil et al., 2023).

Este tipo de aprendizaje tiene una gran variedad de aplicaciones en diversas áreas. En los juegos, sigue siendo fundamental para desarrollar y probar algoritmos. En robótica, es fundamental para permitir que los robots aprendan y se adapten a su entorno en tiempo real, lo que es esencial en la era de la inteligencia artificial. El procesamiento del lenguaje natural utiliza muy frecuentemente este tipo de aprendizaje para tareas como la traducción automática y la generación de texto, mejorando la calidad y coherencia de los resultados. También se aplica en la gestión empresarial para optimizar anuncios, recomendaciones y la gestión de clientes. En el ámbito financiero, se utiliza para el modelado y la toma de decisiones en mercados complejos. En atención médica, ha ganado protagonismo tras la implementación del aprendizaje por refuerzo profundo, ayudando en diagnósticos y tratamientos personalizados. La industria también se beneficia de la integración de aprendizaje por refuerzo en la manufactura, optimizando procesos y reduciendo costos (Li, 2018).

Es importante hacer una distinción clara entre el aprendizaje por refuerzo y las técnicas tradicionales, como el aprendizaje supervisado y el no supervisado. Por un lado, el aprendizaje supervisado se enfoca en capacitar a los sistemas para que generalicen sus decisiones y actúen de manera efectiva en situaciones nuevas que no se encontraron en el conjunto de entrenamiento, donde cada ejemplo de este conjunto de entrenamiento incluye una etiqueta o salida deseada. Aunque el aprendizaje supervisado es esencial, su efectividad es limitada cuando se trata de adquirir conocimiento a través de la interacción directa con el entorno. En escenarios interactivos, resulta poco práctico obtener ejemplos exhaustivos y precisos del comportamiento deseado para cada posible situación. En contextos desconocidos, donde el aprendizaje tiene un valor incalculable, es crucial que los agentes puedan aprender de sus propias experiencias. Por otro lado, el aprendizaje no supervisado tiene como objetivo la detección de patrones en conjuntos de datos sin etiquetar. Aunque podría parecer que los paradigmas de aprendizaje supervisado y no supervisado cubren todo el espectro del aprendizaje automático, el aprendizaje por refuerzo se presenta como una categoría distinta. A diferencia del aprendizaje no

supervisado, el aprendizaje por refuerzo no se centra en la identificación de estructuras ocultas dentro de los datos, sino en la optimización de una señal de recompensa. Aunque el descubrimiento de patrones puede ser útil dentro del aprendizaje por refuerzo, no aborda el desafío principal de maximizar dicha señal de recompensa. De este modo, el aprendizaje por refuerzo se establece como un tercer paradigma dentro del aprendizaje automático, diferente tanto del aprendizaje supervisado como del no supervisado (Sutton & Barto, 2014).

2.1. Funcionamiento de un entorno de aprendizaje por refuerzo

En un marco de aprendizaje por refuerzo, un agente está conectado a su entorno a través de la percepción y la realización de acciones, que se llevan a cabo en una secuencia discreta de pasos temporales $t = 0,1,2,3, \ldots$. Este proceso implica que el agente interactúe con el entorno de manera continua y secuencial. Definimos las acciones y estados del entorno en un momento dado t como $a_t \in A$ donde A es el conjunto de todas las posibles acciones y $s_t \in S$, donde S es el conjunto de estados posibles, lo que se conoce como el espacio de estados. Durante cada interacción, el agente recibe una indicación del estado actual s_t del entorno. A partir de esta información, el agente selecciona una acción a_t para ejecutar. Esta acción cambia el estado del entorno, y esta transición de estado se comunica al agente mediante una señal de refuerzo escalar r_{t+1}. La función de recompensa $R: S \times A \rightarrow R$ devuelve la recompensa inmediata $R(s_t, a_t)$ que el agente recibe después de realizar la acción a_t en el estado s_t. El proceso de aprendizaje por refuerzo se basa en la capacidad del agente para aprender a lo largo del tiempo mediante prueba y error. A través de estas interacciones repetidas con el entorno, el agente ajusta sus acciones para maximizar la recompensa acumulada a largo plazo. La función de transición $P(s_t, a_t, s_{t+1}): S \times A \times S \rightarrow [0,1]$ es una función Markoviana y calcula la probabilidad de transición del estado s_t al estado s_{t+1} después de tomar la acción a_t. El objetivo principal del agente es maximizar el valor acumulado de las recompensas futuras. La recompensa puede ser positiva o negativa, dependiendo de la acción realizada, lo que ayuda al agente a distinguir entre eventos beneficiosos y perjudiciales (Sutton & Barto, 2014; Wiering & van Otterlo, 2012; Yu et al., 2020).

En la *Figura 2* se puede observar gráficamente cómo interactúa el agente con el entorno, proporcionando una visualización clara de este ciclo de percepción, acción y refuerzo. Esta figura ilustra cómo el agente utiliza la información del estado actual para tomar decisiones, cómo estas decisiones afectan al entorno y cómo las recompensas obtenidas influyen en futuras decisiones. Este ciclo continuo es fundamental para el aprendizaje por refuerzo, permitiendo que el agente mejore su rendimiento a lo largo del tiempo mediante la adaptación y la mejora constante de sus estrategias.

Un ejemplo de un entorno de aprendizaje por refuerzo extrapolado a la vida real podría ser de ayuda para entender cómo funcionan cada una de sus partes. En el contexto de un niño aprendiendo a montar en bicicleta, el agente sería el niño que está aprendiendo a andar en bicicleta con el objetivo de mantener el equilibrio, moverse y navegar sin caerse. El entorno es todo lo que rodea al niño, incluyendo la bicicleta, el suelo, las pendientes, los obstáculos y las condiciones meteorológicas. Los estados serían las diferentes situaciones en las que se encuentra el niño durante su aprendizaje. Estos estados pueden incluir el estado inicial de estar en la bicicleta, pero con los pies en el suelo, el estado de mantener la bicicleta recta sin caerse, el estado de caerse de la bicicleta y el estado de

andar en la bicicleta sin ayuda. Las acciones son las decisiones que el niño puede tomar en cada momento. Estas pueden incluir empujar con los pies para empezar a moverse, pedalear para mantener la velocidad, usar el manillar para girar o frenar para detenerse. Las recompensas son los resultados de las acciones del niño, que guían su aprendizaje.

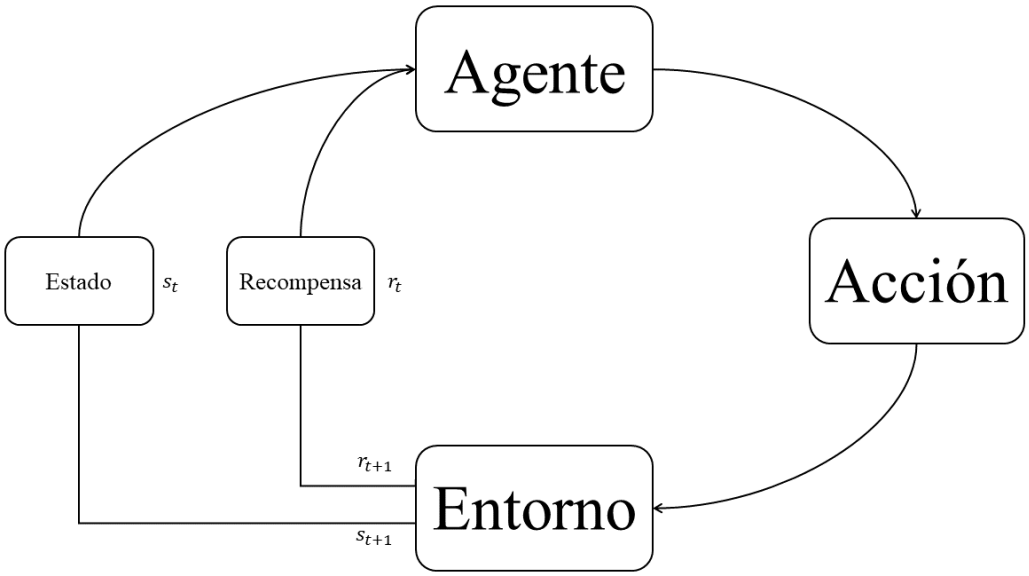


Figura 2. Proceso de interacción entre el agente y el entorno

La recompensa positiva sería no caerse, avanzar una distancia sin problemas, recibir la aprobación de sus padres, mientras que la recompensa negativa sería caerse, no poder mantener el equilibrio o recibir una advertencia por hacerlo mal. El niño, como agente, percibe su estado actual, evaluando si está en equilibrio o a punto de caerse, y utiliza esta información para decidir su siguiente acción, como empujar los pedales, girar el manillar, o frenar. Cada acción provoca una transición en el entorno, es decir, si empuja los pedales, la bicicleta comienza a moverse, si gira el manillar demasiado rápido, puede caerse. Dependiendo del resultado de sus acciones, el niño recibe una recompensa: positiva si mantiene el equilibrio y avanza, o negativa si se cae. Basado en estas recompensas y el nuevo estado, el niño ajusta su estrategia, aprendiendo a girar más suavemente si una caída previa fue causada por un giro brusco. Este proceso de percepción, acción, transición, recompensa y actualización se repite continuamente, permitiendo al niño mejorar su habilidad para montar en bicicleta a través de la experiencia y la retroalimentación constante.

Una mayor variabilidad en la realización de acciones nuevas provoca un mayor conocimiento del entorno, lo cual podemos llamar proceso de exploración. En este proceso exploratorio puede que no tengamos recompensas muy beneficiosas a corto plazo, pero nos ayudará a obtenerlas en el largo plazo en el futuro. Cuando se tiene un suficiente conocimiento del entorno podemos llevar a cabo el proceso de explotación, en el cual el agente utilizará todos los conocimientos que tiene sobre las acciones pasadas y actuara para conseguir la máxima recompensa posible. En el ejemplo mencionado arriba, la exploración es el proceso que hace el niño para descubrir nuevas formas de mantener

el equilibrio, mientras que el proceso de explotación es la repetición frecuente de las acciones que ve que son más efectivas para no caerse y avanzar correctamente gracias al conocimiento adquirido en el proceso exploratorio. El proceso de exploración es muy importante ya que a priori podríamos guiarnos por la recompensa más alta al realizar una determinada acción, pero al conocer más el entorno nos podríamos dar cuenta que recibir menos recompensa al realizar una serie de acciones puede llevarnos a recibir una recompensa final mucho más alta que si nos hubiéramos guiado por la recompensa inicial (Sutton & Barto, 2014; Wiering & van Otterlo, 2012).

Una política π es la estrategia que define las acciones que debe tomar el agente en diferentes estados para maximizar las recompensas acumuladas. El objetivo principal del agente es encontrar una política que sea óptima. Esta optimización se logra aprendiendo a partir de las recompensas recibidas, las cuales son consecuencia de las acciones que el agente realiza. En el ejemplo del niño aprendiendo a andar en bicicleta, la política es el conjunto de reglas que el niño sigue para decidir cuándo empujar los pedales, girar el manillar, o frenar, basándose en su experiencia previa y en las recompensas obtenidas (Jaeger & Geiger, 2023). Dado un proceso Markoviano y una política π, la recompensa esperada de seguir esta política en un estado s determinado se puede definir de la siguiente manera mediante la ecuación estado-valor $V_\pi(s)$:

$$V_\pi(s) = E_\pi\left[\sum_{t=0}^{\infty} \gamma^t R_{t+k+1} | S_t = s\right]$$

donde $\gamma \in [0, 1]$ es el factor de descuento, que determina la importancia de las recompensas futuras en comparación con las recompensas inmediatas. Un valor de γ cercano a 0 hace que el agente se enfoque más en las recompensas inmediatas, mientras que un valor cercano a 1 hace que las recompensas futuras tengan una mayor influencia en la decisión del agente.

El objetivo de un problema Markoviano es calcular una política óptima π^* tal que $V_{\pi^*}(s) \geq V_\pi(s)$ para cada política π y cada estado $s \in S$. Para incluir la información de la acción, se utiliza el valor q, representando el valor óptimo de cada par estado-acción mediante la ecuación acción-valor:

$$q_\pi(s, a) = E_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a\right]$$

La función de estado también se puede expresar usando el operador de Bellman. Este operador calcula el valor actual de un estado basándose en las recompensas inmediatas y los valores esperados de los futuros estados resultantes de las acciones tomadas. Lo mismo ocurre con la función acción-valor, el operador de Bellman proporciona una forma similar de descomponer el valor de tomar una acción en un estado dado:

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

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

Existen diversas técnicas para calcular una política óptima para un proceso de decisión de Markov. Estas técnicas se dividen en dos categorías principales: métodos basados en modelos y métodos libres de modelos. Los métodos basados en modelos requieren un conocimiento completo y previo del modelo, incluyendo las funciones de transición y de recompensa. Por otro lado, los métodos libres de modelos, o métodos de aprendizaje, desarrollan una política óptima a partir de las observaciones y recompensas recibidas durante la interacción con el entorno, sin necesidad de conocer el modelo por adelantado (Sutton & Barto, 2014; Yu et al., 2020; Li, 2018).

Por ejemplo, un método basado en modelo podría utilizarse en un entorno de cuadrícula donde el agente debe encontrar el camino más corto hasta un objetivo, como podemos ver en la *Figura 3*. En este caso, se conoce de antemano la disposición de la cuadrícula, las posibles acciones y sus consecuencias, así como las recompensas asociadas con cada estado y acción. Con esta información, el agente puede calcular una política óptima utilizando algoritmos de planificación dinámica, como el algoritmo de valor iterativo.

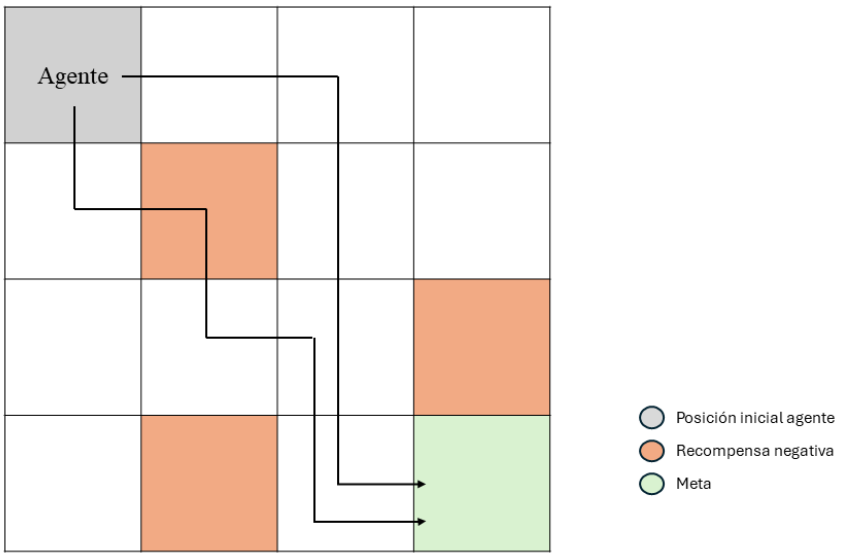


Figura 3. Entorno de cuadrícula en el que el agente tiene que encontrar el camino eficiente más corto.

En contraste, un ejemplo de un método sin modelo puede ser un robot autónomo que debe aprender a moverse en un entorno desconocido. Este robot no tiene conocimiento previo de la disposición del entorno ni de las recompensas asociadas con cada acción. En su lugar, debe explorar su entorno, recibir recompensas o castigos en función de sus acciones y aprender gradualmente una política óptima para moverse de manera eficiente. Algoritmos como *Q-Learning* o SARSA pueden ser utilizados por el robot para aprender de su experiencia y mejorar su comportamiento a lo largo del tiempo.

Además, se distinguen dos tipos algoritmos, los de predicción y los de control. Los algoritmos de predicción reciben como entrada la política $\pi(a|s)$ del agente, es decir, una guía predefinida que indica la probabilidad de ejecutar una cierta acción partiendo de un estado. La tarea de los algoritmos de predicción es calcular el retorno esperado, que es la

suma de todas las recompensas futuras que el agente recibirá siguiendo esa política. En contraste, los algoritmos de control no tienen una política predefinida como entrada. Su objetivo es mejorar continuamente la política del agente para maximizar las recompensas acumuladas a lo largo del tiempo, llegando así a la política optima π^*. Los métodos de control implican tanto la evaluación de la política actual como su mejora, balanceando la exploración de nuevas acciones y la explotación de las acciones que ya se ha determinado que son beneficiosas (Sutton & Barto, 2014; Wiering & van Otterlo, 2012).

Teniendo en cuenta estas dos distinciones podemos resumir en la *Tabla 1* los algoritmos de aprendizaje por refuerzo de esta manera:

	Predicción	*Control*
Basados en modelos	Programación dinámica: - Evaluación de la política	Programación dinámica: - Iteración de valor - Iteración de política
Libres de modelos	- Predicción con Montecarlo - Aprendizaje por diferencias temporales	- *Q-Learning* - SARSA - Montecarlo por control

Tabla 1. Distribución de los algoritmos de aprendizaje por refuerzo en función de si son basados en modelo, libres de modelo, de predicción o de control.

CAPÍTULO 3: Métodos basados en modelos

Los métodos basados en modelos son fundamentales para una toma de decisiones óptima en entornos dinámicos. Estos métodos se centran en la utilización de un modelo detallado del entorno para prever las consecuencias de las acciones y formular estrategias. Entre las técnicas más destacadas dentro de este enfoque se encuentran la programación dinámica, que ofrece un conjunto de herramientas potentes para resolver problemas de decisión secuencial. En particular, la evaluación de política, la iteración de valor y la iteración de política son procedimientos esenciales que permiten calcular políticas óptimas mediante la evaluación iterativa de estados y acciones.

3.1. Programación dinámica

La programación dinámica es un conjunto de algoritmos utilizados para calcular políticas óptimas dado un modelo perfecto del entorno, es decir, uno en el que se puede predecir con exactitud el resultado de cualquier acción en cualquier estado. Fue desarrollada por Richard Bellman en la década de 1950 como parte de su trabajo en procesos de decisión (Bellman, 1957). Aunque no es la técnica más utilizada actualmente, proporciona una base para entender otros métodos, ya que muchos de ellos intentan lograr efectos similares con menos computación y sin asumir un modelo perfecto del entorno. Como se trata de un método basado en modelo, asumimos que el entorno es finito, donde los conjuntos de estados, acciones y recompensas también son finitos, y su dinámica está dada por un conjunto de probabilidades.

Una forma común de obtener soluciones aproximadas es cuantificar los espacios de estados y acciones, para después aplicar métodos de programación dinámica de estado finito. La idea principal de la programación dinámica y del aprendizaje por refuerzo es el uso de funciones de valor v^* y q^*, las cuales satisfacen las ecuaciones de optimalidad de Bellman, para organizar y estructurar la búsqueda de buenas políticas. Los algoritmos de programación dinámica se obtienen convirtiendo estas ecuaciones en reglas de actualización para mejorar las aproximaciones de las funciones de valor deseadas.

Los algoritmos de programación dinámica se obtienen convirtiendo estas ecuaciones en reglas de actualización para mejorar las aproximaciones de las funciones de valor deseadas. Este proceso se basa en el concepto de *bootstrapping*, que en este contexto significa actualizar una estimación utilizando otras estimaciones actuales. En otras palabras, se utilizan las estimaciones de valor actuales para mejorar continuamente las aproximaciones de los valores futuros, iterando este proceso hasta alcanzar una convergencia (Yu et al., 2020).

Para comprender los algoritmos de predicción en el contexto de la programación dinámica, es esencial entender el concepto de evaluación de la política, explicado a continuación.

3.1.1. Evaluación de la política

La evaluación de políticas en programación dinámica, formalizada y popularizada por Richard S. Sutton y Andrew G. Barto en su libro *"Reinforcement Learning: An Introduction"* (Sutton & Barto, 2014), se refiere al proceso de calcular la función de valor del estado v_π bajo una política arbitraria π. Este procedimiento también se conoce como el problema de predicción. Recordemos que para todos los $s \in S$

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \left[r + \gamma v_\pi(s')\right]$$

donde $\pi(a \mid s)$ es la probabilidad de tomar la acción a en el estado s bajo la política π. Si la dinámica del entorno es completamente conocida, la ecuación de Bellman para v_π forma un sistema de ecuaciones lineales simultáneas en $\mid S \mid$ incógnitas, es decir, esto se refiere a tener un modelo perfecto del entorno, lo cual incluye conocer los estados S, las acciones A, la función de transición de estado $P(s' \mid s, a)$, y la función de recompensa $R(s, a, s')$. En la práctica, se utilizan métodos iterativos para aproximar la solución. Comenzamos con una función de valor inicial v_0 y actualizamos iterativamente usando la ecuación de Bellman:

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

Para producir cada aproximación sucesiva v_{k+1} a partir de v_k, la evaluación de políticas iterativa aplica la misma operación a cada estado s. Reemplaza el valor antiguo de s con un nuevo valor calculado a partir de los valores antiguos de los estados sucesores de s y las recompensas inmediatas esperadas en todas las posibles transiciones en un paso, bajo la política evaluada. Este tipo de operación se llama copia de seguridad completa. Cada iteración realiza una copia de seguridad del valor de cada estado una vez para producir la nueva función de valor aproximada v_{k+1}. Todas las copias de seguridad en los algoritmos de programación dinámica se denominan copias de seguridad completas porque se basan en todos los posibles estados siguientes, en lugar de un solo estado de muestra.

Para implementar la evaluación de políticas iterativa en un programa, se pueden usar dos arreglos, uno para los valores antiguos $v_k(s)$ y otro para los nuevos valores $v_{k+1}(s)$, así los nuevos valores pueden calcularse uno por uno a partir de los valores antiguos sin cambiar estos últimos. También se puede usar un solo arreglo y actualizar los valores de manera que cada nuevo valor sobrescribe inmediatamente al antiguo, lo que suele acelerar la convergencia al utilizar los datos nuevos tan pronto como están disponibles. La condición de parada típica se da cuando la diferencia máxima entre los valores antiguos y nuevos es menor que un umbral previamente definido.

Otra cuestión para considerar es la finalización del algoritmo. Teóricamente la evaluación de políticas iterativa converge solo en el límite, pero en la práctica debe detenerse antes

de esto. Una condición de parada típica para la evaluación de políticas iterativa es probar la cantidad $max_{s\in S} \mid v_{k+1}(s) - v_k(s) \mid$ después de cada iteración y detenerse cuando sea suficientemente pequeña. La *Figura 4* presenta un algoritmo completo para la evaluación de políticas iterativa con este criterio de parada.

Entrada π, la política a evaluar
Inicializar un arreglo $V(s) = 0$, para todos los $s \in S^+$
Repetir
    $\Delta \leftarrow 0$
    Para cada $s \in S$:
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) [r + \gamma V(s')]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
Hasta $\Delta < 0$ (número positivo pequeño)
Salida $V \approx v_\pi$

Figura 4. Iteración de evaluación de política

Un ejemplo práctico de evaluación de políticas iterativa podría involucrar un entorno de cuadrícula donde un agente se mueve en cuatro direcciones posibles con recompensas negativas hasta llegar a un estado terminal. Al seguir una política aleatoria equiprobable, la función de valor se actualizará iterativamente hasta que converge a una estimación estable de v_π(Sutton & Barto, 2014; Yu et al., 2020).

Mientras que la evaluación de políticas nos permite estimar la función de valor para una política dada, los algoritmos de control como la iteración de política e iteración de valor nos permiten mejorar y optimizar estas políticas. La iteración de política alterna entre evaluar y mejorar la política, mientras que la iteración de valor combina estos pasos en una sola actualización, facilitando una convergencia más rápida hacia la política óptima.

3.1.2. Iteración de política

La iteración de política tiene como pionero a Richard Bellman, quien lo desarrolló y formalizó en la década de 1950 como parte de su trabajo en programación dinámica en (Bellman, 1957). La iteración de política es un proceso para encontrar una política óptima π^* mejorando continuamente una política dada. Comenzamos con una política π y utilizamos su función de valor v_π, para obtener una política mejor. Luego podemos calcular de nuevo v_π pero en base a esta nueva política, mejorando nuevamente para obtener una política aún más óptima. Este proceso genera una secuencia de políticas y funciones de valor continuamente como podemos observar gráficamente en la *Figura 5* donde $E_{\rightarrow}$ representa una evaluación de política e $I_{\rightarrow}$ denota una mejora de política. Cada política es una mejora sobre la anterior, a menos que ya sea óptima. Dado que un proceso de decisión de Markov es finito y tiene solo un número finito de políticas, este proceso

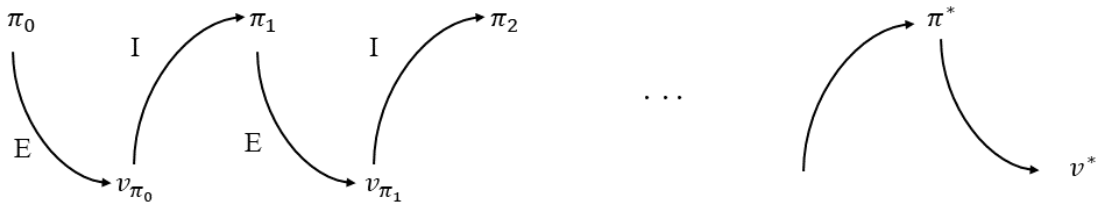


Figura 5. Proceso de iteración de política

debe converger a una política óptima y a una función de valor óptima en un número finito de iteraciones. Esta forma de encontrar una política óptima se llama iteración de política. Podemos observar un algoritmo de iteración de política en la *Figura 6*.

1. Inicio
   $V(s) \in \mathbb{R}$ y $\pi(s) \in A(s)$ de forma arbitraria para todas los $s \in S$

2. Evaluación de política
   Repetir
      $\Delta \leftarrow 0$
      Para cada $s \in S$:
      $v \leftarrow V(s)$
      $V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))[r + \gamma V(s')]$
      $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   Hasta que $\Delta < 0$

3. Mejora de la política
   $Política\ estable \leftarrow Verdadero$
   Para cada $s \in S$:
      $a \leftarrow \pi(s)$
      $\pi(s) \leftarrow argmax_a \sum_{s',r} p(s', r|s, \pi(s))[r + \gamma V(s')]$
      Si $a \neq \pi(s)$, entonces $política\ estable \leftarrow falso$
   Si $política\ estable$, se para y se vuelve a V y π y si no, se vuelve al paso 2

Figura 6. Iteración de política utilizando evaluación de política iterativa para v.*

La iteración de política comienza con una política inicial aleatoria π y la actualiza iterativamente. Primero se calcula la función asociada v_π , siendo la evaluación de política o predicción y luego mejorando la política usando $\pi(s) = \underset{a \in A}{\text{argmax}}\, v(s)$, siendo la mejora de política o control. Cada evaluación de política comienza con la función de valor de la política anterior, lo que acelera la convergencia de la evaluación de políticas debido a que la función de valor cambia poco de una política a otra (Sutton & Barto, 2014).

3.1.3. Iteración de valor

Un inconveniente de la iteración de políticas es que cada iteración implica la evaluación de la política, lo cual puede ser una computación iterativa prolongada. Si se hace iterativamente, la convergencia exacta a v_π solo ocurre en el límite. Sin embargo, es posible truncar la evaluación de la política sin perder las garantías de convergencia. Un caso especial es detener la evaluación de la política después de solo una pasada. Este algoritmo se llama iteración de valor, también fue desarrollado por Richard Bellman en la década de 1950 en (Bellman, 1957) y se describe como:

$$v_{k+1}(s) = \max_a E[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s, A_t = a]$$

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

para todos $s \in S$. Para cualquier v_0 arbitrario, se puede demostrar que la secuencia v_k converge a v^* bajo las mismas condiciones que garantizan la existencia de v^*. En la *Figura 7* podemos observar un algoritmo de iteración de valor.

Iniciar una matriz V arbitraria

Repetir
  $\Delta \leftarrow 0$
  Para cada $s \in S$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
Hasta que $\Delta <$ 0 (un número pequeño positivo)

Salida de una política determinística π como
  $\pi(s) = argmax_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$

Figura 7. Algoritmo de iteración de valor.

La iteración de valor combina la mejora de la política y los pasos de evaluación de políticas truncadas en una sola operación de copia de seguridad. Esto se basa en la ecuación de optimalidad de Bellman y asegura que el algoritmo converge a una política óptima en un proceso de decisión de Markov finito. Las actualizaciones basadas en la ecuación anterior se conocen como copias de seguridad completas, ya que utilizan información de todos los posibles estados sucesores. La iteración de valor es flexible y las asignaciones a V pueden hacerse de manera asincrónica siempre que cada estado se actualice infinitamente a menudo.

Una dificultad aparente es el momento de detener el algoritmo de iteración de valor. Si la diferencia máxima entre dos funciones de valor sucesivas es menor que ϵ, el valor de la política codiciosa (aquella en la que se elige en cada estado la acción que maximiza la recompensa esperada según la función de valor actual) difiere de la función de valor de

la política óptima en no más de $\frac{2\epsilon}{1-\gamma}$. Esto proporciona un criterio de parada efectivo para el algoritmo. Además, la complejidad computacional por iteración es cuadrática en el número de estados y lineal en el número de acciones, siendo lineal en el número de estados y acciones si las probabilidades de transición son dispersas, es decir, si para cada par estado-acción (s, a) solo un número pequeño de transiciones posibles tiene una probabilidad no nula, significa que la mayoría de las transiciones tienen una probabilidad de cero. En el peor de los casos, el número de iteraciones necesarias para alcanzar la función de valor óptima crece rápidamente a medida que el factor de descuento se aproxima a 1, lo que ralentiza la tasa de convergencia (Sutton & Barto, 2014; Yu et al., 2020).

CAPITULO 4: Métodos libres de modelo

A pesar de las ventajas de los algoritmos basados en modelos, los algoritmos sin modelo son más utilizados en la actualidad debido a su flexibilidad y capacidad de adaptación en entornos complejos y desconocidos. Estos algoritmos no requieren un conocimiento previo detallado del entorno, permitiendo a los agentes aprender directamente de la interacción y adaptarse dinámicamente a cambios impredecibles. Esta característica es especialmente valiosa en situaciones donde no es práctico obtener un modelo preciso. A continuación, se explicarán algunos de los algoritmos sin modelo más utilizados, destacando su funcionamiento. En este tipo de métodos libres de modelos también existe una distinción entre los algoritmos que se utilizan para predicción y los que se emplean para control. Los algoritmos de predicción, como Monte-Carlo y diferencias temporales, se enfocan en estimar el valor esperado de las políticas dadas. Por otro lado, los algoritmos de control, como *Q-Learning*, SARSA y Monte-Carlo control, están diseñados para optimizar las políticas mediante la maximización de recompensas a largo plazo.

4.1. Algoritmos de predicción

4.1.1. Método de Montecarlo

Los métodos de Monte Carlo no dependen de un conocimiento completo del entorno, a diferencia de la programación dinámica, por lo que solo requieren de experiencia para poder llevar a cabo el proceso de aprendizaje. Este método, introducido inicialmente en los campos de la física y las matemáticas por científicos como Stanislaw Ulam y John von Neumann en la década de 1940, puede alcanzar un buen rendimiento al aprender de la experiencia con muy poca información previa sobre el entorno (Metropolis & Ulam, 1949). Puede alcanzar un buen rendimiento al aprender de la experiencia con muy poca información previa sobre el entorno. Al utilizar Monte Carlo, se promedian las recompensas para cada par estado-acción a lo largo de diferentes episodios. Por ejemplo, imagina un sistema de recomendaciones en una plataforma de streaming de películas. Cada vez que un usuario ve una película, la plataforma sugiere una lista de recomendaciones basada en las características de la película vista, como el género, los actores o el director. Cada una de las características de la película representa un estado, y la satisfacción del usuario (medida, por ejemplo, por la calificación que da a la película recomendada) es la recompensa. Al principio, las recomendaciones pueden no ser muy precisas, sin embargo, a medida que el usuario ve más películas y la plataforma recopila más datos sobre sus preferencias, las recomendaciones promedio deberían ajustarse mejor a los gustos del usuario, mejorando la satisfacción general. Este método explora cómo realizar estas estimaciones correctamente y cómo aprovechar esta información de manera efectiva. Asumiremos que el problema es episódico y que cada episodio termina independientemente de las acciones tomadas.

Consideraremos el uso de métodos de Montecarlo para estimar la función de valor del estado bajo una política dada π. La manera más sencilla de hacerlo es promediando el retorno de una política particular a partir de la experiencia. Sea $v_\pi(s)$ la función de valor del estado bajo la política π, recopilaremos episodios que pasan por s, llamando a cada

aparición del estado s, una visita a s. Existen dos tipos de estimaciones posibles, Montecarlo de primera visita y Montecarlo de cada una de las visitas. El Montecarlo de primera visita considera solo el retorno de la primera visita al estado s en todo el episodio, mientras que el de cada una de las visitas considera todas las visitas al estado s en el episodio. Estos métodos son similares, pero tienen algunas diferencias teóricas. En la *Figura 8* podemos ver un algoritmo que muestra cómo se calcula $v_\pi(s)$ usando la estimación de Montecarlo de primera visita. Convertir esta predicción a Montecarlo de cada visita es sencillo, tan solo hay que eliminar la verificación de la primera visita. Ambos métodos convergen a $v_\pi(s)$ si el número de visitas al estado s es infinito. A diferencia de la programación dinámica, Montecarlo no usa *bootstrapping*, es decir, el proceso de actualizar estimaciones basadas en otras estimaciones, permitiéndole estimar el valor del estado directamente de los retornos muestreados, dando como resultado un menor sesgo pero una mayor varianza.

Entrada: Inicializar una política π
Inicializar $V(s)$ para todos los estados
Inicializar una lista de devoluciones: Retornos (s) para todos los estados
Repetir
    Generar un episodio bajo π:
    $G \leftarrow 0$
    $t \leftarrow T - 1$

    Para cada $t \geq 0$
        $G \leftarrow \gamma G + R_{t+1}$
        Si $S_0, S_1, \ldots, S_{t-1}$ no tiene S_t entonces
            $Retornos(S_t).añadir(G)$
            $V(S_t) \leftarrow$media(Retornos(S_t))
        Fin si
        $t \leftarrow t - 1$
    Fin para:
Hasta convergencia

Figura 8. Algoritmo de predicción de Montecarlo a primera vista

La función de valor del estado solo es útil cuando se dispone de un modelo, ya que permite elegir la acción óptima para un estado dado, observando el promedio del valor del estado para una acción específica, como en programación dinámica. Sin un modelo, debemos estimar el valor de la acción del estado por separado. Ahora el objetivo es estimar $q_\pi(s, a)$, es decir, el retorno esperado en el estado s al realizar la acción a bajo la política π. Esto es similar a la estimación de la función de valor del estado, promediando el retorno en el estado s al tomar la acción a. El problema es que algunos estados pueden no ser visitados, resultando en un retorno cero. Para elegir la estrategia óptima, es necesario explorar todos los estados. Una solución simple es especificar directamente el par estado-acción inicial para cada episodio, asegurando que cada par estado-acción tenga una probabilidad mayor a cero de ser seleccionado. Esta suposición se denomina inicios exploratorios (Ding et al., 2020).

4.1.2. Método de diferencias temporales

El aprendizaje por diferencias temporales, cuyos orígenes se remontan a la década de 1980 por el trabajo de Richard S. Sutton, quien introdujo formalmente los conceptos fundamentales de las diferencias temporales en su artículo influyente "*Learning to predict by the methods of temporal differences*" (Sutton, 1988). Este tipo de aprendizaje integra conceptos de la programación dinámica y de Montecarlo. Al igual que en programación dinámica, este método emplea el *bootstrapping* en sus estimaciones. Además, como en Montecarlo, no necesita un conocimiento exhaustivo del entorno para funcionar, utilizando en su lugar un enfoque de optimización basado en muestreo.

El método de diferencias temporales emplea el error entre el valor objetivo y el valor estimado para aprender en diferentes intervalos de tiempo. La razón por la cual las diferencias temporales también emplean *bootstrapping* es que el objetivo se forma a partir del retorno observado y el valor estimado del estado siguiente. El método de diferencias temporales más básico realiza la actualización utilizando la siguiente fórmula:

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

Este método se denomina $DT(0)$ o *DT de un paso*, ya que solo considera un único paso hacia adelante. La diferencia temporal de N pasos puede desarrollarse fácilmente ajustando el valor objetivo para incluir recompensas descontadas en el futuro de N pasos y el valor estimado del estado en el $N - ésimo$ paso. En la actualización de Montecarlo, el valor objetivo es G_t, el cual solo se conoce al final de un episodio, mientras que en diferencias temporales el valor objetivo es $R_{t+1} + \gamma V(S_{t+1})$, que puede calcularse de forma iterativa paso a paso. En el *Figura 9*, se presenta un algoritmo que ilustra cómo utilizar $TD(0)$ para la evaluación de políticas (Ding et al., 2020).

Entrada: política π
Inicializar $V(s)$ y tamaño de paso $\alpha \in (0,1]$
para cada episodio hacer

    Inicializar S_0
    para cada paso S_t en el episodio actual hacer
        $A_t \leftarrow \pi(S_t)$
        $R_{t+1}, S_{t+1} \leftarrow$ Entorno (S_t, A_t)
        $V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$
    Fin para
Fin para

Figura 9. Algoritmo TD (0) para la evaluación de políticas

Para entender un poco mejor el método de diferencias temporales vamos a compararlo con los métodos de Montecarlo y programación dinámica.

4.1.3. Comparativa de diferencias temporales, programación dinámica y Montecarlo

La programación dinámica, Montecarlo y diferencias temporales son esenciales en el aprendizaje por refuerzo y la combinación de cada uno de ellos es muy común en la actualidad. Aunque los 3 métodos se utilicen para evaluar y mejorar políticas, sus diferencias puedes resultar en variaciones significativas en el rendimiento del aprendizaje por refuerzo profundo, que explicaremos más adelante.

Los tres métodos emplean formas de iteración general de políticas. Tanto la programación dinámica como diferencias temporales utilizan *bootstrapping* para la actualización de estimaciones, mientras que Montecarlo no lo hace. Programación dinámica requiere un conocimiento completo del modelo del entorno, a diferencia de Montecarlo y diferencias temporales, que no necesitan esta información. En cuanto a la evaluación de políticas, la función de valor del estado $v_\pi(s)$ se puede estimar de diferentes maneras según el método utilizado. En el caso de Montecarlo la estimación se basa en la expectativa del retorno total G_t después de múltiples episodios:

$$v_\pi(s) = E_\pi[G_t|S_t = s] = E_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s]$$

Diferencias temporales combina el muestreo de Montecarlo y el *bootstrapping* de programación dinámica, utilizando la formula

$$v_\pi(s) = E_\pi[R_{t+1} + \gamma v_\pi S_{t+1}|S_t = s]$$

Algunas ventajas de las diferencias temporales respecto a programación dinámica y Montecarlo, es que esta primera no requiere modelo para aprender. Además, puede aprender en cada paso de tiempo, mientras que Montecarlo debe esperar hasta el final de cada episodio, lo que puede ser complicado de manejar. Esto hace que diferencias temporales sea más adecuado para tareas con episodios muy largos o continuos. Por último, tanto Montecarlo como diferencias temporales son métodos que convergen a $v_\pi(s), y$ aunque no se ha demostrado de manera formal, los métodos de diferencias temporales suelen hacerlo más rápido.

Es importante comprender la compensación entre varianza y sesgo en los métodos de diferencias temporales y Montecarlo. Un sesgo alto indica que el modelo no se ajusta bien a la distribución de datos. En la estimación del valor del estado, el sesgo se puede definir como $E[V(S_t)] - V(S_t)$. La varianza describe la variabilidad de las estimaciones. Para la estimación del valor del estado la varianza se puede expresar como $E[(E[V\ (S_t)] - V\ (S_t))^2]$.

En cuanto al manejo del valor objetivo, los métodos de Montecarlo estiman directamente las recompensas acumulativas hasta el final de un episodio, lo que elimina el sesgo, pero puede resultar en una alta varianza debido a la variabilidad de los episodios. Los métodos de diferencias temporales utilizan *bootstrapping*, calculando el valor objetivo de manera

local y dependiente del estado y la recompensa actuales, lo que tiende a reducir la varianza (Ding et al., 2020).

4.2. Algoritmos de control

Hasta ahora, hemos tratado los algoritmos de predicción libres de modelos, como los métodos de Monte Carlo y las diferencias temporales, que se centran en estimar el valor esperado de las políticas dadas. Estos algoritmos son fundamentales para comprender y evaluar cómo se comportan las políticas en distintos estados. Sin embargo, en el aprendizaje por refuerzo, no solo nos interesa predecir los valores de los estados bajo una política fija, sino también optimizar las políticas para maximizar las recompensas acumuladas. Para este propósito, necesitamos introducir los algoritmos de control libres de modelos, como *Q-Learning*, SARSA y Monte Carlo por control. Estos algoritmos no solo estiman el valor de las acciones, sino que también ajustan las políticas para encontrar la estrategia óptima. A continuación, veremos cómo estos algoritmos de control trabajan para mejorar continuamente las políticas a través de la interacción con el entorno.

4.2.1. Algoritmo SARSA

Podemos considerar que el algoritmo SARSA, desarrollado por Richard S. Sutton y Andrew G. Barto en su libro "*Reinforcement Learning: An Introduction*" (Sutton & Barto, 2014), es un método dedicado al control de diferencias temporales. Es un algoritmo en política, es decir, se usa la misma política tanto para tomar acciones como para actualizar la función de valor. Seguimos una metodología similar a la tarea de predicción, pero en lugar de alternar de estado a estado, alternamos entre pares estado-acción. La regla de actualización se formula como:

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

donde $\alpha \in [0,1]$ es la tasa de aprendizaje, parámetro que controla cuánto se ajusta el valor q actual. Cuanto más alto es este número significa que los valores de q se ajustarán más rápido, dando más peso a las nuevas experiencias. Si este número es bajo estos valores se ajustarán más lentamente confiando más en estimaciones anteriores. Es importante recordar la labor del factor de descuento $\gamma \in [0,1]$ en el cual un valor cercano a 1 significa que se valoran mucho las recompensas futuras, mientras que cerca de 0 se valorarán las inmediatas. Esta actualización se realiza después de cada transición desde un estado no terminal S_t. Si S_{t+1} es terminal, entonces $Q(S_{t+1}, A_{t+1})$ se define como cero. El nombre SARSA proviene de la secuencia estado-acción-recompensa-estado-acción $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$, siguiendo sus siglas en inglés *state-action-reward-state-action*. Esto refleja el proceso de estar en un estado, tomar una acción, recibir una recompensa y pasar a un nuevo estado para tomar otra acción. Este proceso permite realizar un paso de actualización simple y continuo. El valor del estado se actualiza en cada transición y este valor actualizado influye en la política que se usa para determinar el comportamiento, haciéndolo un método en política. En la *Figura 10* se muestra el desarrollo de un algoritmo SARSA.

Inicializar $Q(s, a)$ para todos los pares estado-acción.

Para cada episodio, hacer:
Inicializar S_0
Seleccionar A_0 usando una política basada en Q

Para cada paso S_t en el episodio actual, hacer:
Seleccionar A_t desde S_t usando una política basada en Q
$R_{t+1}, S_{t+1} \leftarrow$ Entorno (S_t, A_t)
Seleccionar A_{t+1} desde S_{t+1} usando una política basada en Q
Actualizar $Q(S_t, A_t)$ usando la ecuación de actualización:
$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$
Incrementar $t \leftarrow t + 1$

Fin para

Fin para

Figura 10. Algoritmo SARSA

El algoritmo SARSA por tanto estima la función de valor-acción $q_\pi(s, a)$ para la política de comportamiento actual π y para todos los estados s y acciones a. A diferencia de los métodos fuera de política como *Q-Learning*, que usa una política diferente para la actualización de los valores de Q, SARSA sigue una política de comportamiento que es la misma que la política de actualización. Esto significa que, en SARSA, tanto la acción que se lleva a cabo en el entorno como la acción usada para actualizar los valores de Q son seleccionadas de acuerdo con la misma política. Esto permite una coherencia entre la exploración durante el aprendizaje y la actualización de los valores, resultando en un aprendizaje más alineado con las acciones que el agente está efectivamente tomando. La convergencia a una política y función de valor óptima se garantiza siempre y cuando cada par estado-acción sea visitado un gran número de veces, tendiendo al infinito.

Para mejorar la precisión de la estimación del valor de Q y acelerar la convergencia podemos extender el método a *SARSA de n pasos*, que considera múltiples pasos en el futuro. La actualización para *SARSA de n pasos* se realiza como:

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n Q_{t+n-1}(S_{t+n}, A_{t+n})$$

En la *Figura 11* se muestra el algoritmo *SARSA de n pasos*.

Inicializar $Q(s,a)$ para todos los pares estado-acción.
Inicializar el tamaño del paso $\alpha \in (0,1]$.
Determinar una política π
Para cada episodio, hacer:
    Inicializar S_0
    Seleccionar A_0 usando $\pi(S_0, A)$
    $T \leftarrow$ Valor máximo de la longitud del episodio
    $\gamma \leftarrow 0$
    Para $t \leftarrow 0,1,2,\ldots$ hasta $\gamma - T$, hacer:
        Si $t < T$ entonces
            $R_{t+1}, S_{t+1} \leftarrow$ Entorno (S_t, A_t)
            Si S_{t+1} es terminal, entonces
                $T \leftarrow t + 1$
            De lo contrario:
                Seleccionar A_{t+1} usando $\pi(S_t, A)$

        $\tau \leftarrow t - n + 1$ (el paso de tiempo para actualizar)
        Si $\tau \geq 0$ entonces:
            $G \leftarrow \sum_{i=r+1}^{\min(r+n,T} \gamma^{i-\gamma-1} R_i$
            Si $\gamma + n < T$ entonces:
                $G \leftarrow G + \gamma^n Q(S_{t+n}, A_{t+n})$
                Fin si
                $Q(S_\gamma, A_\gamma) \leftarrow Q(S_\gamma, A_\gamma) + \alpha[G - Q(S_\gamma, A_\gamma)]$
        Fin si
    Fin para
Fin para

Figura 11. Algoritmo SARSA de n pasos

Para garantizar la convergencia a una política óptima, debemos cumplir con ciertas condiciones. La política de aprendizaje debe ser óptima en el límite con exploración infinita (GLIE), siglas en ingles de *Greedy in the Limit with Infinite Exploration*. Esto significa que, si un estado se visita infinitamente, entonces cada posible acción en ese estado se elige infinitamente.

$$\lim_{k\to\infty} N_k(s,a) = \infty \, \forall a \; si \; \lim_{k\to\infty} N_k(s) = \infty$$

Además, la política converge a una política óptima con respecto a la función Q aprendida en el límite. Una política óptima es aquella en la que, en cada estado, se elige la acción que maximiza la función de valor Q actual, es decir, se selecciona la acción con la mayor recompensa esperada basada en las estimaciones actuales de Q.

$$\lim_{k\to\infty} \pi_k(s,a) = 1\left(a == arg\max_{a' \in A} Q_k(s,a')\right)$$

donde " == " es un operador de comparación y $1(a == b)$ es 1 si es verdadero y 0 si es falso. Una política $\varepsilon - codiciosa$ es GLIE si ε se reduce a cero con $\varepsilon_k = \frac{1}{k}$.

Por tanto, el teorema de convergencia para SARSA quedaría así: "Para un proceso Markoviano de estado-acción finito y una política de aprendizaje GLIE, con la función de valor-acción Q estimada con SARSA, Q_t converge a Q^* y la política de aprendizaje π_t converge a una política óptima π^*, si se satisfacen las siguientes condiciones:

1ª condición: Los valores Q se almacenan en una tabla de búsqueda.

2ª condición: La tasa de aprendizaje $\alpha_t(s,a)$ cumpla con:

$$0 \leq \alpha_t(s,a) \leq 1$$

$$\sum_t \alpha_t(s,a) = \infty \text{ y } \sum_t \alpha_t^2(s,a) < \infty$$

$$\alpha_t(s,a) = 0 \text{ a menos que } (s,a) = (S_t, A_t)$$

3ª condición: La varianza de $R(s,a)$ tiene que ser finita (Al-Hamadani et al., 2024; Ding et al., 2020; Sutton & Barto, 2014).

4.2.2. Algoritmo *Q-Learning*

Q-Learning, desarrollado por Chris Watkins en (Watkins & Dayan, 1992), es un algoritmo de diferencias temporales fuera de la política, es decir, se pueden usar diferentes políticas para tomar acciones y para actualizar la función de valor. Su característica principal es la estimación de la función de valor de acción q, la cual aproxima directamente la función de valor de acción óptima q^*, independientemente de la política que se esté ejecutando. Se basa en la siguiente fórmula:

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

donde las actualizaciones de *Q-Learning* utilizan solo los cuatro elementos (S_t, A_t , R_{t+1} , S_{t+1}) mientras asume que S_{t+1} es una variable de decisión para optimizar la función de valor de acción. La implementación de este algoritmo se muestra en la *Figura 12*.

Inicializar $Q(s,a)$ para todos los $s \in S$, $a \in A(s)$, arbitrariamente, y $Q(ultimo\ estado, \cdot) = 0$

Repetir (para cada episodio):
  Inicializar S
  Repetir (para cada paso del episodio):
    Elegir A desde S usando la política derivada de Q
    Tomar acción A, observar R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha [R + \gamma \max_a Q(S',a) - Q(S,A)]$
    $S \leftarrow S'$ hasta que S sea terminal

Figura 12. Implementación de algoritmo Q-Learning

A diferencia de SARSA, que es un método de diferencias temporales en política, *Q-Learning* actualiza los valores de acción de manera independiente de la política seguida durante el aprendizaje. Esto significa que el valor objetivo en *Q-Learning* no depende de la política actual, sino solo de la función de acción del estado. La convergencia de *Q-Learning* sigue condiciones similares a las del algoritmo SARSA. Además de la condición GLIE para la política, la convergencia de la función Q en *Q-Learning* requiere una tasa de aprendizaje adecuada y valores de recompensa acotados.

En *Q-Learning*, el diagrama de actualización muestra cómo se actualizan los valores de las acciones en cada paso. El nodo superior del diagrama representa el par estado-acción (S_t, A_t) que se está actualizando. Los nodos inferiores representan todas las posibles acciones en el siguiente estado S_{t+1}. *Q-Learning* actualiza el valor del par estado-acción actual utilizando la recompensa inmediata más el valor máximo de todas las acciones posibles en el siguiente estado. Un arco que cruza estos nodos inferiores indica que se selecciona el mayor valor. Esto ayuda a que el algoritmo aprenda a elegir las mejores acciones futuras para maximizar las recompensas a largo plazo. Podemos observar un diagrama de respaldo de *Q-Learning* en la *Figura 13* (Al-Hamadani et al., 2024; Ding et al., 2020; Sutton & Barto, 2014).

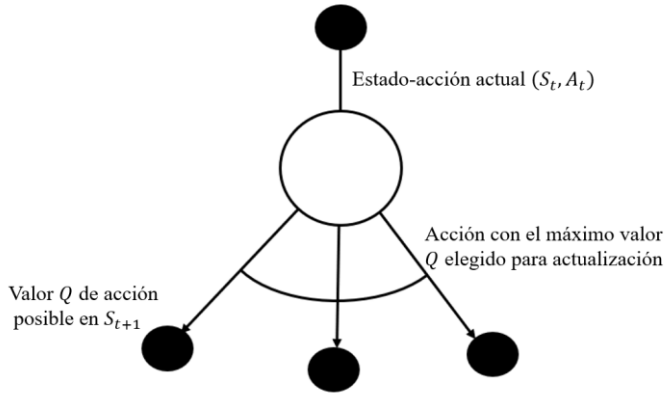


Figura 13. Diagrama de actualización en Q-Learning

4.2.3. Método de Montecarlo por control

Vamos a ver el funcionamiento de Montecarlo cuando se trata de problemas de control, es decir, para la aproximación de políticas óptimas. Este algoritmo fue popularizado por Richard S. Sutton y Andrew G. Barto en su libro "*Reinforcement Learning: An Introduction*" (Sutton & Barto, 2014). La manera de actuar es siguiendo el mismo patrón que en programación dinámica, siguiendo la iteración de política general. En la iteración de política general se mantiene tanto una política aproximada como una función de valor aproximada. Utilizaremos una política codiciosa con respecto a la función del valor de acción, por lo tanto, se seleccionará siempre la acción que tenga el valor más alto para un estado determinado de manera que

$$\pi(s) = arg\max_{a} q(s, a)$$

Ahora nuestra labor es mejorar la política, y para cada una de estas mejoras de políticas se necesita construir π_{t+1} basándose en π_t. Así es como se aplica el teorema de mejora de políticas:

$$q_{\pi_t}\big(s, \pi_{t+1}(s)\big) = q_{\pi_t}(s, arg\max_a q_{\pi_t}(s,a))$$
$$= \max_a q_{\pi_t}(s,a)) \geq q_{\pi_t}(s, \pi_t(s) \geq v_{\pi_t}(s)$$

Este teorema nos asegura que π_{t+1} será mejor que π_t o al menos igual de buena, en cuyo caso ambas políticas serían óptimas. Esto nos asegura que el proceso general converge hacia la política óptima y la función de valor óptima. Por tanto, los métodos Montecarlo pueden ser utilizados para encontrar políticas óptimas, dados únicamente episodios de muestra y sin ningún otro conocimiento sobre la dinámica del entorno. Hay dos suposiciones que debemos resolver.

La primera es que los episodios tienen inicios de exploración, mientras que la otra es que la evaluación de política podría hacerse con infinitos episodios. Para resolver este segundo supuesto hay dos formas de resolver el problema. Una es mantener la idea de aproximar q_{π_k} en cada evaluación de política. Se realizan mediciones y suposiciones para obtener límites sobre la magnitud y la probabilidad de error en las estimaciones, y luego se toman suficientes pasos durante cada evaluación de política para asegurar que estos límites sean suficientemente pequeños. Este enfoque probablemente pueda satisfacer de manera completa con la intención de garantizar una convergencia correcta hasta cierto nivel de aproximación. Sin embargo, también es probable que requiera demasiados episodios para ser útil en la práctica, excepto en los problemas más pequeños. El segundo enfoque para evitar el número infinito de episodios requerido para la evaluación de políticas es renunciar a intentar completar la evaluación de políticas antes de volver a la mejora de políticas. En cada paso de evaluación ajustamos la función de valor hacía q_{π_k}, pero no esperamos acercarnos realmente, excepto a lo largo de muchos pasos.

Inicializa π para todos los estados
Inicializa $Q(s,a)$ y $Devuelve(s,a)$ para todos los pares estado-acción
Repetir:
  Selecciona aleatoriamente S_0 y A_0 de manera que todas las probabilidades de los pares estado-acción sean diferentes de cero.
  Genera un episodio desde S_0, A_0 bajo $\pi : S_0, A_0, R_0, S_1, \ldots, S_{T-1}, A_{T-1}, R_t$
  $(G \leftarrow 0)$
  $t \leftarrow T - 1$
  para $t \geq 0$ hacer:
    $G \leftarrow G + R_{t+1}$
    si $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$ no contiene S_t, A_t entonces
      $(Devuelve(S_t, A_t)).append(G)$
      $Q(S_t, A_t) \leftarrow media(Devuelve(S_t, A_t))$
      $\pi(S_t) \leftarrow arg\max_a Q(S_t, a))$
    fin si
    $t \leftarrow t - 1$
  fin para
hasta la convergencia

Figura 14. Montecarlo con inicios de exploración

En la evaluación de políticas con Montecarlo, es lógico alternar entre las etapas de evaluación y mejora después de cada episodio. Tras cada episodio, se utilizan los retornos observados para evaluar la política, y luego se mejora la política en todos los estados que se visitaron durante el episodio. Un algoritmo que sigue este enfoque es Montecarlo con inicios de exploración y se muestra a continuación en la *Figura 14* (Ding et al., 2020; Sutton & Barto, 2014).

CAPÍTULO 5: Problemas comunes

El aprendizaje por refuerzo se enfrenta a una serie de desafíos que pueden limitar su efectividad y aplicabilidad en entornos difíciles de manejar. Uno de los problemas más comunes en este tipo de aprendizajes es encontrar un balance entre la exploración y la explotación, y el problema de la alta dimensionalidad. A continuación, se explicará en qué consisten cada uno de ellos y que posibles soluciones podemos llevar a cabo.

5.1. Dilema de Exploración/Explotación

El dilema de exploración-explotación se basa en encontrar el equilibrio adecuado entre dos estrategias, la exploración y explotación. La exploración implica obtener información sobre el entorno con el objetivo de descubrir nuevas posibilidades y aprender más sobre el entorno. Por otro lado, la explotación se enfoca en maximizar el retorno esperado utilizando el conocimiento actual, lo que implica seguir la estrategia que, según la experiencia acumulada, parece ser la que promete mejores resultados. Un agente debe equilibrar entre aprender más sobre el entorno, es decir, explorar, y aplicar la estrategia más efectiva basada en la experiencia, explotar.

Existen diferentes formas de abordar el dilema de exploración-explotación. En la primera el agente debe rendir bien sin una fase de entrenamiento separada. En este caso, el agente explora solo cuando las oportunidades de aprendizaje son valiosas para el futuro, compensando así lo que la explotación directa puede proporcionar. La configuración más común es aquella en la que el agente sigue una política de entrenamiento durante una fase inicial de interacciones para acumular datos y aprender una política de prueba. Durante la fase de entrenamiento, la exploración está limitada por las interacciones con el entorno, mientras que, en la fase de prueba, la política aprendida se utiliza para maximizar una suma acumulativa de recompensas en una fase de interacción separada. En este tipo de configuración, la exploración y explotación implícita juegan un papel crucial. El agente debe asegurarse de que las partes menos conocidas del entorno no sean prometedoras, es decir, partes del entorno que tienen poco potencial de recompensa o aprendizaje y, al mismo tiempo, acumular experiencia en las partes más prometedoras del entorno para refinar su conocimiento de las dinámicas (François-Lavet et al., 2018).

5.1.1. Enfoques para la exploración

Las técnicas de exploración en el aprendizaje por refuerzo se dividen en dos categorías principales, exploración no dirigida y exploración dirigida. La exploración no dirigida no se basa en el conocimiento específico del entorno. Algunos ejemplos son $\varepsilon - greedy$ y *Softmax*. En la técnica $\varepsilon - greedy$, las acciones se toman de manera aleatoria con una probabilidad ε y se sigue la política óptima con una probabilidad $1 - \varepsilon$. Este enfoque permite al agente explorar nuevas acciones, mientras que se va reduciendo poco a poco el valor de ε. Esta reducción permite al agente transitar de un enfoque exploratorio a uno más enfocado en la explotación de las estrategias aprendidas que han demostrado ser más efectivas, optimizando así las recompensas acumuladas. Por otro lado, en el método

Softmax, las acciones se toman con una probabilidad basada en el retorno esperado asociado a cada acción. Las acciones con mayores retornos esperados tienen más probabilidades de ser seleccionadas, lo que equilibra la exploración y la explotación de manera más suave que *ε-greedy*.

Por otro lado, la exploración dirigida se basa en el uso de la memoria de interacciones pasadas con el entorno. Esto significa que el agente recuerda y utiliza información de experiencias anteriores para guiar sus futuras decisiones de exploración. Entre las técnicas de exploración dirigida se encuentran los métodos de bonificación de exploración, que utilizan medidas como la maximización de las ganancias de información para dirigir la exploración. Estas técnicas proporcionan bonificaciones por visitar estados menos explorados, incentivando al agente a descubrir nuevas áreas del espacio de estados. Sin embargo, la exploración dirigida es más desafiante en espacios de estados de alta dimensión. En el caso del aprendizaje por refuerzo profundo, la exploración adopta métodos avanzados para mejorar la eficiencia y eficacia, como por ejemplo la exploración en *Deep Q-Network*, donde el esquema *ε-greedy* proporciona una exploración relativamente eficiente. También se utilizan recompensas de exploración basadas en la novedad, otorgando recompensas adicionales por visitar estados nuevos, de manera que estiman la novedad de los estados y motivan al agente a seguir explorando continuamente, asegurándose así un aprendizaje más exhaustivo y efectivo del entorno (François-Lavet et al., 2018).

5.2. Problema de la dimensionalidad

El problema de la alta dimensionalidad es un desafío que surge cuando el espacio de estados o de acciones tiene un número muy grande de dimensiones, dificultando la representación, el almacenamiento y la exploración eficiente del espacio. Esto puede provocar que el aprendizaje sea lento o incluso impracticable. Este problema es muy común en entornos como los videojuegos, control de robots o sistemas de recomendaciones. En el ámbito de los videojuegos, la alta dimensionalidad se aborda mediante técnicas avanzadas de aprendizaje profundo por refuerzo, como se demostró en la investigación por (Mnih et al., 2015), donde se logró un control a nivel humano en videojuegos Atari utilizando redes neuronales profundas. Para el control de robots, los desafíos de alta dimensionalidad se exploran en la revisión de (Kober et al., 2013),que describe el uso del aprendizaje por refuerzo en robótica y cómo las técnicas específicas pueden ayudar a manejar espacios de estado complejos. En cuanto a los sistemas de recomendaciones, (Covington et al., 2016)discuten cómo YouTube implementa redes neuronales profundas para manejar la alta dimensionalidad de los datos de usuarios y contenido, lo que mejora significativamente la eficiencia y precisión de sus recomendaciones. Esta condición puede provocar una serie de consecuencias negativas, como la imposibilidad de hacer una exploración completa, necesitar altos requisitos computacionales, la presencia de una convergencia lenta o una generalización limitada.

Cuando hay un gran número de estados y acciones, es muy difícil para un agente explorar correctamente el espacio para aprender las políticas óptimas, haciendo imposible cubrir todas las posibles combinaciones de estados y acciones de manera eficiente. El almacenamiento y procesamiento de un elevado número de estados y transiciones requiere una enorme cantidad de recursos computacionales, por lo que la memoria necesaria para almacenar las tablas Q y los modelos del entorno, así como la capacidad

de procesamiento para actualizar estas tablas y modelos, crece de manera exponencial con el número de dimensiones. Una alta dimensionalidad también puede provocar una convergencia extremadamente lenta, ya que el agente necesita visitar suficientes estados para aprender valores precisos. Esto significa que el agente puede necesitar una cantidad demasiado grande de iteraciones para cubrir adecuadamente el espacio y poder ajustar sus políticas. Por último, la capacidad del agente para generalizar el aprendizaje a estados no visitados previamente puede ser limitada. En un espacio de alta dimensión, las diferencias entre los estados pueden ser tan grandes que las técnicas tradicionales de generalización no funcionan bien, dificultando el aprendizaje efectivo y eficiente.

Una posible solución para el manejo de la alta dimensionalidad puede ser la utilización del aprendizaje por refuerzo profundo, proporcionando una representación más compacta y eficiente del espacio de estados y de acciones, permitiendo a los agentes aprender y tomar decisiones en entornos complejos, siendo imposible de hacerlo de otra manera. Una de las técnicas más importantes en aprendizaje por refuerzo profundo es la aproximación de funciones mediante las redes neuronales profundas. Esto permite a los agentes aprender representaciones más fáciles de manejar. Vamos a dividir los métodos en dos categorías, los métodos basados en valor como la red Q profunda, en inglés conocida como *Deep Q-Network*, o métodos basados en política como el algoritmo de Optimización de políticas optimas, en ingles conocido como *Proximal Policy Optimization*. El algoritmo *Deep Q-Network* utiliza redes neuronales convolucionales para aproximar las funciones de valor Q. Este tipo de redes son muy efectivas para el manejo de entradas visuales de alta dimensionalidad, como las imágenes en los videojuegos. Además, fue el primer algoritmo en demostrar que un agente puede aprender políticas complejas directamente de los píxeles, llevando a cabo tareas que piden una percepción visual muy detallada. Por otro lado, los métodos basados en políticas representan la política directamente, entrenándose para maximizar el retorno esperado. Por tanto, algoritmos como *Proximal Policy Optimization* se encargan utilizar las redes neuronales profundas para representar políticas, ajustando continuamente los parámetros para mejorar el rendimiento del agente (Mnih et al., 2015; Schulman et al., 2017; Sutton & Barto, 2014).

CAPÍTULO 6: Aprendizaje por refuerzo profundo

El aprendizaje profundo surge de la investigación en redes neuronales artificiales, diseñadas para imitar el comportamiento de las neuronas del cerebro. Estas redes consisten en capas de neuronas interconectadas que procesan información ajustando las conexiones entre nodos internos. El aprendizaje profundo utiliza redes neuronales multicapa, conocidas como perceptrones multicapa, que incluyen múltiples capas ocultas. Esta estructura permite manejar problemas complejos al ajustar conexiones entre numerosos nodos internos, facilitando el procesamiento avanzado de información. En este tipo de aprendizaje las características de bajo nivel, las cuales se refieren a las representaciones más simples y directas derivadas directamente de los datos de entrada, se combinan para formar representaciones de alto nivel más abstractas, permitiendo al sistema descubrir características complejas y mejorar la comprensión y clasificación de datos. Aunque el aprendizaje por refuerzo y el aprendizaje profundo son algoritmos distintos, pueden combinarse para formar el aprendizaje por refuerzo profundo, en el cual se utiliza el aprendizaje profundo para aproximar funciones de valor o políticas, permitiendo que el aprendizaje por refuerzo maneje problemas con grandes espacios de estados y acciones. El aprendizaje por refuerzo define el objetivo de optimización, que es maximizar la recompensa, mientras que el aprendizaje profundo proporciona el mecanismo operativo de redes neuronales para caracterizar y resolver el problema (Tan et al., 2017).

Una red neuronal está compuesta por numerosas neuronas artificiales, modeladas según el funcionamiento de las neuronas biológicas. Al igual que una neurona biológica recibe y procesa señales de múltiples fuentes antes de generar una salida, una neurona artificial procesa entradas y produce una salida utilizada para activar otras neuronas en la red. Las neuronas artificiales se modelan para procesar datos de entrada y producir salidas. Las entradas $x_1, x_2, \dots, x_n$ se representan como un vector de entrada X. Estas entradas pueden representar datos en bruto que a menudo necesitan ser escalados o normalizados para un funcionamiento efectivo. Cada entrada se pondera con un peso correspondiente $w_1, w_2, \dots, w_n$, formando un vector de pesos W. El objetivo es entrenar estos pesos W para minimizar el error entre los valores predichos y los valores reales. Además de los pesos, la neurona incluye un sesgo b, un número real que se suma a la entrada ponderada. El procesamiento dentro de la neurona se realiza a través de una función de activación. La entrada a esta función, Z, se calcula como la suma ponderada de las entradas más el sesgo $Z = W^T X + b$. La salida de la neurona $\hat{y}$, también conocida como los valores predichos, se obtiene aplicando la función de activación a Z, es decir, $\hat{y} = Activación(Z)$. La elección de la función de activación depende del tipo de predicción deseada entre categórica o continua. En la *Figura 15* podemos ver el funcionamiento de una neurona artificial (Sewak, 2019).

6.1. Entrenamiento de una neurona artificial

El entrenamiento de una neurona artificial es un proceso iterativo que implica varias etapas clave para garantizar que la red neuronal pueda hacer predicciones precisas.

Primero, se realiza la pasada hacia delante, etapa en la que un único elemento de datos se introduce en la red neuronal, y las entradas son procesadas a través de la red para calcular la salida predicha $\hat{y}$. La neurona combina las entradas ponderadas y el sesgo a través de su función de activación para producir una salida predicha. Esta salida es comparada con la salida real para evaluar el rendimiento de la red. Para entrenar la red de manera efectiva, se necesita procesar múltiples elementos de datos. El ajuste de pesos permite ajustar los

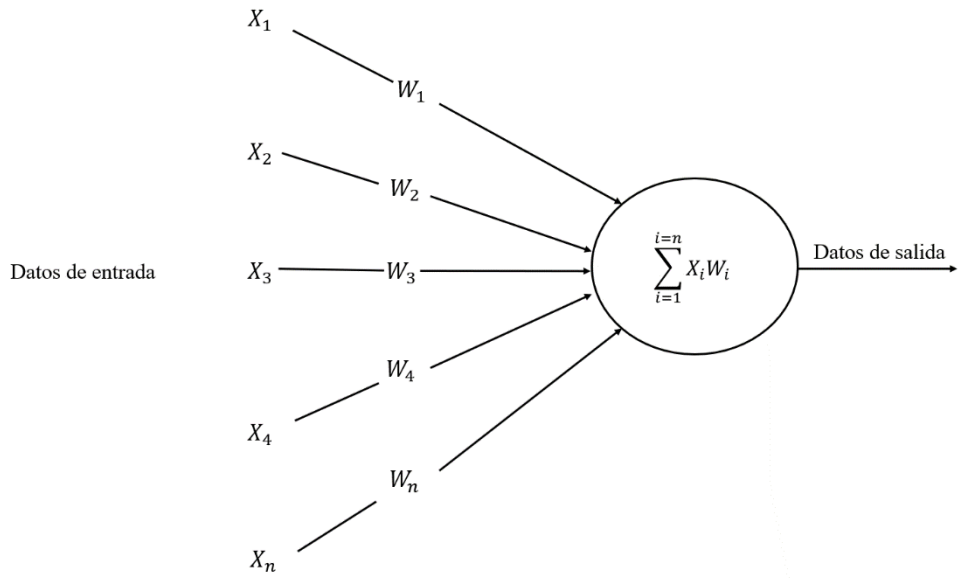


Figura 15. Proceso de funcionamiento de una neurona artificial

pesos de la red a través de muchas iteraciones, lo que mejora la precisión de las predicciones. Trabajar con un conjunto variado de datos ayuda a la red a generalizar mejor, evitando el sobreajuste a datos específicos y mejorando su desempeño en datos no vistos. La pérdida o error se calcula comparando la salida real y con la salida predicha $\hat{y}$ mediante una función de pérdida. Esta función mide como de lejos está la predicción de la neurona de la realidad. Existen diversas funciones de pérdida, como el error cuadrático medio

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

donde y_i son los valores reales y $\hat{y}_i$ son los valores predichos por la red neuronal. Otra función de pérdida puede ser la entropía cruzada

$$-\sum_{i=1}^{n} y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)$$

para problemas de clasificación binaria, donde y_i es la etiqueta real y $\hat{y}_i$ es la probabilidad predicha de que la muestra pertenezca a la clase positiva. La elección de cada una depende de la naturaleza de la aplicación y las características de los datos de entrada.

El objetivo del entrenamiento es minimizar la pérdida en todo el conjunto de datos de entrenamiento y evaluación. Esto se logra ajustando iterativamente los pesos y el sesgo de la neurona. Se utilizan algoritmos como el descenso de gradiente, que calculan las derivadas parciales de la función de pérdida con respecto a los pesos y el sesgo. La ecuación del descenso de gradiente se puede describir como:

$$\theta' = \theta - \eta \nabla \theta J(\theta)$$

donde θ representa los parámetros del modelo, es decir, los pesos y sesgos, $J(\theta)$ es la función de pérdida, $\nabla_\theta J(\theta)$ es el gradiente de la función de pérdida, y η es la tasa de aprendizaje. Estos algoritmos ajustan los pesos en la dirección que reduce la pérdida. El proceso de calcular la salida predicha, evaluar la pérdida y ajustar los pesos se repite muchas veces a lo largo del entrenamiento. Con suficientes iteraciones, el proceso de optimización converge, es decir, los ajustes en los pesos y el sesgo se vuelven muy pequeños y la pérdida se minimiza, indicando que la neurona ha aprendido a hacer predicciones precisas. Una vez que el entrenamiento está completo, la neurona está lista para realizar predicciones o estimaciones en nuevos datos no vistos. La red ahora puede generalizar y aplicar el conocimiento adquirido durante el entrenamiento para proporcionar salidas precisas en diversas situaciones (Sewak, 2019).

La mayoría de los trabajos de aprendizaje por refuerzo profundo se basan en enfoques sin modelo. En el caso de no tener modelo, se puede estimar el estado, la función de valor y la función de recompensa del agente a través de un gran número de muestras para optimizar la política de acción $\pi_0(a|s)$, que tiene como objetivo obtener más recompensas realizando la acción a en el estado s. Vamos a dividir el aprendizaje por refuerzo profundo principalmente en dos categorías, métodos basados en valor y métodos basados en políticas. Los métodos basados en valor, como por ejemplo la red Q profunda, en ingles *Deep Q-Networks*, se centran en aprender una función de valor que estima la recompensa futura esperada para cada estado o acción, permitiendo que el agente seleccione acciones que maximicen este valor. Por otro lado, los métodos basados en políticas, como la optimización de la política próxima, en ingles *Proximal Policy Optimazation*, se enfocan en aprender directamente una política que mapea estados a acciones, optimizando la selección de acciones para maximizar las recompensas esperadas a largo plazo. La elección entre estos dos enfoques depende del problema específico y las características del entorno en el que se aplica el aprendizaje por refuerzo profundo (Wang et al., 2020).

CAPÍTULO 7: Aplicación práctica

En este apartado se llevará a cabo un ejemplo práctico enfocado en un controlador de temperatura que gestiona la temperatura de una casa. El objetivo del agente es mantener la temperatura lo más cercana posible a una temperatura objetivo, en este caso de 22 grados centígrados, durante una serie de episodios. Utilizaremos un algoritmo de *Q-Learning* para que el agente aprenda una política óptima que le permita tomar las mejores acciones en función de la temperatura actual.

El algoritmo *Q-Learning* es especialmente adecuado para este tipo de aplicaciones debido a varias razones. En primer lugar, permite que el agente aprenda directamente de la experiencia sin necesitar un modelo previo del entorno, adaptándose a las dinámicas del entorno que pueden ser complejas o impredecibles en aplicaciones como el control de temperatura. Bajo condiciones ideales de aprendizaje, *Q-Learning* garantiza la convergencia a la política óptima, lo que es crítico para optimizar el control de la temperatura en la manera más eficiente. Utilizando una política *epsilon-greedy*, *Q-Learning* permite un balance efectivo entre explorar nuevas estrategias y explotar aquellas que han demostrado ser efectivas, esencial para el aprendizaje inicial y la mejora continua. Además, su implementación es relativamente simple y no requiere una intensa carga computacional, adecuada para aplicaciones en tiempo real como el control de temperatura doméstico.

7.1. Descripción del problema

El entorno en este problema es la casa con su sistema de control de temperatura. La temperatura puede variar dentro de un rango definido, en este caso entre 15 y 30 grados centígrados. Este entorno se define por los estados posibles, que son las diferentes temperaturas, y las acciones que el agente puede tomar para modificar la temperatura.

El agente es un controlador de temperatura cuyo objetivo es optimizar la temperatura del sistema. Debe aprender a ajustar la temperatura de manera eficiente, evitando desviaciones significativas de la temperatura objetivo y minimizando el número de pasos necesarios para alcanzar la temperatura deseada.

Un estado representa la temperatura actual del entorno. Dado que el rango de temperaturas es de 15 a 30 grados, hay un total de 16 estados posibles.

El controlador puede realizar tres posibles acciones:

1. Disminuir la temperatura: La temperatura disminuye en 1 grado.
2. Mantener la temperatura: La temperatura se mantiene sin cambios.
3. Aumentar la temperatura: La temperatura aumenta en 1 grado.

El sistema de recompensas está diseñado para guiar al agente hacia el objetivo. Si la temperatura actual es igual a la temperatura objetivo, el agente recibe una recompensa de +1. Si la temperatura difiere de la temperatura objetivo, el agente recibe una penalización proporcional a la diferencia entre la temperatura actual y la temperatura objetivo, dividida por 10. El propósito principal de hacer esta división es normalizar la penalización,

manteniéndola dentro de un rango manejable para el agente. Este sistema de recompensas incentiva al agente a mantener la temperatura lo más cercana posible al objetivo.

Durante el entrenamiento, el agente utiliza una política épsilon-greedy (ϵ) para la selección de acciones. Esto significa que, con una probabilidad ϵ, el agente elige una acción de manera aleatoria, favoreciendo así la exploración de nuevas estrategias de control. Por otro lado, con una probabilidad $1 - \epsilon$, el agente selecciona la acción que tiene el valor más alto en la tabla Q, lo cual fomenta la explotación de las estrategias que ya han demostrado ser eficientes. La tabla o matriz Q es una estructura que almacena valores Q para cada combinación de estado y acción posible en el entorno. Estos valores Q representan la calidad de una acción específica tomada desde un estado específico, e indican al agente la utilidad de tomar una acción en un estado dado, considerando las recompensas futuras esperadas.

El proceso de entrenamiento del controlador se lleva a cabo a través de una serie de episodios. En cada uno de estos episodios, el agente comienza con una temperatura inicial aleatoria y ejecuta una serie de acciones para ajustar la temperatura hasta alcanzar el objetivo o hasta que el episodio finalice. Durante este proceso, la tabla Q, que almacena los valores de calidad de las acciones, se actualiza continuamente. Esta actualización se basa en las recompensas recibidas por las acciones tomadas y en el valor futuro esperado de los estados subsiguientes. Así, el agente mejora progresivamente su política de control de temperatura, optimizando las acciones para futuros ajustes.

Durante este proceso de entrenamiento, se utilizan varios parámetros clave para controlar y optimizar el aprendizaje del agente:

- Tasa de aprendizaje (α): Establecida en 0.1, esta tasa controla la velocidad a la que se actualiza la tabla Q. Una tasa de aprendizaje más alta permite ajustes más rápidos, mientras que una más baja proporciona cambios más graduales.

- Factor de descuento (γ): Con un valor de 0.95, este parámetro determina la importancia de las recompensas futuras en comparación con las inmediatas. Un factor de descuento alto indica que se valoran más las recompensas a largo plazo.

- *Epsilon*: Inicialmente fijado en 1, épsilon se reduce gradualmente para favorecer la explotación sobre la exploración a medida que avanza el entrenamiento. Esto permite que el agente explore al principio, pero se enfoque en las mejores estrategias descubiertas más adelante.

- Tasa de decrecimiento de épsilon: Establecida en 0.0001, define la velocidad a la que ϵ decrece. Una tasa de decaimiento baja significa que el agente explorará durante más tiempo antes de centrarse en la explotación.

La función *reiniciar* selecciona aleatoriamente una temperatura inicial del espacio de estados y devuelve ese estado. Es esencial para restablecer el entorno al comienzo de cada episodio de entrenamiento. La función *paso* actualiza la temperatura del entorno según la acción tomada, calcula la recompensa correspondiente y verifica si el objetivo ha sido alcanzado. Es fundamental para la interacción del agente con el entorno y para la actualización de la tabla Q basada en las experiencias del agente.

El entrenamiento se realiza durante un número especificado de episodios, en este caso, 15000. A lo largo de estos episodios, se registra la recompensa acumulada para cada episodio y al finalizar el entrenamiento, se imprime la mejor tabla Q obtenida y se grafica

la suma acumulada de recompensas para visualizar el progreso del aprendizaje del agente. Este análisis permite evaluar la efectividad del algoritmo y el desempeño del agente en la optimización del control de temperatura.

7.2. Explicación del código en Python

El código para el proyecto de control de temperatura en una casa fue implementado por mí, inspirándome inicialmente en códigos de programación de entornos famosos en el campo del aprendizaje por refuerzo, tales como el problema del Lago Helado y el problema de los taxis, ambos disponibles a través de la librería Gymnasium.

El problema del Lago Helado implica navegar por un entorno de cuadrícula donde el objetivo es cruzar un lago helado desde un punto de inicio a un objetivo sin caer en agujeros abiertos en el hielo. Este entorno es una prueba clásica de la capacidad del algoritmo para planificar bajo incertidumbre y aprender de las interacciones con un entorno donde las acciones pueden tener resultados peligrosos. El problema de los taxis simula un entorno donde un taxi debe recoger y dejar pasajeros en diferentes ubicaciones. El desafío consiste en aprender la ruta más eficiente, considerando la ubicación del pasajero y del destino, lo que requiere que el algoritmo maneje múltiples estados y planifique secuencias de acciones en un entorno dinámico. Estos ejemplos clásicos me proporcionaron una sólida base teórica y técnica sobre cómo estructurar y desarrollar entornos de aprendizaje por refuerzo. A partir de este conocimiento, creé mi propio código adaptado a un nuevo entorno específico, el control de la temperatura en una casa, permitiendome aplicar y profundizar mi comprensión de los principios de aprendizaje por refuerzo, sino también abordar un problema práctico y cotidiano de una manera innovadora.

Elegí Python como el lenguaje de programación para desarrollar este proyecto por varias razones. Primero, Python cuenta con un extenso ecosistema de bibliotecas como NumPy y Matplotlib, que son fundamentales para el manejo de datos y la visualización en proyectos de inteligencia artificial. Además, la sintaxis clara y concisa de Python facilita la escritura de código legible y mantenible, una ventaja crucial en proyectos de investigación y desarrollo donde la experimentación y la iteración son frecuentes. La amplia comunidad de Python proporciona un increíble soporte y una gran cantidad de recursos que enriquecen el proceso de aprendizaje y desarrollo. Finalmente, la flexibilidad y la capacidad de Python para permitir el desarrollo y prueba de ideas de manera rápida y eficiente hacen de este lenguaje una elección óptima para implementar conceptos complejos en aprendizaje por refuerzo, asegurando que el proceso de desarrollo sea efectivo y eficiente.

Primero se importan las librerías necesarias. Numpy se utiliza para operaciones numéricas y manipulación de matrices, *matplotlib.pyplot* para la visualización de datos y *seaborn* para mejorar la estética y la interpretación de los gráficos.

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

La clase *EntornoTemperatura* se define a continuación. Esta clase modela el entorno en el que operará el agente de aprendizaje. El constructor *__init__* inicializa varios atributos del entorno, como la *temperatura_objetivo* (por defecto 22 grados) y la *temperatura_inicial* (por defecto 20 grados). Además, se definen *espacio_acciones*, que representa las posibles acciones (disminuir, mantener, aumentar la temperatura), y *espacio_estados*, que representa los posibles estados de temperatura. También se establece *max_pasos* como el número máximo de pasos en un episodio.

```
class EntornoTemperatura:
    def __init__(self, temperatura_objetivo=22,
temperatura_inicial=20):
        self.temperatura_objetivo = temperatura_objetivo
        self.temperatura_actual = temperatura_inicial
        self.espacio_acciones = np.array([-1, 0, 1])  # Acciones:
disminuir, mantener, aumentar la temperatura
        self.espacio_estados = np.arange(15, 30)  # Posibles
estados de temperatura
        self.max_pasos = 100
```

El método *reiniciar,* restablece el entorno a un estado inicial aleatorio y reinicia el contador de pasos, devolviendo el estado inicial.

```
def reiniciar(self):
        self.temperatura_actual =
np.random.choice(self.espacio_estados)
        self.pasos = 0
        return self.temperatura_actual
```

Podemos observar en el siguiente código el método *paso,* que aplica una acción, actualiza la temperatura y calcula la recompensa. Cada vez que se llama a este método, se incrementa el número de pasos. La temperatura se ajusta según la acción tomada y se limita dentro del rango válido usando *np.clip*. La recompensa se calcula en función de la proximidad a la temperatura objetivo: si la temperatura actual coincide con la objetivo, la recompensa es 1, de lo contrario, es proporcional a la distancia a la temperatura objetivo. El método devuelve el nuevo estado, la recompensa, y si el episodio ha terminado o ha sido truncado.

```
def paso(self, accion):
        self.pasos += 1
        self.temperatura_actual += self.espacio_acciones[accion]

        # Limitar la temperatura al rango del espacio de estados
        self.temperatura_actual = np.clip(self.temperatura_actual,
self.espacio_estados[0], self.espacio_estados[-1])

        # Recompensa basada en la cercanía a la temperatura
objetivo
        if self.temperatura_actual == self.temperatura_objetivo:
            recompensa = 1
```

```
        else:
            recompensa = -abs(self.temperatura_objetivo -
self.temperatura_actual) / 10

        terminado = self.pasos >= self.max_pasos
        truncado = False

        return self.temperatura_actual, recompensa, terminado,
truncado, {}
```

La función entrenar se encarga de entrenar al agente mediante el algoritmo de *Q-Learning*. Inicializa el entorno y una tabla Q con ceros para representar las estimaciones de los valores Q para cada estado y acción. Se definen varios parámetros del algoritmo, como la tasa_aprendizaje, el factor_descuento, el epsilon y la tasa_decaimiento_epsilon. Además, se utiliza un generador de números aleatorios y un array para almacenar las recompensas obtenidas en cada episodio.

```
def entrenar(episodios):
    entorno = EntornoTemperatura()

    tabla_q = np.zeros((len(entorno.espacio_estados),
len(entorno.espacio_acciones)))

    tasa_aprendizaje = 0.1
    factor_descuento = 0.95
    epsilon = 1
    tasa_decaimiento_epsilon = 0.0001
    rng = np.random.default_rng()

    recompensas_por_episodio = np.zeros(episodios)
```

Dentro del bucle de entrenamiento, que se ejecuta para cada episodio, se reinicia el entorno y se establece el estado inicial. Luego, en un bucle mientras el episodio no haya terminado, la acción se selecciona usando una política épsilon-greedy (exploración/explotación). Se ejecuta la acción y se recibe la nueva información del entorno, incluyendo el nuevo estado y la recompensa. La tabla Q se actualiza usando la fórmula de actualización *Q-Learning*. Se ajusta el valor de épsilon para reducir la exploración con el tiempo y se registra la recompensa obtenida en el episodio. Además, se imprime la recompensa cada 100 episodios.

```
    for i in range(episodios):
        estado = entorno.reiniciar()
        terminado = False
        truncado = False

        while not terminado and not truncado:
            if rng.random() < epsilon:
                accion =
rng.integers(len(entorno.espacio_acciones))
            else:
```

```
                accion = np.argmax(tabla_q[estado -
entorno.espacio_estados[0], :])

            nuevo_estado, recompensa, terminado, truncado, _ =
entorno.paso(accion)

            tabla_q[estado - entorno.espacio_estados[0], accion] +=
tasa_aprendizaje * (
                recompensa + factor_descuento *
np.max(tabla_q[nuevo_estado - entorno.espacio_estados[0], :]) -
tabla_q[estado - entorno.espacio_estados[0], accion]
            )

            estado = nuevo_estado

        epsilon = max(epsilon - tasa_decaimiento_epsilon, 0)
        recompensas_por_episodio[i] = recompensa

        if (i + 1) % 100 == 0:
            print(f"Episodio: {i + 1} - Recompensa:
{recompensas_por_episodio[i]}")
1
```

Al final del entrenamiento se imprime la mejor tabla Q obtenida. Luego, se calcula la suma acumulada de recompensas en ventanas de 100 episodios y se grafica la evolución de la recompensa acumulada a lo largo de los episodios.

```
print(f"Mejor Q: {tabla_q}")

    suma_recompensas = np.zeros(episodios)
    for t in range(episodios):
        suma_recompensas[t] =
np.sum(recompensas_por_episodio[max(0, t - 100):(t + 1)])
    plt.plot(suma_recompensas)
    plt.xlabel("Episodios")
    plt.ylabel("Suma de recompensas")
    plt.title("Evolución de la recompensa acumulada")
    plt.show()
```

También imprimiremos un gráfico de calor de la tabla Q. Esta es una representación intuitiva y fácil de entender, que de otro modo sería solo un conjunto de números.

```
plt.figure(figsize=(10, 8))

    sns.heatmap(tabla_q, annot=True, fmt=".2f", cmap="coolwarm",
cbar=True)

    plt.title('Mapa de Calor de la Tabla Q')

    plt.xlabel('Acciones (0: disminuir, 1: mantener, 2: aumentar)')

    plt.ylabel('Estados (Temperaturas de 15 a 29 grados)')

    plt.show()
```

Finalmente, el código ejecuta la función de entrenamiento con 15.000 episodios en caso de que se ejecute como programa principal. Esto se logra mediante el uso de la condición

if __name__ == '__main__': en Python, que asegura que la función entrenar solo se llame y ejecute directamente cuando el archivo Python sea ejecutado independientemente. De esta manera, si el archivo es importado en otro script como un módulo, la función de entrenamiento no se ejecutará automáticamente, evitando así ejecuciones no deseadas y facilitando la reutilización del código en diferentes contextos sin interferencias.

```
if __name__ == '__main__':
    entrenar(15000)
```

7.3. Resultados

La actualización de la tabla Q es un proceso muy importante en el algoritmo *Q-Learning*, y a continuación se detalla el proceso de actualización de esta tabla paso a paso, basándose en las interacciones del agente con el entorno y los cálculos realizados para ajustar los valores Q.

Antes de comenzar el entrenamiento, la tabla Q se inicializa con valores cero o con valores pequeños aleatorios. Esto representa el conocimiento inicial del agente, que es básicamente nulo o muy limitado sobre qué acciones son mejores en cada estado. Durante cada episodio de entrenamiento, el agente experimenta una secuencia de estados, acciones y recompensas. Para cada acción tomada en un estado dado, el agente observa una recompensa y un nuevo estado resultante. El agente selecciona una acción basada en la política épsilon-greedy, por lo que con probabilidad $1 - \epsilon$ se elige la acción que tiene el máximo valor Q en la tabla Q para el estado actual, y con probabilidad ϵ elige una acción al azar. Después de tomar una acción, el agente observa la recompensa resultante y el nuevo estado al que llega. El valor Q para el par estado-acción se actualiza usando la fórmula de actualización de *Q-Learning*:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$

Este proceso se repite para cada paso dentro de un episodio y para cada episodio durante el entrenamiento. Con el tiempo, los valores Q convergen a valores que representan las recompensas acumuladas óptimas para cada par de estado-acción, bajo una política óptima. A medida que se repite este proceso a lo largo de muchos episodios, los valores Q en la tabla tienden a converger hacia los valores que representan la recompensa total máxima que el agente puede esperar a partir de cada par estado-acción, siguiendo la mejor estrategia posible a partir de ese estado. Esto permite al agente tomar decisiones cada vez más informadas sobre qué acciones tomar en diferentes situaciones, basándose en el aprendizaje acumulado reflejado en la tabla Q

Mediante un mapa de calor en la *Figura 16*, se muestra cómo los valores Q varían para diferentes combinaciones de estados y acciones. La escala de color a la derecha indica la magnitud de los valores Q, donde el rojo representa valores más altos y el azul valores más bajos.

Las temperaturas en los extremos del rango, es decir, las filas 0,1,13 y 14, tienden a tener valores Q más altos para las acciones que ajustan la temperatura hacia el objetivo central de 22 grados. Por ejemplo, en las temperaturas más bajas, aumentar la temperatura (acción 2) tiene valores más altos, mientras que, en las más altas, disminuir la temperatura

(acción 0) es más favorable. Las temperaturas cercanas al objetivo de 22 grados, como las filas 6,7 y 8 tienen valores Q más altos para la acción de mantener la temperatura (acción 1), indicando que el agente ha aprendido que, en estos estados, mantener la temperatura es óptimo.

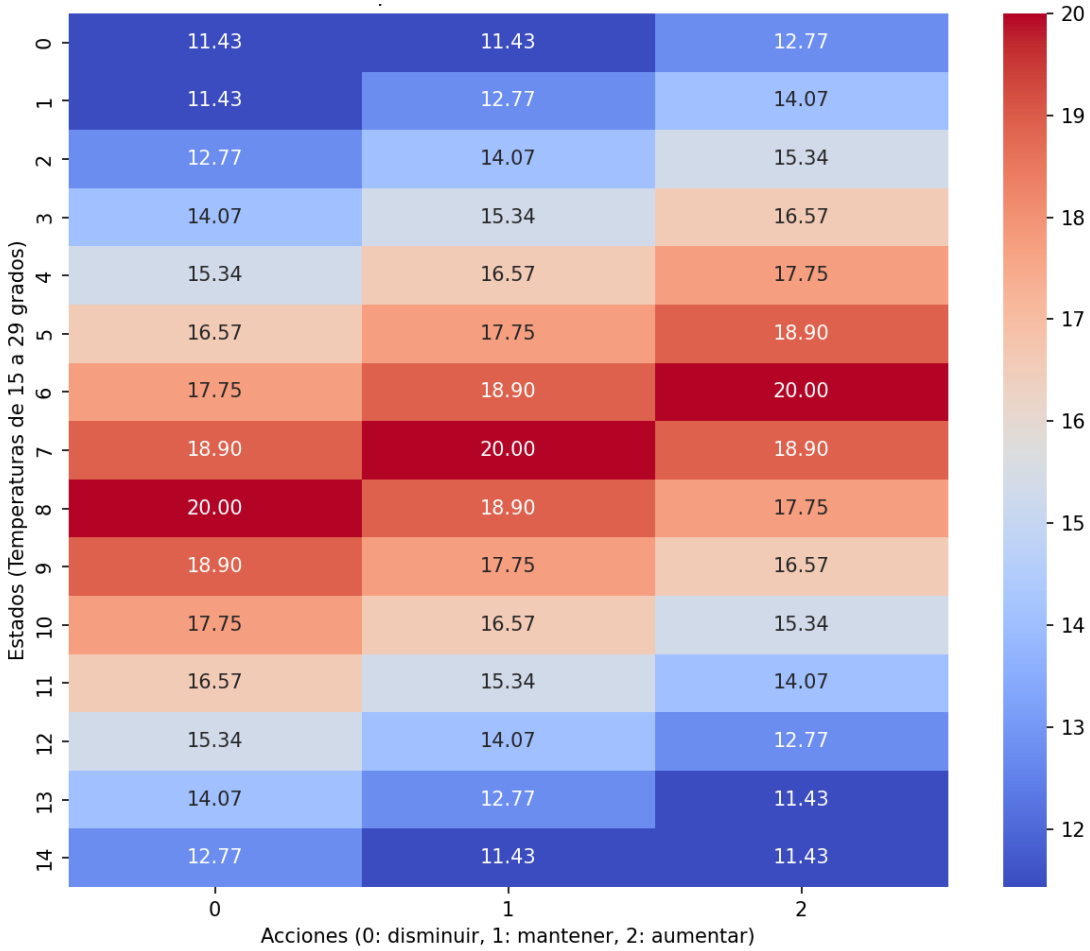


Figura 16. Mapa de calor de la tabla Q

La columna 0, que indica la acción de disminuir la temperatura, muestra valores más altos en estados con temperaturas más altas, indicando que disminuir la temperatura es beneficioso cuando la temperatura actual es superior a la deseada. En la columna 1, de mantenimiento de la temperatura, los valores son generalmente más altos en el centro del rango de temperaturas, sugiriendo que mantener la temperatura es la mejor acción cuando la temperatura está cerca del objetivo. Por último, en la columna 2, que indica aumentar la temperatura, los valores Q son más altos en los estados de temperaturas más bajas, lo que es lógico ya que aumentar la temperatura ayuda a acercarse al objetivo cuando está demasiado frío.

Para evaluar los resultados del entrenamiento del sistema de control de temperatura, se procede a analizar el gráfico de recompensas acumuladas de la *Figura 17* obtenida al finalizar los 15.000 episodios. El gráfico muestra la evolución de la recompensa acumulada a lo largo de los episodios durante el entrenamiento del agente para mantener la temperatura cerca del objetivo utilizando el algoritmo de *Q-Learning*.

El eje horizontal representa el número de episodios, que en este caso son 15.000. Cada episodio es una iteración completa del proceso de aprendizaje, donde el agente comienza

desde un estado inicial aleatorio y ejecuta acciones hasta que el episodio termina. El eje vertical muestra la suma de recompensas acumuladas en ventanas de 100 episodios. La recompensa acumulada es una medida de cuán bien está desempeñándose el agente en su tarea de mantener la temperatura cerca del objetivo a lo largo del tiempo.

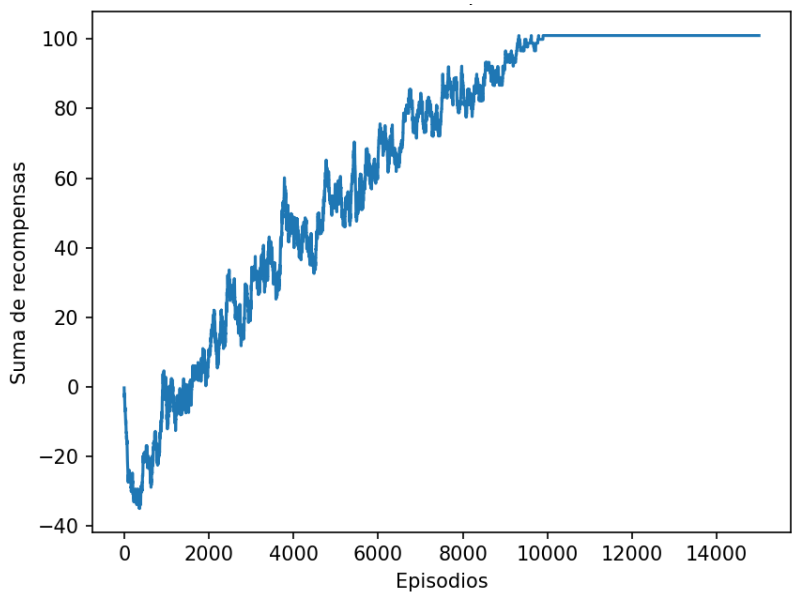


Figura 17. Gráfico de recompensas acumuladas

En los primeros episodios, la suma de recompensas muestra fluctuaciones significativas, llegando incluso a valores negativos. Esto es normal en las fases iniciales de entrenamiento, ya que el agente está explorando diferentes acciones y aprendiendo la dinámica del entorno. A medida que el número de episodios aumenta, se observa una clara tendencia ascendente en la suma de recompensas acumuladas. Esto indica que el agente está mejorando su desempeño y aprendiendo a mantener la temperatura cerca del objetivo de manera más consistente. Hacia los últimos episodios, la curva de recompensas se estabiliza cerca de un valor alto. Esta estabilización sugiere que el agente ha aprendido una política eficaz para controlar la temperatura y está obteniendo recompensas positivas de manera consistente.

Este ejemplo práctico demuestra la aplicabilidad de *Q-Learning* en un problema como el control de temperatura en un entorno doméstico. Los resultados sugieren que los algoritmos de aprendizaje por refuerzo pueden ser efectivos para gestionar sistemas en entornos dinámicos y complejos, adaptándose continuamente a las condiciones cambiantes para optimizar el desempeño.

CAPÍTULO 8: Conclusiones

Este trabajo ha abarcado una revisión exhaustiva del aprendizaje por refuerzo, profundizando en su teoría, aplicaciones, y desafíos. A través de una serie de capítulos temáticos, hemos explorado desde los principios fundamentales hasta implementaciones específicas que demuestran la versatilidad y potencial del aprendizaje por refuerzo para abordar problemas de optimización complejos en ambientes dinámicos y a menudo inciertos.

El aprendizaje por refuerzo se ha distinguido claramente de otros paradigmas de aprendizaje automático por su enfoque único en la toma de decisiones secuenciales y su objetivo de maximizar la suma de recompensas a lo largo del tiempo. A diferencia del aprendizaje supervisado y no supervisado, el aprendizaje por refuerzo opera mediante un ciclo de retroalimentación dependiente de las acciones previas tomadas por el agente, lo cual introduce una complejidad adicional en la modelación de problemas y algoritmos.

Los métodos basados en modelos ofrecen ventajas en entornos controlados, pero requieren suposiciones y conocimientos previos que pueden no estar disponibles o ser inexactos. Por otro lado, los métodos libres de modelo, como *Q-Learning* y SARSA, proporcionan flexibilidad y robustez en entornos no estructurados o desconocidos, sacrificando la eficiencia y, a veces, la velocidad de convergencia.

La sinergia entre el aprendizaje profundo y el aprendizaje por refuerzo ha generado avances significativos, particularmente en el manejo de grandes volúmenes de datos y en la capacidad de generalización en espacios de estado complejos. Esta integración ha facilitado desarrollos notables en áreas como la navegación autónoma y los juegos de inteligencia artificial, donde la capacidad para procesar y actuar sobre información visual o sensorial compleja es crítica.

Se discutieron desafíos intrínsecos como el dilema de exploración versus explotación y la maldición de la dimensionalidad. Las soluciones propuestas, que incluyen algoritmos de reducción de dimensionalidad y estrategias de exploración mejoradas, como la política *epsilon-greedy*, ilustran enfoques prometedores para mejorar el rendimiento y la eficiencia de los sistemas de aprendizaje por refuerzo.

El desarrollo y evaluación de un sistema de control de temperatura mediante *Q-Learning* proporcionó una aplicación concreta de los principios de aprendizaje por refuerzo. Este caso no solo demostró la aplicabilidad de la teoría en un contexto práctico, sino que también ofreció perspectivas valiosas sobre el ajuste de parámetros y la configuración del entorno de aprendizaje para optimizar el rendimiento del sistema.

En conclusión, este trabajo ha confirmado el potencial transformador del aprendizaje por refuerzo en el campo del aprendizaje automático. Con su enfoque único en la optimización de decisiones secuenciales y su capacidad para adaptarse y aprender de manera autónoma, el aprendizaje por refuerzo se establece como una herramienta fundamental para enfrentar algunos de los problemas más complejos y dinámicos de la actualidad. A medida que continuamos explorando y expandiendo los límites de esta tecnología, es crucial mantener un enfoque equilibrado entre teoría y práctica, asegurando que los avances en el campo se traduzcan en mejoras concretas y aplicables en múltiples sectores.

BIBLIOGRAFÍA

Al-Hamadani, M., Fadhel, M., Alzubaidi, L., & Harangi, B. (2024). Reinforcement Learning Algorithms and Applications in Healthcare and Robotics: A Comprehensive and Systematic Review. *Sensors*, *24*(8), 2461. https://doi.org/10.3390/s24082461

Bayes, T., & Prince, M. (1763). LII. An essay towards solving a problem in the doctrine of chances. By the late Rev. Mr. Bayes, F. R. S. communicated by Mr. Price, in a letter to John Canton, A. M. F. R. S. *Philosophical Transactions of the Royal Society of London*, *53*, 370–418. https://doi.org/10.1098/rstl.1763.0053

Bellman, R. (1957). Dynamic Programming. Princeton University Press, Princeton, N.J.,. *Science*, *127*(3304), 976–976. https://doi.org/10.1126/science.127.3304.976.a

Bengio, Y., Lecun, Y., & Hinton, G. (2021). Deep learning for AI. *Communications of the ACM*, *64*, 58–65. https://doi.org/10.1145/3448250

Benzécri, J. (1973). L'analyse des données: L'analyse des correspondances. *Volumen 2 de L'analyse Des Données: Leçons Sur l'analyse Factorielle et La Reconnaissance Des Formes, et Travaux Du Laboratoire de Statistique de l'Université de Paris VI, J.-P. Benzécri*.

Berner, C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., Józefowicz, R., Gray, S., Olsson, C., Pachocki, J., Petrov, M., d. O. Pinto, H. P., Raiman, J., Salimans, T., Schlatter, J., … Zhang, S. (2019). *Dota 2 with Large Scale Deep Reinforcement Learning*. https://doi.org/https://doi.org/10.48550/arXiv.1912.06680

Boser, B., Guyon, I., & Vapnik, V. (1996). A Training Algorithm for Optimal Margin Classifier. *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory*, *5*. https://doi.org/10.1145/130385.130401

Colliot, O. (Ed.). (2023). *Machine Learning for Brain Disorders* (Vol. 197). Springer US. https://doi.org/10.1007/978-1-0716-3195-9

Covington, P., Adams, J., & Sargin, E. (2016). *Deep Neural Networks for YouTube Recommendations*. https://doi.org/10.1145/2959100.2959190

Crevier, D. (1993). *AI: The Tumultuous History of the Search for Artificial Intelligence*.

Ding, Z., Huang, Y., Yuan, H., & Dong, H. (2020). Introduction to Reinforcement Learning. In *Springer eBooks* (pp. 47–123). https://doi.org/10.1007/978-981-15-4095-0_2

Fei, Y., Liao, W., Lu, X., Taciroglu, E., & Guan, H. (2023). Semi-supervised learning method incorporating structural optimization for shear-wall structure design using small and long-tailed datasets. *Journal of Building Engineering*, *79*, 107873. https://doi.org/https://doi.org/10.1016/j.jobe.2023.107873

François-Lavet, V., Henderson, P., Islam, R., Bellemare, M. G., & Pineau, J. (2018). An Introduction to Deep Reinforcement Learning. *Foundations and Trends® in Machine Learning*, *11*(3–4), 219–354. https://doi.org/10.1561/2200000071

Jaeger, B., & Geiger, A. (2023). *An Invitation to Deep Reinforcement Learning*. https://doi.org/https://doi.org/10.48550

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An Introduction to Statistical Learning* (Vol. 103). Springer New York. https://doi.org/10.1007/978-1-4614-7138-7

Jordan, M., & Mitchell, T. M. (2015). Machine Learning: Trends, Perspectives, and Prospects. *Science (New York, N.Y.)*, *349*, 255–260. https://doi.org/10.1126/science.aaa8415

Kober, J., Bagnell, J., & Peters, J. (2013). Reinforcement Learning in Robotics: A Survey. *The International Journal of Robotics Research*, *32*, 1238–1274. https://doi.org/10.1177/0278364913495721

Levine, S., Pastor, P., Krizhevsky, A., & Quillen, D. (2016). Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection. *The International Journal of Robotics Research*, *37*. https://doi.org/10.1177/0278364917710318

Li, Y. (2018). *Deep Reinforcement Learning: An Overview*. https://doi.org/https://doi.org/10.48550/arXiv.1810.06339

Lloyd, S. (1982). Least squares quantization in PCM. *IEEE Transactions on Information Theory*, *28*(2), 129–137. https://doi.org/10.1109/TIT.1982.1056489

MacQueen, J. (1967). *Some methods for classification and analysis of multivariate observations*.

McCorduck, P. (1979). *Machines Who Think: A Personal Inquiry into the History and Prospects of Artificial Intelligence*.

Metropolis, N., & Ulam, S. (1949). The Monte Carlo Method. *Journal of the American Statistical Association*, *44*(247), 335–341. https://doi.org/10.1080/01621459.1949.10483310

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529–533. https://doi.org/10.1038/nature14236

Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective*. MIT Press.

Nilsson, N. J. (2009). *The Quest for Artificial Intelligence*. Cambridge University Press. https://doi.org/10.1017/CBO9780511819346

Núñez Reiz, A., Armengol de la Hoz, M. A., & Sánchez García, M. (2019). Big Data Analysis y Machine Learning en medicina intensiva. *Medicina Intensiva*, *43*(7), 416–426. https://doi.org/10.1016/j.medin.2018.10.007

Pearson, K. (1901). On lines and planes of closest fit to systems of points in space. *Philosophical Magazine Series 1*, *2*, 559–572. https://api.semanticscholar.org/CorpusID:125037489

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, *1*(1), 81–106. https://doi.org/10.1007/BF00116251

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*(6088), 533–536. https://doi.org/10.1038/323533a0

Rummery, G., & Niranjan, M. (1994). On-Line Q-Learning Using Connectionist Systems. *Technical Report CUED/F-INFENG/TR 166*.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal Policy Optimization Algorithms. *ArXiv*, *abs/1707.06347*.

Sewak, M. (2019). Introduction to Deep Learning. In *Deep Reinforcement Learning* (pp. 75–88). Springer Singapore. https://doi.org/10.1007/978-981-13-8285-7_6

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., & Hassabis, D. (2017). Mastering the game of Go without human knowledge. *Nature*, *550*(7676), 354–359. https://doi.org/10.1038/nature24270

Sivamayil, K., Rajasekar, E., Aljafari, B., Nikolovski, S., Vairavasundaram, S., & Vairavasundaram, I. (2023). A Systematic Study on Reinforcement Learning Based Applications. *Energies*, *16*(3), 1512. https://doi.org/10.3390/en16031512

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, *3*(1), 9–44. https://doi.org/10.1007/BF00115009

Sutton, R. S., & Barto, A. G. (2014). *Reinforcement Learning: An Introduction. Second edition. MIT Press*.

Tan, F., Yan, P., & Guan, X. (2017). *Deep Reinforcement Learning: From Q-Learning to Deep Q-Learning* (pp. 475–483). https://doi.org/10.1007/978-3-319-70093-9_50

TURING, A. M. (1950). Computing Machinery and Intelligence. *Mind*, *LIX*(236), 433–460. https://doi.org/10.1093/mind/LIX.236.433

Vemuri, V. K. (2020). The Hundred-Page Machine Learning Book. *Journal of Information Technology Case and Application Research*, *22*(2), 136–138. https://doi.org/10.1080/15228053.2020.1766224

Wang, H., Liu, N., Zhang, Y., Feng, D., Huang, F., Li, D., & Zhang, Y. (2020). Deep reinforcement learning: a survey. *Frontiers of Information Technology & Electronic Engineering*, *21*(12), 1726–1744. https://doi.org/10.1631/FITEE.1900533

Watkins, C. J. C. H., & Dayan, P. (1992). Q-learning. *Machine Learning*, *8*(3–4), 279–292. https://doi.org/10.1007/BF00992698

Wiering, M., & van Otterlo, M. (2012). *Reinforcement Learning* (M. Wiering & M. van Otterlo, Eds.; Vol. 12). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-27645-3

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, *8*(3–4), 229–256. https://doi.org/10.1007/BF00992696

Yarowsky, D. (1995). Unsupervised word sense disambiguation rivaling supervised methods. *Proceedings of the 33rd Annual Meeting on Association for Computational Linguistics* -, 189–196. https://doi.org/10.3115/981658.981684

Yu, C., Liu, J., Nemati, S., & Yin, G. (2021). Reinforcement Learning in Healthcare: A Survey. *ACM Computing Surveys*, *55*, 1–36. https://doi.org/10.1145/3477600

Zhang, A., Lipton, Z. C., Li, M., & Smola, A. J. (2023). *Dive into Deep Learning*. https://doi.org/https://doi.org/10.48550/arXiv.2106.11342