Proyecto Fin de Máster

Máster en Ingeniería Electrónica, Robótica y Automática

Aplicación de técnicas de Deep Reinforcement Learning a misiones de exploración para vehículos autónomos en escenarios lacustres.

Autor: Samuel Yanes Luis

Tutores: Sergio Luis Toral Marín

Daniel Gutiérrez Reina

Dpto. Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2020

ACE-TI | Aplicaciones Cibernéticas de la Electrónica a las Tecnologías de la Información

Proyecto Fin de Máster
Máster en Ingeniería Electrónica, Robótica y Automática

Aplicación de técnicas de Deep Reinforcement Learning a misiones de exploración para vehículos autónomos en escenarios lacustres.

Autor:
Samuel Yanes Luis

Tutores:
Sergio Luis Toral Marín
Catedrático
Daniel Gutiérrez Reina
Profesor Contratado

Dpto. Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2020

Proyecto Fin de Máster: Aplicación de técnicas de Deep Reinforcement Learning a misiones de exploración para vehículos autónomos en escenarios lacustres.

Autor: Samuel Yanes Luis
Tutores: Sergio Luis Toral Marín
Daniel Gutiérrez Reina

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

A mi madre, Marianela.
A mi padre, Pedro.
A mi hermana, Raquel.
A mi gato, Luke.

Agradecimientos

En el fondo, los científicos somos gente con suerte: podemos jugar a lo que queramos durante toda la vida

Lee Smolin

Este trabajo es el inicio de una etapa extraordinariamente nueva y apasionante para mí. No existo en una forma más feliz que cuando dedico mi tiempo a la ciencia y a la técnica y en este trabajo está la ilusión naciente de años de estudio, entusiasmada por hacerse grande y aprender más.

No obstante, como ocurre en la ciencia, sin la contribución de grandes personas no hubiera tenido la *inercia* suficiente para seguir avanzando.

En primer lugar quiero agradecer profundamente a los profesores Sergio Luis Toral Marín y Daniel Gutiérrez Reina por haberme dado la oportunidad de empezar a trabajar junto a ellos y comenzar mi carrera investigadora. Son el mejor referente de buena docencia, profesionalidad y brillantez que pude pedir.

También quiero agradecer a mi familia: mi padre, mi madre, mi hermana y a mi gato, el cariño constante, el apoyo durante los años de formación en la ETSI y la paciencia tierna a la hora de escucharme divagar sobre cada ecuación de cada libro que leí. Ellos son lo único *necesario* para mí, todo lo demás es *contingente*.

Por último, quiero agradecer a las personas más fieles y buenas que tengo el orgullo de llamar amigos: Salomé, Alejandro y Andrea, triunvirato de mis alegrías y fuente infinita de risas. Si algo ha merecido la pena de tantos meses de confinamiento, es haberlos tenido a ellos cerca. Este trabajo también es suyo.

Samuel Yanes Luis

Escuela Técnica Superior de Ingeniería de Sevilla.

Sevilla, 2020

Resumen

Este trabajo se enmarca dentro un proyecto de colaboración entre la Escuela Técnica Superior de Ingeniería de Sevilla (ACE-TI) y la Universidad Nacional de Asunción de Paraguay para la monitorización y supervisión del estado de contaminación del lago Ypacaraí (Paraguay). En este proyecto se ha llevado a cabo un estudio de distintas metodologías de planificación y exploración con vehículos autónomos basadas en las, cada vez más populares, técnicas de Reinforcement Learning y Deep Reinforcement Learning.

Se ha diseñado un escenario basado en la geometría espacial del lago y se han parametrizado y aplicado distintos algoritmos como Deep Q-Learning y Double Deep Q-Learning para acometer la tarea de que el agente, un vehículos de superficie autónomo, aprenda a cubrir con eficiencia todas las zonas del lago siguiendo distintos criterios de ponderación. Adicionalmente, se ha realizado un acercamiento a la sintonización de los hiperparámetros de entrenamiento para alcanzar resultados cercanos a los óptimos.

Abstract

This research takes place within the context of a collaborative project between the Engineering School of Seville (ACE-TI research group) and Asunción University of Paraguay. Its objetive are the supervision and monitoring of the contamination state of Ypacari Lake in Paraguay. In this academic research, a study of several methodologies has been done whith the focus in exploration and path planning in Autonomous Surface Vehicles based on recently trending Reinforcement Learning and Deep Reinforcement Learning technics.

An environment based on the spatial geometry of Ypacarai Lake has been designed and some algorithms as Deep Q-Learning and Double Deep Q-Learning have been implemented in order to achieve the main goal: training an autonomous surface vehicle to explore with efficiency the lake following some importance criteria. In addition, a hyperparameter tunning approach has been achieved for obtaining more optimal-near results.

Índice

Capítulo 1: Introducción

1.1 Antecedentes

El lago Ypacaraí es una masa de agua situada entre los departamentos Central y Cordillera de Paraguay. Está rodeado de las ciudades Areguá y San Bernardino (entre otras) y tiene una extensión de unos 60 Km^2 con una profundidad media de 3m. Poseía una importancia considerable en relación al turismo local (como lago de recreo), no obstante, debido al desarrollo económico (sobretodo de corte industrial y agropecuario) de las zonas aledañas, ha experimentado en los últimos años un aumento de la contaminación de sus aguas.

Figura 1.1 Vista aérea del lago Ypacaraí..

Los residuos humanos de las ciudades junto con los provenientes de las ciudades que rodean la zona, han causado también un aumento considerable de colonias de cianobacterias en el agua (de ahí su característico color verde esmeralda). Los vertidos, ricos en nitrógeno, fósforo y metales constituyen un catalizador para las colonias de bacterias y suciedad en toda la superficie navegable y en las zonas de humedales que conforman el lago. Este efecto compromete no solo al ecosistema animal de la cuenca sino también la salubridad del suministro de agua de las ciudades cercanas, cuya única fuente de recursos hídricos está en el propio Ypacaraí.

Figura 1.2 Efecto de las cianobacterias en el lago..

Las cianobacterias, producto del proceso de eutrofización (enriquecimiento del medio por razones externas; humanas), son organismos que consumen muy rápidamente el oxígeno disuelto propiciando la anoxia del medio. Así, con la severa degradación de la calidad del agua y el fitoplancton y/o peces muertos surgen olores nauseabundos causando enfermedades en el ganado y los seres humanos, incluso la muerte [1].

Cabe destacar que la naturaleza de los brotes de cianobacterias es dinámica y bastante caótica [1], por lo que la predicción de dónde y cuando surgirán no es un proceso trivial. Este dinamismo inherente sugiere que es necesaria una tarea de monitorización continua como primera medida de prevención de enfermedades humanas y animales. Así, la contaminación se distribuye de forma irregular por la superficie del Ypacaraí con especial concentración en las zonas cercanas a las ciudades de Areguá y San Bernardino, donde la ausencia de infraestructura de alcantarillado y filtrado séptico supone un repunte de vertidos y de bacterias.

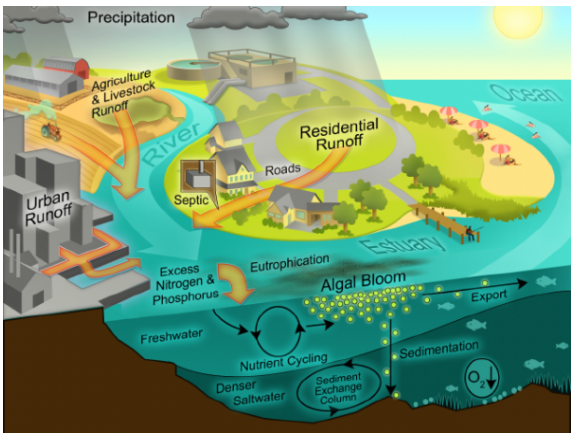


Figura 1.3 Esquema de generación de la eutrofización. Fuente: *https://projecteutrophication.weebly.com*.

En este sentido, se han dedicado numerosos esfuerzos técnicos y de investigación para tratar esta problemática: el gobierno de Paraguay ha instalado 3 estaciones de monitorización del bioma y entre el 2015 y el 2016 se llevaron a cabo 12 estudios

[1] Fuente: *http://cooperacion.us.es/sistema-de-monitorizacion-de-agentes-contaminantes-en-el-lago-ypacarai-mediante-el-uso-de-vehiculos*

de campo para el análisis de los brotes agresivos surgidos en esos años y también se han contemplado otras ideas desde la perspectiva de la infraestructura: construcción de desvíos de los ríos que desembocan en el Ypacaraí, cordones flotante para la contención de las algas de superficie, mejora del saneamiento de residuos de las industrias colindantes... [2].

La problemática de la contaminación de la cuenco hidrográfica del Ypacaraí es un reto interdisciplinar en el que trabajan perfiles ingenieriles y científicos diversos. Las soluciones vienen de manos de un conjunto muy variado de técnicas que no solo se centran en reducir los vertidos y la contaminación sino en supervisar y estudiar el estado actual del lago para analizar cuáles es la forma más eficiente de revertir la situación.

1.2 Monitorización mediante ASVs

Resulta obvio que es necesario, aún cuando las medidas infraestructurales son indispensables, trabajar en el sentido de una monitorización eficiente de los brotes para tener una imagen siempre actualizada del estado biológico de las distintas zonas del lago. En esta línea, un proyecto de cooperación de la Universidad de Sevilla [2] (grupo ACETI del Departamento de Ingeniería Electrónica de la misma universidad) y la Universidad Nacional de Asunción (FIUNA), denominado "Autonomous Surface Vehicle for the study of water quality in Lakes" ha planteado la utilización de Vehículos Autónomos de Superficie, ASV de sus siglas en inglés, para sustituir las inspecciones humanas con prototipos de barcos pequeños no tripulados que sean capaces de obtener y trasmitir la información de medida de los sensores de forma conjunta.

La tarea de monitorización es harto difícil debido a la gran extensión del lago (de hasta 70 Km^2 si se tiene en cuenta las zonas de humedales colindantes). Está claro que es necesario una flota importante de vehículos con capacidad de medición suficiente y autonomía prácticamente total. En este sentido, los ASV que se plantean en el proyecto de cooperación anteriormente citado suponen un avance importante respecto de las misiones humanas que se realizaban anteriormente, en las que se tomaban los parámetros de calidad del agua a mano y gracias a lanchas motoras. La toma de medida a mano supone un tiempo y recursos a la larga intolerables y es una tarea que es fácilmente automatizable mediante el despliegue de vehículos autónomos.

El diseño de los ASV para la exploración del lago ha sufrido varias transformaciones a lo largo del tiempo de desarrollo del proyecto. Finalmente, el último prototipo de ASV que se diseñó para esta tarea, denominado *Cormorán-II*, resultó un vehículo con estructura de catamarán doble, longitud de 4 metros y ancho de 2 metros. Dispone de tres paneles solares (con potencia de 300W) que alimentan a los microcontroladores (PixHawk para la navegación y control de bajo nivel de motores y Nvidia Jetson

[2] Número de referencia del proyecto: PINV15-177

para la algoritmia y procesamiento de datos) y a los sensores a bordo: GNSS (Global Navigation Satellite System - localización geográfica), sensores externoceptivos para el *mapping* del entorno (LIDAR para el mapeo y sensor ultrasonido para la evitación de obstáculos), sensores de calidad del agua (batímetros) y de calidad del aire y módulos de comunicaciones [3].

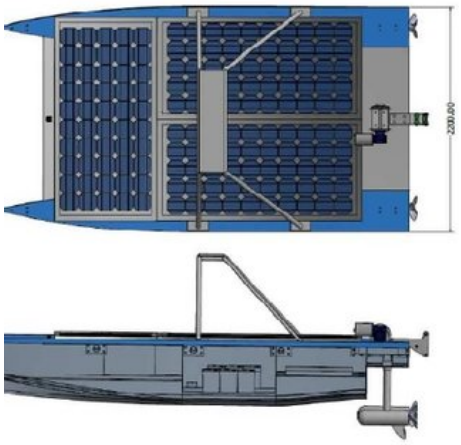


Figura 1.4 Diseño del vehículo de superficie Cormorán-II.

Tabla 1.1 Listado de módulos y componentes del Cormorán-II.

Módulo	Componente
Controladores principales	- Nvidia Jetson TX2 (procesamiento de datos y *pathplanning*) - PixHawk (control de motores)
Módulo de potencia	- 3 paneles solares de 300 W - 4 baterías de 250 mAh
Módulo de sensores de mapeo	- LIDAR de 360º - Receptor GNSS - 2 sensores de ultrasonido - Video Cámara HD
Módulo de sensores ambientales	- Sensor de PH - Sensor OPR (oxidación) - Sensor de conductividad - Sensor de concentración de oxígeno disuelto (DO)
Módulo de comunicaciones	- Receptor WiFi 2.4G - Receptor auxiliar WiFi 5G - Transmisor 4G-LTE
Módulo de propulsión	- 2 motores fueraborda de 25 Kg de empuje

Además, en los últimos meses se han fabricado otros drones de pequeño tamaño y mismas características, de los cuales, dos están en las instalaciones de la Escuela Técnica Superior de Ingeniería de Sevilla.

(a) (b)

Figura 1.5 Imágenes de los nuevos ASV en las instalaciones de la US.

Con todo esto, tendremos un vehículo capaz de realizar misiones teledirigidas o computadas desde la misma inteligencia del vehículo, especificando coordenadas geográficas o *waypoints* que el vehículo seguirá gracias al control de bajo nivel de los motores. Aguas arriba del esquema de control, tendremos un generador de trayectorias (o interpolador) que traducirá las consignas de *waypoints* en valores de posición y velocidad en cada instante de tiempo. Estos *waypoints* son puntos concretos y ordenados en el mapa disponible.

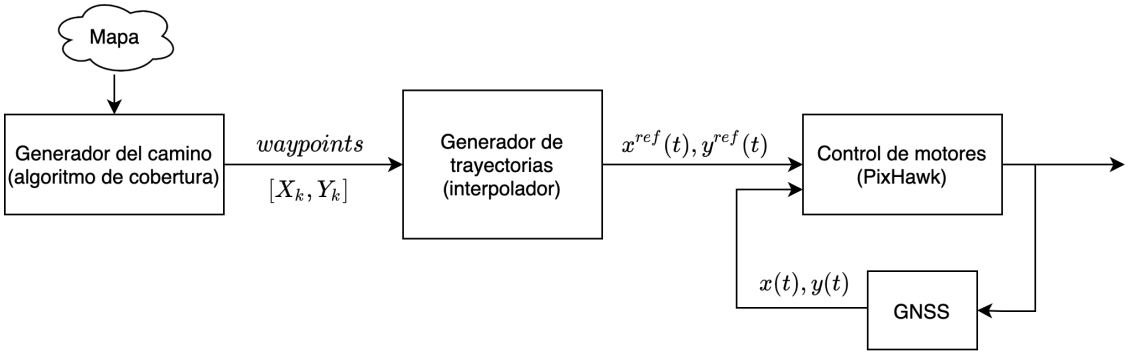


Figura 1.6 Esquema de control del ASV.

1.3 Objetivos del proyecto

Este trabajo se centra en el módulo de más alto nivel, que recibe un mapa del lago y traza los waypoints siguiendo una estrategia más o menos eficiente para poder cubrir toda la superficie navegable, de modo que se pueda recoger toda la información posible de todas las zonas, especialmente de las que presentan grandes masas de

florecimientos de cianobacterias.

Los **objetivos principales** de este proyecto son:

1. Recorrer el estado de la técnica en el campo de la exploración mediante *Reinforcement Learning*.
2. Desarrollo de un algoritmo de exploración adaptativo basado en las técnicas de *Reinforcement Learning*.
3. Un primer estudio basado en dichas técnicas de inteligencia artificial para las tareas de monitorización ambiental en entornos lacustres que demuestre que es posible encontrar soluciones buenas al problema planteado.
4. Presentar los parámetros de aprendizaje y ajuste típicos de las técnicas utilizadas y realizar una comparativa de cómo interviene cada una en el desempeño final.

El diseño del algoritmo estará orientado a obtener un método autónomo que siga un criterio de eficiencia. Como **requisitos funcionales**, se han marcado las siguientes características del algoritmo desarrollado:

- **Capacidad de cobertura**: el algoritmo deberá intentar generar un patrón que cubra cuanta más área del lago mejor para evitar que haya zonas ciegas de medidas.
- **Redundancia temporal**: el algoritmo poseerá cierto nivel de redundancia temporal, es decir, que se valorará el *repasar* zonas ya visitadas para obtener datos actualizados de esos lugares.
- **Capacidad para rechazar acciones ilegales**: el algoritmo deberá rechazar acciones que intenten llevar al vehículo a zonas no navegables o prohibidas del mapa.
- **Resolución y entorno variable**: el algoritmo desarrollado debería ser capaz de adaptarse a nuevas condiciones ambientales y a trabajar de forma similar independientemente de la morfología y resolución del escenario lacustre. En este sentido, el algoritmo podría servir para otros contextos más allá del lago Ypacaraí.
- **El algoritmo será parametrizable**: se podrá configurar los parámetros relacionados con el aprendizaje.

Se deja para más adelante la incorporación del problema multi-agente en el que se tendrá en cuenta la existencia de varios ASVs explorando simultáneamente la superficie, pero el algoritmo desarrollado en este trabajo tendrá en cuenta las futuras necesidades para facilitar una posterior implementación del paradigma multi-agente.

1.4 Motivación

En las últimas décadas la comunidad científico-técnica, en gran parte debido al desarrollo de la capacidad de computación, ha podido observar el auge de la inteligencia artificial. La inteligencia artificial, entendida como la capacidad de las máquinas para actuar de forma autónoma y racional actuando ante estímulos consecuentemente, es un campo de estudio creciente y prometedor. Vemos ejemplos del uso de la inteligencia artifical o el *machine learning* en casos paradigmáticos como *Google Alpha Zero*, donde la estrategia del algoritmo es muy superior a la habilidad de un humano[3].

Cuando tratamos con problemas complejos en la ingeniería en los que se hace difícil encontrar una solución óptima, el *machine learning* aparece como una solución buena que es capaz de lidiar con incertidumbres estadísticas, modelos incompletos de la realidad y sesgos en los datos de entrada. El *machine learning* (y todas sus técnicas derivadas, entre ellas el *Reinforcement Learning*) es, en esencia, una metodología de aprendizaje basada en la inducción del conocimiento. Estos mecanismos de aprendizaje basados en la experiencia (ya sea un proceso supervisado o no) tienen la particularidad de que permiten una flexibilidad mayor para resolver estos problemas complejos al **no necesitar la intuición humana o el conocimiento experto de los datos de entrada**.

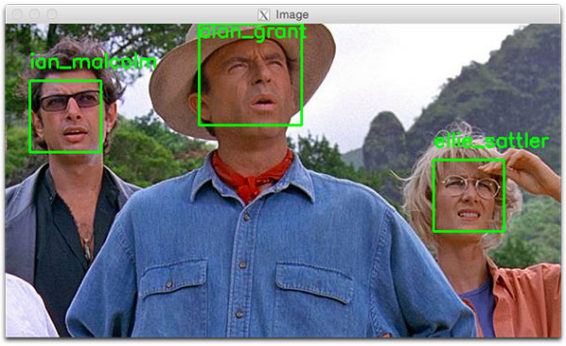


Figura 1.7 Ejemplo de aplicación de reconocimiento de caras humanas basadas en redes neuronales convolucionales, una aplicación muy extendida en la robótica actual. Fuente: *https://www.pyimagesearch.com/2018/06/18/face-recognition-with-opencv-python-and-deep-learning/*.

Esta característica las convierte en técnicas extraordinariamente aptas para los problemas de la robótica: en este campo se trabaja con muchas incertidumbres, entornos dinámicos y estocásticos con múltiples dimensiones. La imposibilidad de crear algoritmos de comportamiento deterministas que lidien con todos estos aspectos al mismo tiempo y encima busquen la optimalidad, abre la puerta a la necesidad del aprendizaje autónomo para realizar tareas como el *pathplanning*, la exploración de

[3] En octubre de 2015, la inteligencia artificial AlphaGo de Google derrotó al campeón mundial Lee Sedol del desafiante juego Go, marcando el camino del desarrollo de este tipo de algoritmos. *https://www.bbc.com/news/technology-35761246*

entornos desconocidos o el problema de colaboración entre robots.

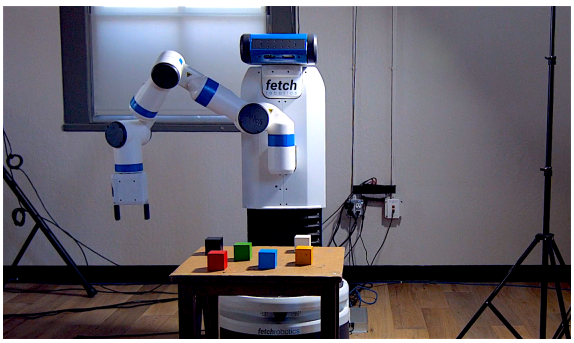


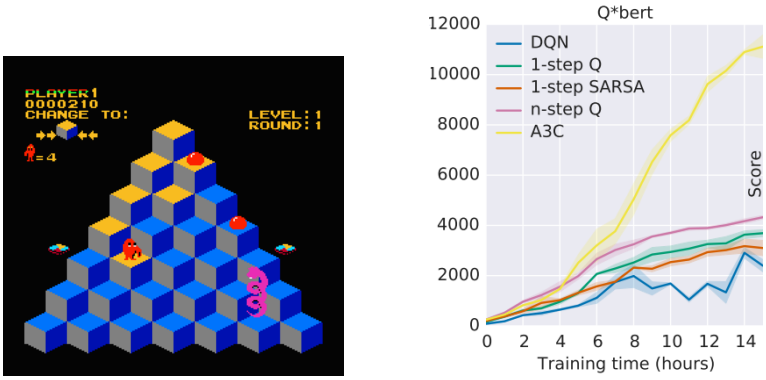
Figura 1.8 Ejemplo de uso del Reinforcement Learning para tareas hiper-complejas como la manipulación de múltiples objetos con brazos robots. Fuente: *FetchRobotics*.

El Reinforcement Learning (concretamente el *Deep* Reinforcement Learning) surge como una posible solución a estas condiciones difusas: bajo el paraguas de estas técnicas, el problema queda definido como un conjunto de reglas que darán forma al entrenamiento a través de la ley de recompensa que todos los algoritmos RL poseen. Una ventaja clara de usar estos algoritmos es que podemos conseguir resultados muy aceptables sin necesidades de modelar el problema de forma muy exacta o considerar las diversas restricciones que tiene todo algoritmo de optimización. Simplemente modelaremos *qué esperamos del algoritmo* a través de valorar su comportamiento mediante la ley de recompensa.

Nos aprovechamos entonces de la enorme flexibilidad de estas técnicas para poder utilizar ambientes variables, estocásticos o desconocidos a priori como es el caso del lago Ypacaraí, donde el mapa geográfico es conocido previamente pero no así el estado de las colonias de cianobacterias.

El razonamiento pudiera extenderse al caso multi-agente fácilmente ya que el paradigma de entrenamiento es análogo. Una lección importante que se puede sacar de los experimentos realizados por Google Deep Mind con los juegos Atari y reflejados en [4] es que el Reinforcement Learning y concretamente el Deep Q-Learning es capaz de lidiar con ambientes y dinámicas temporales radicalmente distintas y seguir dando unos resultados extraordinarios. Este trabajo intentará aprovecharse de los reseñables descubrimientos de esta investigación (entre otras) para encontrar la optimalidad en la tarea del *patrolling problem* que más adelante desarrollaremos.

Así, esta investigación intenta entender el problema de la exploración del lago Ypacaraí como una especie de entorno parecido a un videojuego Atari que tiene que entrenarse para obtener la máxima recompensa, fijada según se considere una acción mejor que otra y que servirá para poder incluir fenómenos y objetivos en un futuro y que aquí no se han contemplado.



(a) (b)

Figura 1.9 Escenario (a) y resultado de entrenamiento (b) del juego Atari Q-bert en [5] mediante varias técnicas. El juego Q-bert recuerda un poco al problema del lago Ypacaraí, puesto que el objetivo es cubrir todas las casillas de manera uniforme.

Capítulo 2: Estado del arte

2.1 Los problemas de cobertura

Definiremos *el problema de la cobertura total* (*Complete Coverage Path Planning - CCPP*) como el desafío, dado un espacio continuo o discreto transitable por un agente, de recorrer de forma efectiva total (o casi total) su área atendiendo a criterios de superficie total cubierta, redundancia mínima u optimalidad en las trayectorias. La cobertura total de un área es un problema bastante cotidiano y recurrente en el ámbito de la robótica en general. Un ejemplo puede encontrarse en los robots domésticos de limpieza (tipo Roomba) en la que la misión se reduce a recorrer todas las superficies de un mapa precontruido, de forma equitativa, sin olvidarnos de ninguna zona. En estos casos, hablaremos de una *cobertura total*, puesto que el objetivo es que no quede ninguna zona del mapa sin visitar (considerando un entorno del robot como parte de ese área que consideramos visitada) y en las que todas las zonas tienen **la misma importancia**.

Toda la rama de problemas de cobertura total tienen en común que existe un estado terminal del problema, en el que todas las zonas han sido visitadas al menos una vez. Una vez se alcance esta condición, tendremos que el mapa queda resuelto. Ya dependerá de la formulación y naturaleza del problema si se considera que el objetivo es que se alcance este estado final habiendo repetido menos veces ciertas zonas (condición de mínima redundancia) como ocurre en [6] o en [7].

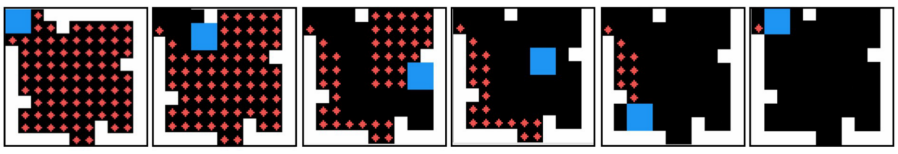


Figura 2.1 Secuencia de cobertura total en [6].

Un problema análogo al de la cobertura total es el *patrolling problem* o *problema*

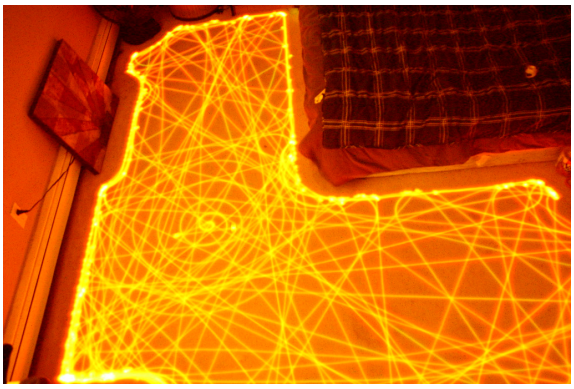


Figura 2.2 Trayectoria real seguida por un robot Roomba obtenida mediante exposición prolongada.

del vigilante. En este caso, no interesa tanto el cubrir todas las zonas de un mapa alguna vez sino el no dejar zonas del mapa sin cubrir *durante demasiado tiempo.* Podemos hacer una analogía con un vigilante de un museo: el vigilante tiene que pasar por todas las zonas del museo y además debe revisitar cada cierto tiempo de forma que ninguna zona quede olvidada. En este problema podemos diferenciar entre unas zonas más interesantes que otras, que habrá que visitar con más asiduidad que otras de menos importancia. Este problema está bien definido y se aborda en [8] o en [9].

El problema del vigilante será especialmente interesante cuando queramos ejercer tareas de supervisión y monitorización de un entorno a lo largo del tiempo y con condiciones de mapa dinámicas. Es nuestro caso, el de la monitorización del Lago Ypacaraí, en la que visitar todo el Lago es tan importante como tener información redundante (hasta cierto punto) y uniforme.

2.2 La definición del entorno

La definición del entorno en estos dos problemas es fundamental desde un principio, pues establecerá cómo el robot o agente puede moverse y definirá las condiciones de contorno y optimalidad.

Muchos trabajos académicos que cubren alguno de los problemas anteriores (o al menos casi todos los que utilizan Reinforcement Learning o similares) consideran los mapas como espacios bidimensionales discretizados para poder simplificar la naturaleza del mismo. La discretización se realiza generalmente obteniendo un equivalente en celdas cuadradas del mapa real, esto es, considerando que el robot puede ocupar una celda concreta [7] o varias [10] del mapa discreto y que cada uno de esos espacios abarca una cantidad de área real. Cuanto más pequeña cada celda, más representativo es el mapa de la realidad pero más computacionalmente desafiante será el algoritmo.

El escenario puede/suele estar compuesto por zonas de distinta naturaleza: zonas

transitables, zonas no transitables (obstáculos) y zonas sin visitar o desconocidas, entre otras. En [6] se distinguen dos tipos de casillas visitables: casilla ya visitada y casilla sin visitar (enfoque típico del problema de CCPP). Otro enfoque más orientado al patrolling problem es el de considerar cada celda con un valor de interés asociado variable con el tiempo y con las visitas que recibe, como ocurre en [11] y en [8].

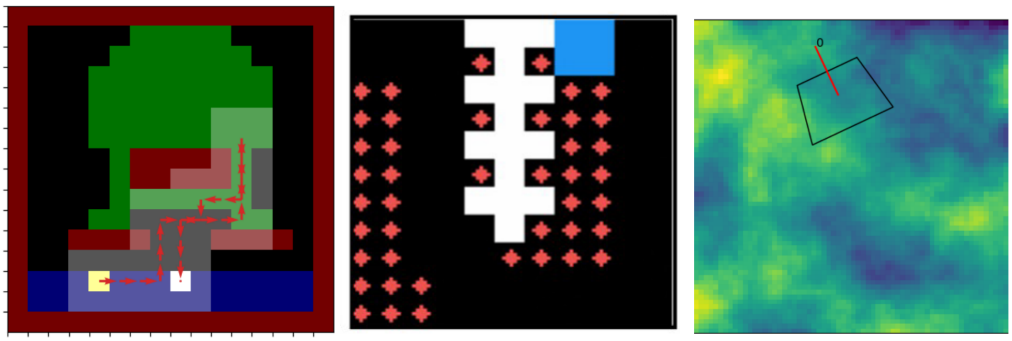


Figura 2.3 Representación del estado en [7],[6] y [8] respectivamente..

De estos dos problemas, surgen variaciones en las que se pueden incluir incertidumbres en el mapa o directamente que el mapa no se conoce a priori como en [10], en el que, gracias a un algoritmo de SLAM (*Simultaneous Localization And Mapping*), se ejecuta un *pathplanning* en línea para reducir la energía dedicada en la cobertura y descubrimiento de zonas desconocidas. De esta forma, surgen dos tipos de entornos [12]:

- Los deterministas, en los que conoceremos el valor del estado (significado como la recompensa en el caso del RL) dado un estado concreto. Es el caso de [6].
- Los estocásticos, en los que el valor del estado tiene una componente aleatoria como ocurre en [9].

2.3 El agente y el estado

El agente forma parte fundamental también de la definición del problema. Distinguiremos las dos situaciones posibles: caso mono-agente (en el que solo hay un robot en el entorno) o caso multi-agente (en el que existe un número de robots variables que comparten el mismo entorno). Este último caso introduce la idea de que en el algoritmo se tendrá que tener en cuenta ya no solo la cobertura del mapa sino la posibilidad de colisiones entre agentes. Este problema multi-agente es tratado en [13] y en [11].

Otro asunto es el cómo el agente o agentes observan el entorno. En ocasiones se tiene acceso al estado completo del entorno en todo momento [7][11][6] ya que

el mapa es fijo y las condiciones son estáticas, no obstante, pudiera ocurrir que la información del estado completo sea desconocida y solo tengamos acceso a una parte observable (más o menos representativa) que es lo que ocurre en [8]

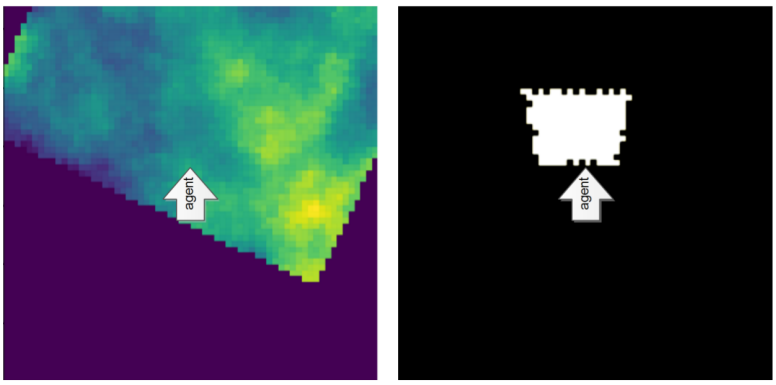


Figura 2.4 Estado como observación parcial del entorno en [8]. Un dron con una cámara sobrevuela un mapa con distintos valores de importancia para cada píxel (color) y el estado observado es el fragmento de mapa que alcanza a captar..

Qué entendemos por estado es también un asunto importante puesto que definirá la información que nutre el algoritmo a la hora de buscar las acciones óptimas para acometer el objetivo. El estado puede ser cualquier información que defina *cómo se encuentra el entorno y cómo nos encontramos respecto a él*. Por ejemplo, un estado posible en el caso de un problema de CCPP podría ser una imagen RGB del mapa en el que se pueda distinguir la posición del agente y los obstáculos. Otro estado menos representativo del entorno podría ser simplemente la posición del agente en el mapa, lo que no sería tan representativo de la situación real del agente en el medio.

Las investigaciones suelen utilizar esta primera representación por ser compacta y bastante eficiente (y normalmente accesible) aunque no siempre es así como puede verse en el caso de la figura 2.4.

2.4 Resumen de investigaciones previas

Se expondrá en la siguiente tabla un resumen de los artículos más reseñables que tratan alguno de los problemas anteriormente descritos junto con su metodología, hiperparámetros más importantes y descripción del estado y el entorno.

Referencia	Escenario	Estado	Recompensa	Agente	Algoritmo RL	Función de estimación	Hiperparámetros
Lakshmanan, 2020	- **Objetivo**: CCPP - 120 escenario diferentes tipo **gridmap**. - Tres tipos de casillas: visitadas, sin visitar y obstáculo.	- **Imagen RGB** del escenario completo (Fully Observable)	- **Recompensa cte.** por casilla nueva visitada (+1) y por completar la tarea (+50) - **Penalización cte.** por cada movimiento (variable, dependiente de la geometría) y acción ilegal (-0.5).	- **Mono-agente** de geometría variable (Tetrómino-Like). - 11 acciones: 4 de **traslación** y 7 de **transformación de la forma**.	- Async. Advantage Actor Critic with Experience Replay (**Off-policy A3C**).	**Secuential Neural Network:** - Convolutional Layer 8x8x32 (ELU) - Convolutional Layer 8x8x32 (ELU) - Convolutional Layer 8x8x32 (ELU) - Fully Connected 512 (ELU) - LSTM 256 (Sigmoid)	- Learning Rate = 0.0001 - Disctount factor = 0.9 - Exp. Buffer size = 20.000 - Epochs = 10.000
Liu, 2019	- **Objetivo**: CCPP multi-agente (drones) con reducción de la energía de movimiento. - Escenario continuo 3D sin obstáculos.	- **Vector de conjunto** del espacio de estado de cada agente (posición XY, altura Z y área de cobertura D)	- **Recompensa cte.** por no solapar áreas de observación al final (+10) y por completar el coverage (+100). - **Penalización cte.** cuando dos agente colisionan (-100) y con cada movimiento (-1).	- **Multi-agente.** 3 drones con cámara cenital caracterizados por su posición y área de cobertura. - **6 acciones**: up, down, north, south, east & west.	- **Advantage Actor Critic.**	**Gradient descent method & no Neural Network**	- Learn. Rate Actor = 0.0025 - Learn. Rate Critic = 0.05 - Epochs = 5000
Piciarelli, 2019	- **Objetivo**: Patrolling (drones) con zonas de interés variable. - Escenario tipo **gridmap** con valor de interés de cada celda.	- **Fragmento del mapa visible para el agente** (cámara) y posición del agente en el mapa en dos matrices. (Partially Observable)	- **Recompensa variable** en función de un mapa de interés dinámico. Visitar una casilla reduce su interés temporalmente, y se regenera con el tiempo. Recompensa entre 1 (no visitada y de gran interés y -0.9 con mínimo interés).	- **Mono-agente** con una cámara que delimita la información que capta. - **12 acciones discretas** posibles: 6 movimientos de traslación, 4 movimientos de cámara y 2 para zoom in/out.	- **Double Deep Q Learning** con soft-update de la *función de target*.	**Secuential Neural Network:** - Convolutional Layer 8x8x16 (ReLU) - Fully Connected Layer 1024 (ReLU) - Fully Connected Layer 1024 (ReLU) - Fully Connected 12 (Linear)	- Num. of steps = 20
Theile, 2020	- **Objetivo**: CCPP mono-agente (dron) de ciertas zonas con reducción de la energía de movimiento. - Escenario **gridmap** con zonas de despegue, aterrizaje, obstáculos y zona de interés.	- **Imagen de 5 canales** del escenario completo (Fully Observable) con las distintas zonas (5 posibles) del mapa.	- **Recompensa constante:** cuando cubrimos una celda nueva por 1ª vez. - **Penalización constante:** cuando se solicita una acción ilegal, con cada movimiento realizado y cuando se acaban los movs. sin haber llegad a la zona de *landing*.	- **Mono-agente** con una entorno de observación al rededor. - **5 acciones discretas** posibles: 4 movimientos (N,S,E,W) y aterrizaje.	- **Double Deep Q Learning** con soft-update de la *función de target*.	**Secuential Neural Network:** - Conv. Layer 16x16x5 (ReLU) - Conv. Layer 16x16x16 (ReLU) - Fully Connected Layer 256 (ReLU) - Fully Connected Layer 256 (ReLU) - Fully Connected Layer 256 (ReLU) - Fully Connected Layer 5.	- Batch Size = 10.000 - Minibatch Size = 128 - Learn. Rate = 0.001 - Discount factor = 0.95 - Epochs = 10.000 - Target Update Rate = 0.005

Referencia	Escenario	Estado	Recompensa	Agente	Algoritmo RL	Función de estimación	Hiperparámetros
Shah, 2020	- **Objetivo**: CCPP mono-agente (Cont. Area Sweeping) con interés en cubrir eventos aleatorios. - Escenario **gridmap (20x20)** con casillas de dos tipos: obstáculo y libre.	- **Mapa del escenario** (Fully Observable), posición del robot e incertidumbre creciente en el tiempo de cada celda no visitada en ese instante.	- **Recompensa** dependiendo del incremento de detecciones de los eventos estocásticos medios por segundo en cada instante.	- **Mono-agente** con 4 **acciones discretas** posibles: (N,S,E,W).	- **Double Deep R Learning** (variante DDQL) con soft-update de la *función de target*.	**Secuential Neural Network:** - Conv. Layer 32x32x5 (ReLU) - Max Pool 2x2 - Conv. Layer 16x4x4 (ReLU) - Conv. Layer 16x4x4 (ReLU) - Fully Connected Layer 500 (ReLU)	- Learn. Rate = 0.0001 - Discount factor = 0.95
Xiao, 2020	- **Objetivo**: CCPP multi-agente (Cont. Area Sweeping) con interés en la incertidumbre. - Escenario **gridmap)** con casillas con valor de interés asociado.	- **Mapa de interés temporal del escenario** (Fully Observable), y posición de cada agente.	- **Recompensa** como función de la posición de cada agente mareada sobre el mapa de interés compartido. Si un agente visita una celda, *roba* el interés de esa celda, que va creciendo cuasi-linealmente con el tiempo.	- **Multi-agente** con 8 **acciones discretas** posibles: (N,S,E,W,NE,NW,SE,SW).	- **Q Learning**	-	- Learn. Rate = 1 - Discount factor = 0.8
Niroui, 2019	- **Objetivo**: CCPP mono-agente. - Escenario **occupancy grid** desconocido a priori (se va creando on-line con SLAM)	- **Mapa de puntos** que representan las fronteras del mapa (entre zona conocida y desconocida) candidatas a ser exploradas y posición del agente.	- **Recompensa** no discreta de un epoch completo dependiente de la información recopilada a lo largo de todo el episodio para minimizar la distancia recorrida.	- **Multi-agente** con espacio de acciones variable. Cada acción se corresponde con el movimiento a un punto de frontera entre zona conocida y zona sin explorar.	- **Asynchronous Advantage Actor Critic (A3C)**	**Múltiple Secuential Paralell Convolutional Neural Network.** Se incorpora una capa final de LSTM.	- Learn. Rate = 0.0001 - Discount factor = 0.99 - Epochs = 3570
Adepegba, 2016	- **Objetivo**: CCPP multi-agente basado en mapa de Voronoi y función de riesgo de colisión. - Escenario **contínuo.**	- **Posición y velocidad de cada agente** (espacio de estados de cada agente)	- **Recompensa** calculada como el error entre cada posición de cada y agente y el centroide de su mapa de Voronoi asociado (solución óptima).	- **Multi-agente** con espacio de acciones continua basado en una dinámica de doble integrador (acción como aceleración de cada dron).	- **Advantage Actor Critic (A2C)**	- **Neural Network (input + hidden + output)**	- Learn. Rate = 0.0009 - Discount factor = 0.99 - Epochs time = 350 segs.

Capítulo 3: El problema del vigilante

3.1 El problema del vigilante en el lago Ypacaraí

En el contexto del lago Ypacaraí se contempla una variación ligeramente distinta de los casos tratados en la literatura. Cuando hablamos de la exploración del lago Ypacaraí hablamos de un problema con el escenario conocido previamente, por lo que muchos de los enfoques expuestos en las investigaciones anteriores son demasiado extensivos. En el caso que nos ocupa, el problema del patrolling toma una perspectiva más cercana a la esencia de la información obtenida más que al área total recubierta que es la métrica principal del problema del *Complete Coverage Path Planning*.

Desde la perspectiva de la información recogida por los sensores el objetivo del ASV que explora el lago es doble:

- Cubrir el lago de forma completa, visitando en algún momento todas las zonas, para tener información completa del estado del algo.
- Revisitar aquellas casillas que hace tiempo que no visitamos para tener una imagen actualizada de la métrica del lago.

Aquí se introduce el concepto de *información redundante* o *redundancia útil*: la información redundante puede ser útil puesto que afianza el conocimiento que tenemos sobre el estado del lago, pero puede ir en contra de la eficiencia de la cobertura ya que visitar muchas veces una zona es perder el tiempo (redundancia inútil).

Por ello, el algoritmo que se desarrollará será más parecido al problema del *patrolling* que al del CCPP: será necesario cubrir todo el lago pero, al tratarse de un algoritmo no episódico (no existe un fin de la exploración como tal) tendremos que revisitar ciertas zonas del mapa. La preferencia está clara: se preferirá revisitar las zonas que más tiempo hayan pasado sin recibir visita alguna.

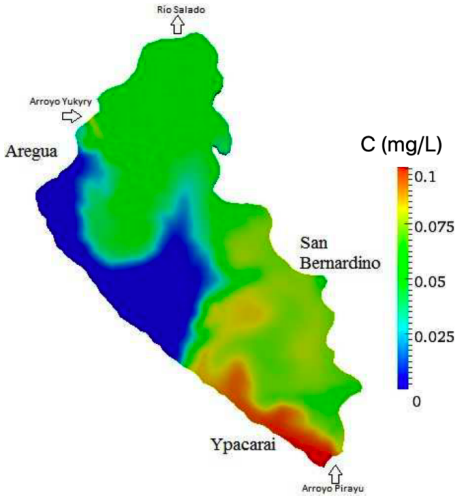


Figura 3.1 Mapa de distribución espacial de un contaminante en ausencia de viento en el lago Ypacaraí. Fuente: [14].

De esta forma, el problema a abordar será el de reducir el tiempo medio de visita de cada zona del lago: una zona que se visita asiduamente será una zona bien cubierta y una zona que no se ha visitado o que hace muchísimo tiempo que se visitó estará mal cubierta.

3.2 Importancia relativa del mapa

Está claro que cubrir completamente el lago asumiendo que la importancia de todas las zonas es la misma pudiera resultar una condición muy poco realista. Está claro que existen zonas de mayor interés en relación a la necesidad de cobertura: a nivel biológico, los florecimientos de bacterias surgen en zonas cercanas a las ciudades y en las zonas de vertidos urbanos y agropecuarios y es necesario incidir mucho más en esos lugares, como se ve en la figura 3.1.

De esta forma, distinguiremos este nuevo caso del anterior con que el tiempo medio entre visita y visita de una casilla debe ser menor en las zonas de mayor importancia y en las de importancia mínima debe ser menor.

Este problema entraña mayor dificultad que el anterior, puesto que existen múltiples soluciones y ante perfiles de importancia distintos, es difícil encontrar un enfoque único que de solución a todos de manera sistemática.

3.3 Reinforcement Learning en el problema de exploración

Como resultado de esta investigación bibliográfica inicial, podemos decir que en los últimos años se han experimentado éxitos relativamente satisfactorios cuando se

trata de encontrar soluciones óptimas a los problemas de *coverage* y *patrolling*. El desarrollo de las técnicas de Deep Q Learning y Actor Critic (en sus dos variantes, A2C y la más moderna, A3C) por parte de autores como [4], viene a abrir un camino prometedor para las tareas de exploración.

Una ventaja importante que se ha encontrado en las metodologías estudiadas para resolver el problema de exploración del lago, es que el Reinforcement Learning, concretamente en los algoritmos de tipo Actor-Critic y Q-Learning, es capaz de trabajar en el espacio de las acciones. Esto significa que el resultado del entrenamiento nos da, directamente, una función que mapea el estado en la acción a realizar. A diferencia de métodos metaheurísticos del tipo genéticos (muy utilizados) como los propuestos por [3] y que intentan encontrar solución a un problema complejo (tipo NP, en ocasiones), los algoritmos de Reinforcement Learning pueden resultar convenientes cuando tratamos escenarios dinámicos o con una fuerte componente estocástica o de estado no observable [15]. Concretamente, cuando hablamos de algoritmos de Q-Learning o Actor-Critic, no necesitamos un modelo de entorno, lo que lo convierte siempre en candidatos capaces de lidiar con este tipo de incertidumbres estructurales de escenarios y sistemas dinámicos.

En el extraordinario trabajo de [5] y [4], se demostró que el Deep Q-Learning es capaz de entrenar agentes para obtener niveles competitivos en el desempeño de videojuegos tipo Atari. La versatilidad del algoritmo permitió entrenar a dichos agentes con una variabilidad muy grande de escenarios y dinámicas, casi siempre estocásticas. Una ventaja en este algoritmo reside también en la sencillez a la hora de aplicarlo a dichas condiciones variantes: en [5] el estado será una captura RGB (o varias) en cada intante o instantes previos del juego.

En algoritmos metaheurísticos, el problema es tratado generalmente como una caja negra, en el que solo se tiene en cuenta el desempeño final de cada generación. Así, estaremos desaprovechando información del desempeño *a lo largo del proceso* puesto que no tenemos en cuenta el gradiente de mejora sino al final. Si bien este gradiente pudiera resultar ruidoso [15], suele tener mucha más información sobre la bondad de las acciones tomadas individualmente que el índice de fitness en los algoritmos evolutivos, evaluado al final de cada episodio para cada individuo.

Dentro de la literatura del Reinforcement Learning, en el ámbito de los algoritmos de exploración de entornos parece que existen dos metodologías preferidas: por un lado, tenemos el uso del Double Deep Q-Learning que es un algoritmo de tipo *Value Optimization*. Otra variante dentro del Deep Q-Learning digna de estudio está en el Dueling Network Q-Learning, que viene a mejorar la convergencia del anterior y supuso un avance importante en los resultados obtenidos por Google Deep Mind en el año 2016 en el entrenamiento de agentes para el caso de los juegos Atari [16].

Estos algoritmos, sobradamente efectivos, se vieron ensombrecidos con el surgimiento de estrategias como el Actor-Critic [17], y cada vez es más frecuente ver que la resolución y optimización recae sobre estrategias de tipo *Policy Optimizacion*. El método más moderno de este subconjunto (Asynchronous Advantage Actor Critic) no solo mejoró en resultados a sus anteriores, sino que resultaba útil para espacios de

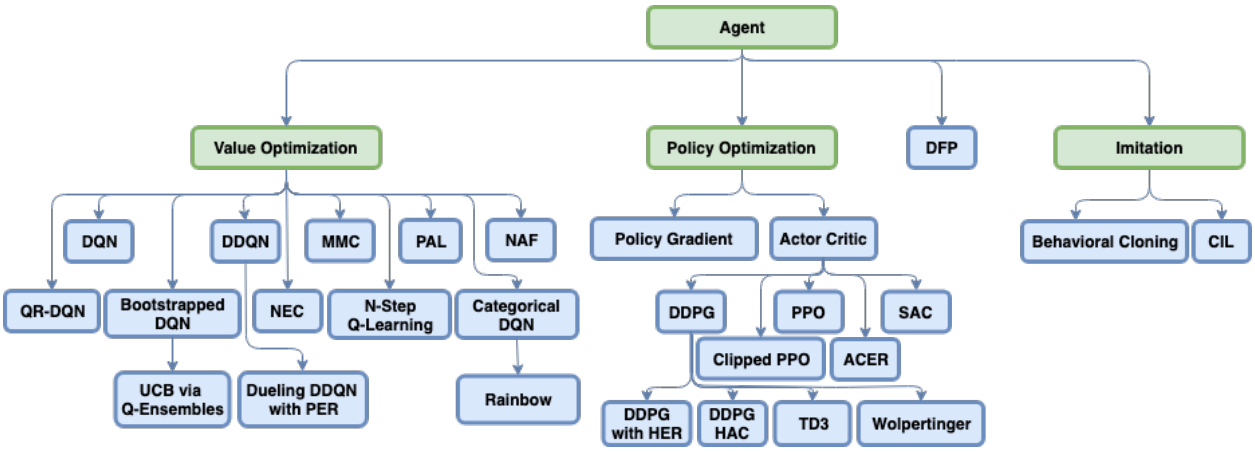


Figura 3.2 Familia de algoritmos dentro del Reinforcement Learning..

acción discretos y continuos por igual. Incluso, su implementación paralela supone una mejora extraordinaria a la hora de mejorar los tiempos de entrenamiento y la complejidad computacional [12].

En este trabajo se plantea la viabilidad de encontrar soluciones buenas a estos dos problemas anteriores mediante las técnicas desarrolladas en [4]. Al tratar al agente y su escenario como una especie de videojuego y mediante el diseño de una ley de recompensa adecuada, se plantea que los buenos resultados obtenidos en las investigaciones de Google Deep Mind puedan contribuir a desarrollar un planificador de trayectorias para este caso con buenos resultados.

Capítulo 4: Metodología

4.1 Reinforcement Learning

El Reinforcement Learning (RL) es un proceso de aprendizaje en el que un agente es capaz de *aprender* a realizar una tarea mediante un mecanismo de condicionamiento basado en recompensa. Esta idea es bastante intuitiva y no resulta difícil para el lector encontrar ejemplos de aprendizaje por refuerzo en la sociedad actual: desde el condicionamiento cognitivo clásico en el caso del perro de Pavlov [1] hasta la teorías conductivistas de Burrhus F. Skinner nutren la base filosófica del aprendizaje por refuerzo. A grandes rasgos, el aprendizaje por refuerzo consiste en aprender mediante la observación de la relación causal entre una acción o estímulo y su respuesta en forma de recompensa o castigo.

Esencialmente, todo algoritmo de Reinforcement Learning funciona de la siguiente forma [12]:

1. El agente interactúa con el entorno realizando una acción.
2. El estado de ese agente cambia en el contexto del escenario.
3. Dependiendo de si ese cambio es mejor o peor para el comportamiento objetivo que queremos obtener, proporcionamos al agente una recompensa o un castigo.
4. El agente evalúa la causalidad entre la acción y el estado y la recompensa. A la larga, comenzará a asociar las acciones en un estado dado como potencialmente buenas o malas basándose en la experiencia previa.

[1] *https://en.wikipedia.org/wiki/Classical_conditioning*

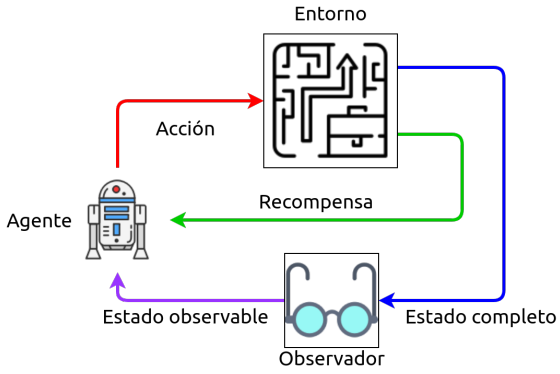


Figura 4.1 Esquema de interacción de elementos en el Reinforcement Learning..

En este problema de aprendizaje por refuerzo distinguimos distintos elementos:

- **Agente**: El agente es el sujeto que debe tomar decisiones inteligentes (desde la perspectiva de la optimalidad) y el que realiza el entrenamiento. Tiene disponible un conjunto de acciones para interactuar con el escenario.

- **Acciones**: Las acciones son las formas que tiene el agente de interactuar con el entorno. El conjunto de acciones posibles son todas aquellas que se pueden realizar en un estado dado.

- **Política**: La política es una función que define el comportamiento o estrategia del agente. Es, esencialmente, la función que mapea el estado del agente en la acción que va a realizar. Queda simbolizada por $\pi(s)$, donde s es el estado.

- **Función de valor de estado**: La función de valor indicará cómo de bueno es el estado en el que se encuentra el agente en el contexto de la recompensa acumulada futura que podría recibir partiendo de ese mismo estado. Se notará como $V(s)$, siendo s el estado del agente.

- **Función de valor estado-acción**: La función de valor estado-acción indica cómo de buena es cada acción a posible dado un determinado estado s en el contexto de la recompensa acumulada futura. Se notará como $Q(s,a)$

- **Modelo**: El modelo es la representación del entorno para el agente. Cómo vemos el entorno y el estado. Un algoritmo sin modelo del entorno (*model-free*) solo recibirá la recompensa y aprenderá puramente mediante prueba y error.

En el Reinforcement Learning, el objetivo principal, es maximizar la recompensa a largo plazo. Esto quiere decir que los procesos de entrenamiento por refuerzo ponen el énfasis no tanto en obtener una recompensa inmediata alta gracias a unas acciones buenas a corto plazo sino que a lo largo de todo el tiempo de acción del agente (denominado episodio) la recompensa sea máxima **al final**.

En el caso de la exploración del lago Ypacaraí se busca que el aprendizaje permita escoger no solo las mejores acciones que devuelvan una recompensa instantánea alta,

sino maximizar, tras haber completado un número N (generalmente un valor alto) de movimientos, la recompensa.

4.2 Proceso de Decisión de Markov

Para poder definir **matemáticamente** el problema del Reinforcement Learning se usará una estructura denominada **Proceso de Decisión de Markov** (MDP). Este marco matemático permite modelar un problema RL para su resolución mediante algoritmos de optimización como el Q-Learning.

Un MDP está basado en la hipótesis de causalidad de Markov. Cuando esta hipótesis se cumple, tenemos que los estados y recompensas futuras solo dependen del estado actual y no de los estados pasados, esto es, que *el futuro es independiente del pasado*. Esta hipótesis es, en ocasiones, restrictiva y no se aplica completamente, sobretodo cuando tenemos procesos que son puramente estocásticos, no obstante, permite trabajar bien con muchos problemas. En el caso de la exploración del lago Ypacaraí podríamos sostener que se cumple, puesto que habiendo visitado un conjunto de celdas y dada una posición del agente, el estado futuro solo depende del estado actual (y de la acción).

En el MDP distinguimos distintos elementos:

- El estado en el que se encuentra el agente s. Este estado, como se verá en el capítulo siguiente, puede tener distintas formas.
- El conjunto de todos los estados posibles S.
- El conjunto de todas las acciones posibles A.
- Una acción concreta a.
- La probabilidad de pasar de un estado s a otro s' realizando la acción a: $P^a_{ss'}$ (la probabilidad medida como lo conveniente que es pasar de un estado a otro mediante esa acción).
- La probabilidad de recibir una recompensa pasando de s a s' realizando la acción a: $R^a_{ss'}$
- El factor de descuento γ que controla la importancia de una recompensa en el siguiente instante respecto a una recompensa en un horizonte de predicción. Si nuestro horizonte de acción es de N instantes, entonces tendremos que la recompensa acumulada al final del episodio (o etapa de entrenamiento) será:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^{N-1} r_{t+N} = \sum_{k=0}^{N} \gamma^k r_{t+k+1} \tag{4.1}$$

4.3 Función Q y optimalidad de Bellman

La función Q es como se conoce a la función acción-estado. Esta función definirá cómo de bueno (en términos de la esperanza matemática de tener una recompensa R) es realizar una acción a en un estado s dada una política π:

$$Q(s,a) = \mathbb{E}_\pi[R_t|s_t = s, a_t = a] = \mathbb{E}_\pi[\sum_{k=0}^{N} \gamma^k r_{t+k+1}|s_t = s, a_t = a] \tag{4.2}$$

Ahora, podemos acudir al *Principio de Optimalidad de Bellman* para *solucionar* el MPD planteado. Solucionar un MPD no es más que encontrar la política $\pi(s)$ que proporciona la máxima recompensa a lo largo del tiempo.

La ecuación de Bellman definirá la función V como:

$$V_\pi(s) = \sum_a \pi(s,a) \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma V^\pi(s')\right] \tag{4.3}$$

Asimismo, puede definirse la ecuación de Bellman para la función Q como:

$$Q_\pi(s,a) = \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma \sum_{a'} Q^\pi(s',a')\right] \tag{4.4}$$

Y se define la ecuación de optimalidad de Bellman como:

$$V^{optimo}(s,a) = \max_a \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma \sum_{a'} Q^\pi(s',a')\right] = \max_a Q_\pi(s,a) \tag{4.5}$$

La resolución entonces del MPD consiste en encontrar el valor de V^{optimo} verdadero, es decir, cómo de beneficioso es cada estado (de todos los posibles) y movernos de un estado a otro, o lo que es lo mismo, encontrar la política $\pi(s)$ que maximiza la recompensa moviéndonos por los estados y las acciones que hacen $\max_a Q_\pi(s,a)$.

Para cada problema de optimización discreto existirá una ecuación por cada estado (si hablamos de la ecuación de valor de estado 4.4) o por cada estado y acción en ese estado (ecuación 4.5). Es fácil ver que, a poco que el problema sea complejo, tengamos muchos estados distintos o las acciones sean numerosas, la resolución del sistema de ecuaciones se hace complicado debido a su alto coste computacional.

Otro aspecto que dificulta la resolución de un MPD mediante la ecuación de optimalidad de Bellman, está en que es indispensable tener un modelo del sistema con el que podamos enlazar un estado con el siguiente, mediante sus probabilidades

de transición P y la probabilidad de recompensa R.

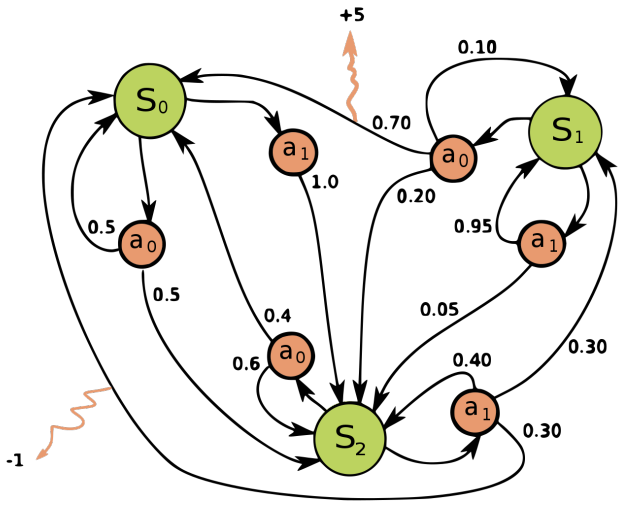


Figura 4.2 Esquema de un proceso de decisión de Markov con tres estados (círculos verdes) y dos acciones (círculos de color naranja), con dos premios (flechas naranja).

4.3.1 Algoritmo Q-Learning

Sabemos que si encontramos el valor real de $V(s)$, podemos obtener la política óptima para resolver nuestro MPD como aquella que nos indica las acciones que hacen $\max_a Q_\pi(s,a)$. De este modo, nos centramos la función $Q(s,a)$, que indicará, dado un estado concreto s, cómo de buena es cada acción a del conjunto A.

En un principio, la función Q permanece desconocida y es nuestro objetivo el descubrir su valor. Para ello, se utilizará una técnica denominada *Time Difference*. Esta técnica consiste en estimar la función Q a través de la recompensa r obtenida a cada paso:

$$Q(s,a) = Q(s,a) + \alpha \times (r + \gamma max Q(s',a) - Q(s,a)) \tag{4.6}$$

Si nos fijamos, con el *Time Difference* podemos realizar la estimación de la función Q en cada paso, observando el estado y realizando una acción a, por lo que no hace falta modelar la dinámica subyacente para estimar Q. Simplemente realizamos una serie de acciones, observamos la recompensa a cada paso y actualizamos la función Q mediante la ecuación 4.6. La técnica de *Time Difference* permite no tener que modelar el escenario y estimar el valor de los estados en base a únicamente la experiencia. Realmente la estimación mediante *TD* no es más que aplicar la técnica de gradiente descendente a la función Q con un valor de *learning rate* α.

El algoritmo del Q-Learning qudará como sigue:

Algorithm 1: Algoritmo Q-Learning

```
Q ← Tabla vacía de A × S valores;
end ← 0;
while end == 0 do
    step ← 0;
    Reseteamos el escenario y obtenemos el estado s;
    while step < step_MAX do
        Tomamos una acción a mediante una política ϵ-greedy basada en
         Q(s,a);
        Aplicamos a y adquirimos el nuevo estado del escenario s' y la
         recompensa r;
        Actualizamos la tabla Q:
         Q(s,a) ← Q(s,a) + α × (r + γ max Q(s',a) − Q(s,a));
        step ← step + 1;
        if ΔQ < Umbral then
            end ← 1;
        end
    end
end
```

Este algoritmo será ejecutado hasta que la tabla de valores de Q no cambie (o lo haga por debajo de un umbral). El resultado será una tabla de valores para cada estado y para cada acción. Para ejecutar el camino óptimo simplemente tendremos que, en cada estado, encontrar la acción a que hace $\max_a Q_\pi(s,a)$ y tomarla.

4.3.2 Ventajas y desventajas de la función Q

Una ventaja importante que tiene estimar las acciones óptimas con mediante el algoritmo de Q-Learning frente a trabajar estimando directamente la acción de valor óptima $V(s)$ reside en el hecho de que se puede utilizar una política arbitraria para escoger las acciones a tomar en cada paso y aún así encontrar el valor óptimo de la tabla Q.

Esto permite desacoplar del problema de qué política de comportamiento tiene el agente para explorar el espacio de los estados. En la literatura se puede encontrar que a este tipo de algoritmos que tienen distinta política de comportamiento (aleatoria) de la política de optimización (*greedy*) se les conoce como *Off-Policy Algorithms*. Mediante la selección de una política de comportamiento arbitraria podemos alcanzar la resolución del MPD.

No obstante, se tiene que la dimensión de la función/tabla Q aumenta por cada acción discreta que podamos realizar, puesto que ahora existirá un valor de Q por cada acción del conjunto. Esto, en problemas con alto número de acciones y estados se hace inabordable aún más incluso cuando hablamos de un rango continuo de

acciones y no discreto.

En un ejemplo paradigmático del Reinforcement Learning llamado *Frozen Lake*, tenemos un agente que pretende aprender a llegar a la posición de *goal* partiendo de la casilla de *start* pasando por un lago helado en el que hay dos tipos de casillas: heladas (que son transitables) y huecas (que hacen que el agente caiga devolviendo una penalización). En este caso, como el escenario tiene 16 casillas, se tienen 16 estados posibles. Si los movimientos posibles son arriba/abajo e izquierda/derecha, se tendrá que se puede modelar la función Q como una tabla de $16 \times 4 = 64$ valores discretos. Un número bajo de dimensiones nos permite resolver este problema de Reinforcement Learning fácilmente con el método del Q-Learning tal como se realiza en [12].

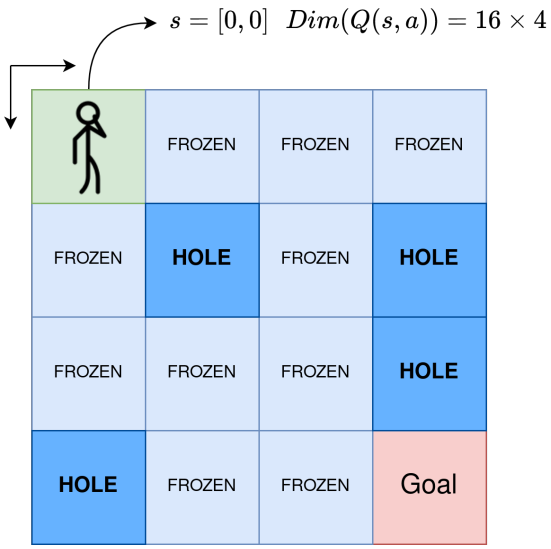


Figura 4.3 Esquema del problema del *Frozen Lake*.

Un caso en el que las dimensiones de acción y estado sean mayores requerirá de una función más fácil de manipular que una tabla de valores. La solución recae sobre funciones como las redes neuronales

4.3.3 Política ϵ-greedy

A la hora de enunciar cómo escogemos las acciones durante el entrenamiento se ha hablado de la política ϵ-greedy. El agente, para poder aprender de sus acciones, tiene que poseer un mecanismo que le permita *explorar* nuevos estados para comprobar cuál es la repercusión de sus acciones en forma de recompensa. También, todo proceso de aprendizaje debe valerse de la *explotación* del conocimiento previo para encontrar el comportamiento que lleve a la política óptima [18].

Debe existir una solución de consenso entre las dos puesto que son estrategias opuestas: **explorar** consiste en tomar caminos inexplorados y **explotar** consiste en

seguir el camino que indica el mejor incremento de la recompensa en el momento. Sin exploración podemos acabar atascados en un mínimo local puesto que no tenemos una idea lo suficientemente amplia de la relación estímulo-respuesta y sin explotación tendremos una búsqueda puramente aleatoria y será difícil encontrar un óptimo.

La estrategia $\epsilon - greedy$ (épsilon greedy) consiste en elegir estocásticamente con una probabilidad ϵ si explorar o explotar. Si exploramos, tomaremos una acción aleatoriamente muestreada de todas las acciones posibles. Si explotamos, escogeremos la acción a como la acción que tiene asociada a ese estado $\text{máx}_a Q(s,a)$.

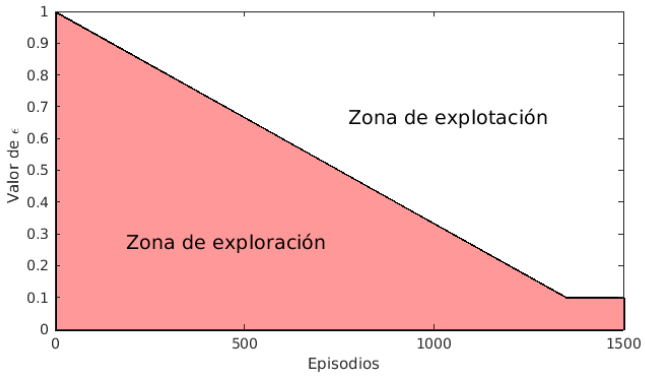


Figura 4.4 Posible valor de ϵ a lo largo de un entrenamiento como método de consenso entre exploración y explotación..

Como al principio del entrenamiento lo que se pretende es explorar y al finalizar el entrenamiento nos gustaría tener un comportamiento que genere recompensas altas, se impondrá que el valor de ϵ varía a lo largo del entrenamiento para balancear ambas actitudes. Siempre mantendremos un pequeño valor de exploración para no dejar de intentar mejorar nunca.

Finalmente, el algoritmo $\epsilon - greedy$ será:

Algorithm 2: Política $\epsilon - greedy$

```
v ← random();
if v < ϵ then
    a ← sample(A);
else
    a ← max(Q(s,A));
end
```

4.4 Estimación de funciones con Deep Learning

Se ha hablado de la dificultad que tiene el Reinforcement Learning cuando tratamos con dimensiones altas, como podría ser un mapa de celdas grande y un número grande de acciones posibles o un estado muy variable y poco adecuado para clasificar en una table (por ejemplo, el estado de un videojuego Atari en [5] es un pantallazo RGB con $40x192$ píxeles por cada canal, es decir $3 \times 40 \times 192 = 23040$ valores distintos en un pantallazo).

Se ve fácilmente que es difícil modelar una función Q en forma de tabla que contenga todas las combinaciones posibles de parejas estado-acción, por lo que se plantea la necesidad de encontrar un útil matemático capaz de modelar dicha función Q adecuadamente con un coste computacional realista. En este punto es donde aparece el *Deep Reinforcement Learning*. Por primera vez, en [5], se planteó utilizar las *redes neuronales*, concretamente las redes neuronales artificiales profundas y convolucionales para modelar la función Q.

La ventaja de uso de una red neuronal es que con un conjunto variable de parámetros podemos modelar una función compleja con una estructura de cálculo aritmético relativamente sencilla. De esta forma, tendremos que en el *Deep Reinforcement Learning* (o DRL) la función Q queda como:

$$Q(s,a) \approx Q(s,a;\theta) \tag{4.7}$$

donde θ es un conjunto de parámetros denominados $weights$ (pesos) de la red.

4.4.1 Redes Neuronales

Una red neuronal artificial es un modelo computacional basado en los modelos biologistas de las neuronas reales en las que se tiene como unidad de mínima información a la neurona. Una neurona realiza una operación de ponderación de un estímulo proveniente de otra neurona aguas arriba. La red neuronal queda conformada entonces por una interconexión múltiple entre neuronas que gracias a la evaluación ponderada de cada una de sus entradas mediante sus pesos (W) y sus biases (b) en una función no lineal ($\sigma(x)$) es capaz de modelar una función matemática cuando sus parámetros son lo suficientemente adecuados.

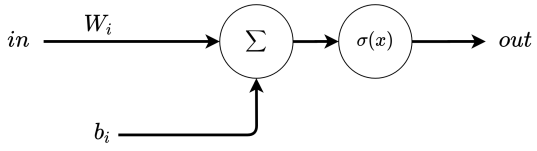


Figura 4.5 Neurona artificial.

Esta propiedad de *plasticidad* matemática permite, cuando se incorporan múltiples capas con múltiples neuronas obtener una función no lineal que traduzca un conjunto de entrada x en un espacio de salidas y.

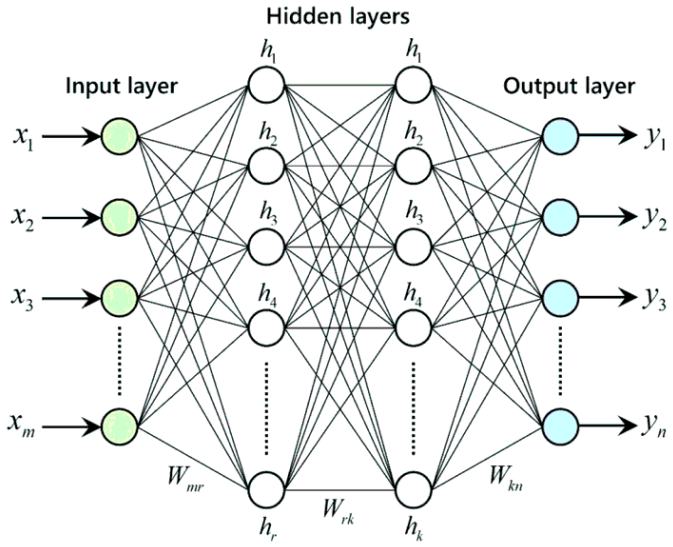


Figura 4.6 Red neuronal *fully-connected*, también llamada red densa.

Las redes neuronales artificiales *fully-connected* o *densas* son una de las formas más comunes de disponer de una red neuronal y son utilizadas en tareas de clasificación o identificación de elementos basados en las características de entrada x. En el caso del DRL tendremos que el espacio de entrada será el estado (codificado de alguna forma) y la salida será el valor de de la función Q para cada acción posible. Si tenemos A acciones distintas, veremos una capa de salida (*Outpur Layer*) de A neuronas que evalúan $Q(s,a_i)|i \in A$.

Una red neuronal tiene dos operaciones principales:

- **Propagación hacia adelante**: El *forwarding* (o propagación hacia adelante) es el cálculo de la salida y dada una entrada x. Esta operación se completa mediante el cálculo del valor de cada neurona i de cada capa j mediante su función de red que se pasa a la siguiente capa correspondiente:

$$f_{ij}(x) = \sigma\left[\sum_i W_{ij}x_i + b_i\right] \tag{4.8}$$

- **Propagación hacia atrás**: Está claro que la elección manual de los parámetros $\theta = [W,b]$ es inviable. Para hacer que estos parámetros modelen a través de la red la función deseada, se utiliza el método del gradiente descendiente [2]. La propagación hacia atrás consiste entonces en encontrar la sensibilidad (el gradiente si se prefiere) de una función de coste J que evalúa el error de las

[2] *https://en.wikipedia.org/wiki/Gradient_descent*

salidas y con de su valor real y_r, respecto de cada parámero de la red. De esta forma, en cada paso del entrenamiento de la red neuronal, tendremos que se varía cada parámetro como:

$$\theta_{k+1} = \theta_k - \alpha \frac{\partial J}{\partial \theta} 7 \tag{4.9}$$

donde α es el valor del *learning rate* que implica cómo de grande es el paso que aporta el gradiente.

El cálculo de las derivadas parciales de J se hace mediante la regla de la cadena para cada una de las capas, calculando la derivada primero de los últimos parámetros y propagando el valor de sensibilidad aguas arriba de la red (de ahí su nombre, *propagación hacia atrás*).

Nota: La función de activación $\sigma(x)$ se utiliza para introducir un componente de no-linealidad en la red neuronal, lo que otorga la capacidad de plasticidad no lineal a las redes. Existe una gran variedad de funciones de activación y la elección de qué funciones se escoge en cada caso no es única y viene marcada por la experiencia previa. Una función de activación (una que se utilizará más adelante) podría ser la función ReLU o la eLU:

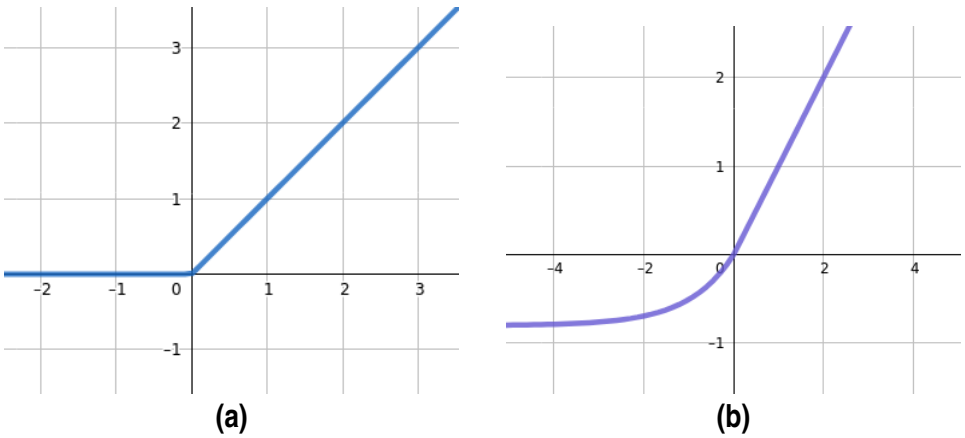


Figura 4.7 Función ReLU (a) y función eLU (b).

Al final, la elección de la función de coste, el número de neuronas de la red y el número de capas es una elección que responde a la experiencia, la intuición y el método de ensayo y error. El objetivo final será minimizar la función de coste J que explicita la diferencia entre los valores estimados y los reales.

4.4.2 Redes Neuronales Convolucionales

Cuando la entrada es una imagen representada por una matriz de $N \times M$ píxeles con n canales (normalmente 3 canales, uno por cada color del RGB), resulta conveniente trabajar con las denominadas Redes Neuronales Convolucionales.

Estas redes son una extensión natural de las redes neuronales aplicadas a una segunda dimensión espacial. Ahora, la operación esencial que realiza la red sobre la capa anterior es la *convolución*. La operación de la convolución 2D puede definirse matemáticamente para una imagen $\mathbf{x}$ de NxM píxeles como

$$\mathbf{x} * \mathbf{K} = \mathbf{y}[i,j] = \sum_{m=0}^{M} \left[\sum_{n=0}^{N} \mathbf{x}(m,n) \times \mathbf{K}(i-m, j-n) \right] \tag{4.10}$$

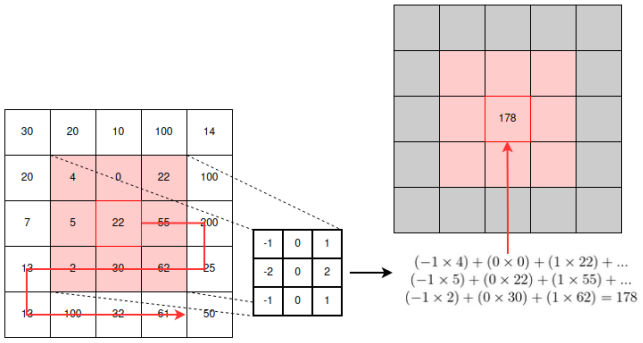


Figura 4.8 Cálculo de la convolución en un píxel dado.

donde $\mathbf{K}$ es el *kernel* o filtro de la operación. En estas redes, los valores de los píxeles de los filtros constituyen los pesos de la red y servirán para destacar los *features* y características típicas de las imágenes que queremos obtener. El resultado de la operación de convolución puede volverse a procesar por sucesivas capas convolucionales con distintos canales para seguir detectando *features* y elementos.

Figura 4.9 Efecto de convolucionar (a) con un kernel de Sobel.

Una Red Neuronal Convolucional típica posee unas capas iniciales convolucionales que extraen los elementos característicos de la imagen y simplifican la clasificación o estimación de la información de las imágenes. Seguidamente, la imagen/imágenes que serán mapas de características, se pasan a una red neuronal densa normal donde se reduce la dimensión (de 2D pasamos a 1D). A esto se le conoce como *Red Neuronal Convolucional Densa*.

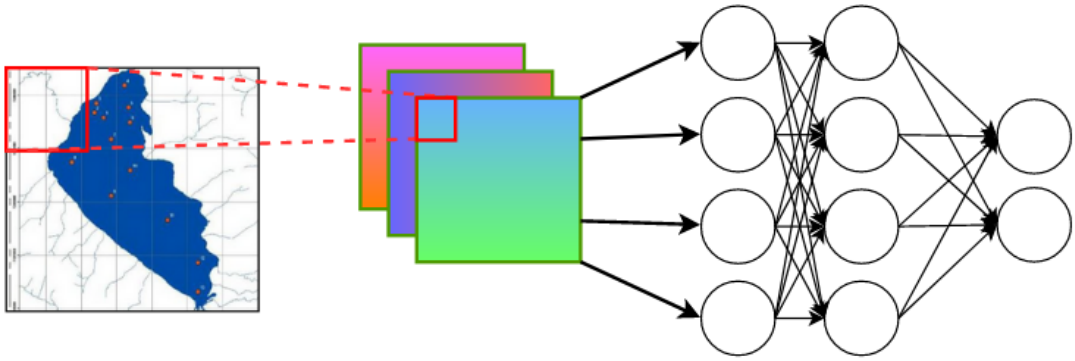


Figura 4.10 Esquema de una CNN densa.

4.5 Técnicas de Deep Reinforcement Learning

4.5.1 Deep Q-Learning

Se comenzará desarrollando el algoritmo más clásico de Deep Q-Learning.

Recordemos que en el algoritmo de Q-Learning se utiliza la función Q, denominada función de valor de estado-acción (*state-action value funcion*) que especifica cómo de buena es realizar una acción a en dado un estado s [[**?**]]. En el DQL se tiene que la estimación de la función $Q(s,a)$ recae sobre una red neuronal profunda (*Deep Neural Network*) en lugar de una tabla para cada s y cada a posibles. Así, en el proceso de estimación de Q se tiene que:

$$Q(s,a) \approx Q(s,a;\theta) \tag{4.11}$$

Esto se hace así debido a que es muy difícil recorrer todos los valores posibles de s para cada a en cada episodio cuando se tiene un espacio de estados y acciones muy grandes. La cantidad de valores distintos en una tabla Q sería de:

$$card(Q(s,a))\ \forall(s,a) = card(Estados) \times card(Acciones) \tag{4.12}$$

En el caso de considerar como estado solo la posición del vehículo en el mapa, se tiene $N \times M \times 8$ casillas posibles (siendo M y N las filas y columnas del mapa discretizado en celdas). Cuando se consideran estados más ricos en información, como una imagen RGB del escenario, la dimensión de Q explota pues existe un estado por cada posición y recorrido posible (se denomina en la literatura a este fenómeno *dimensionality curse*, maldición de la dimensionalidad). Esto es computacionalmente insostenible y es necesario introducir algunas funciones de estimación como son las redes neuronales que explicamos en el punto anterior.

Del método de *Time Difference* que utiliza el Q-Learning para actualizar los valores de la tabla Q se tiene:

$$Q(s,a)^{k+1} = Q(s,a)^k + \alpha(r + \gamma \max_{a'} Q(s',a';\theta)^k - Q(s,a;\theta)^k)$$

Así, queda definida la función de *target* o función objetivo como:

$$Q_{target} = r + \gamma \max_{a'} Q(s',a';\theta)$$

De esta forma, reordenando términos, se tiene que:

$$Q(s,a)^{k+1} = (1-\alpha) \times Q^k + \alpha \times Q^k_{target}$$

Se puede ver que la función objetivo es una función que intenta estimar el valor real de Q utilizando lo que se llamó *Time Difference*. Así, si imponemos que la función objetivo y la función Q son funciones parametrizadas de tipo red neuronal, se puede establecer que el error (según una función de pérdida arbitraria) de estimación de la función objetivo es:

$$Loss = J_{loss}(Q_{target}, Q)$$

Si J_{loss} tiene un gradiente estrictamente negativo y es decreciente con mínimo absoluto cuando $Q_{target} = Q$, mediante la actualización de los parámetros θ de las redes neuronales mediante el método del *gradient descent*, se puede llegar a minimizar la función de pérdida, esto es, llegar a tener una buena estimación de la función Q [12].

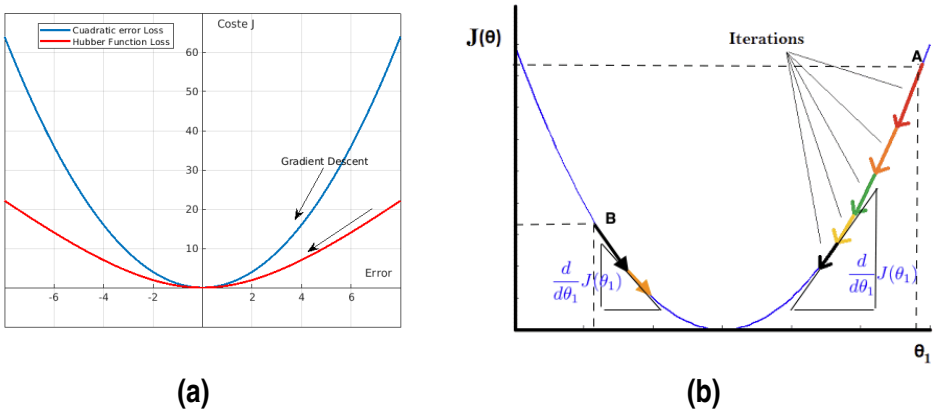


Figura 4.11 Distintas funciones de costo (a) y visualización del método del gradiente descendiente (b). Fuente: *https://machinelearningmedium.com/*.

Nota: La función de Loss de Huber es una función útil para el entrenamiento robusto de las redes neuronales debido a que es menos sensibles a los *outliers*. Se puede ver que es muy convexa en zonas de error pequeño pero con grandes errores tiene gradiente constante, lo que ayuda a evitar el problema del gradiente explosivo.

$$L_\delta(e) = \begin{cases} \frac{1}{2}e^2 & \text{for } |e| \leqslant \delta, \\ \delta(|e| - \frac{1}{2}\delta), & \text{otro caso.} \end{cases} \tag{4.13}$$

Esta función permitirá realizar un entrenamiento más estable que usando la famosa ecuación de Loss del error cuadrático medio, en el que a errores grandes, los gradientes son muy grandes también. La función de Huber se ha demostrado útil en regresión robusta y se ha utilizado con éxito en tareas de Reinforcement Learning por el autor de este proyecto.

Finalmente, una vez se tiene el concepto de Deep Q-Learning, se procede como en el algoritmos del Q-Learning con ciertas variaciones:

Algorithm 3: Algoritmo DQN con Buffer Replay

```
Inicialización aleatoria de los pesos θ de la función Q(s,a;θ);
Inicializamos epoch ← 0;
while epoch < epoch_MAX do
    step ← 0;
    Reseteamos el escenario y se tiene el estado s;
    while step < step_MAX do
        Tomamos una acción a mediante una política epsilon-greedy basada en
         Q(s,a;θ;
        Aplicamos a y adquirimos el nuevo estado del escenario s' y la
         recompensa r;
        Guardamos (s',a,r,s) en el replay memory;
        if replay memory tiene más de N experiencias then
            experiencias ← N experiencias (s',a,r,s) de replay memory;
            for (s,a,r,s') in experiencias do
                Q_target = r + γ máx_a' Q(s',a';θ);
                Actualizamos θ: dθ ← dθ + ∂J(Q,Qtarget)/∂θ;
            end
        end
        step ← step + 1;
    end
    epoch ← epoch + 1;
end
```

En este punto merece la pena explicar el concepto de **buffer replay**. Cuando se trabaja con el entrenamiento de redes neuronales, se tiene que evitar el efecto del *overfit*. Este fenómeno surge cuando se entrena utilizando una secuencia muy correlada de s y a y es lo que ocurriría si no se realiza una implementación del *buffer replay*.

El concepto de *buffer replay* es simple: tras cada acción, se guarda la 4-upla (s,a,r,s') en una memoria en la que caben hasta B tuplas. Tras esto, se saca de la memoria un conjunto de N experiencias y se entrena la red con estas. La selección de estas experiencias es aleatoria, por lo que la correlación entre experiencias será baja, se evita sesgar la red con experiencias semejantes y ayuda a que el agente pueda aprender de un rango mucho más amplio de vivencias.

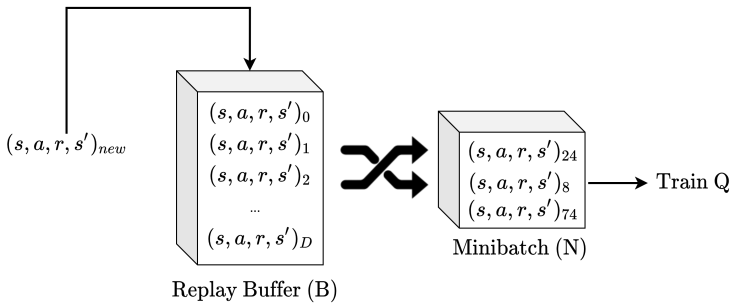


Figura 4.12 Esquema del concepto de *Buffer Experience*..

Ventajas y desventajas del método

Este algoritmo tiene sus ventajas y también sus limitaciones. Si bien es capaz de dar resultados buenos tales como los que fueron publicados en [5], se encuentra que estamos estimando y entrenando una función Q mediante otra función también estimada Q_{target}. Esta técnica, denominada *bootstrapping*, introduce en ocasiones problemas de convergencia en el aprendizaje y puede resultar en una sobreestimación de la misma función del *target* con unos resultados lejos de los obtenidos por otras técnicas.

4.5.2 Double Deep Q-Learning

Los estudios de Van Hasselt sobre la estimación de la función Q objetivo en [4] permitieron demostrar que hacer bootstrapping con la misma red del modelo Q hace que se sobre-estime el valor de los estados. Este efecto incurre en inestabilidades de entrenamiento y en recompensas sustancialmente más bajas causadas por la divergencia inicial entre la función del modelo y la función Q real.

En este mismo artículo se plantea una variante bautizada como Double Deep Q-Learning. En ella, tendremos dos redes neuronales distintas, una para estimar la función Q y otra para la función Q_{target}. Así, el loss de la función se calcula como:

$$Loss = J(\underbrace{r + \gamma \max_{a'} Q(s',a';\theta)}_{Q_{target}} - Q(s,a;\theta)) \tag{4.14}$$

donde tenemos que:

$$Q_{target}(s,a) \approx Q_{target}(s,a;\theta^{-1}) \tag{4.15}$$

De esta forma, el algoritmo Double-DQN consiste esencialmente en tomar la función de *target* e igualarla a la del modelo cada cierto tiempo mientras el resto del tiempo permanece inalterada. En [[4]] se propone una función del modelo desactualizada como una candidata adecuada para reducir el *bias*, esto es, inicializar la función de *target* y la del modelo con los mismos parámetros y cada cierto número de episodios copiar la función del modelo (que nunca para de entrenarse) en la del *target*.

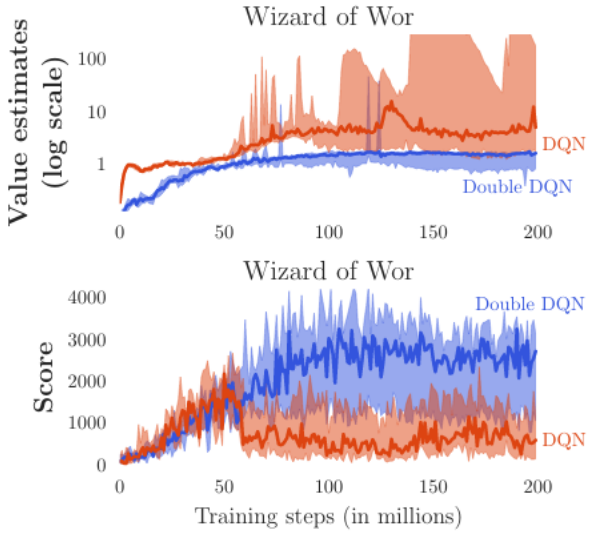


Figura 4.13 Resultado comparativo en [4] del entrenamiento con DDQN y DQN para el juego de Atari Wizard of Wor.

El algoritmo Double-DQN sería de la siguiente forma:

Algorithm 4: Algoritmo Double DQN

Inicialización aleatoria de los pesos θ de la función $Q(s,a;\theta)$;
Copiamos los pesos θ de la función $Q(s,a;\theta)$ en θ^- de $Q_target(s',a';\theta^{-1})$;
Inicializamos $epoch \leftarrow 0$;
while $epoch < epoch_{MAX}$ **do**
    step $\leftarrow 0$;
    Reseteamos el escenario y obse tiene el estado s;
    while $step < step_{MAX}$ **do**
        Tomamos una acción a mediante una política $epsilon$-greedy basada en $Q(s,a;\theta$;
        Aplicamos a y adquirimos el nuevo estado del escenario s' y la recompensa r;
        Guardamos (s',a,r,s) en el *replay memory*;
        if *replay memory tiene más de N experiencias* **then**
            $experiencias \leftarrow N$ experiencias (s',a,r,s) de *replay memory*;
            for (s,a,r,s') ***in*** $experiencias$ **do**
                $Q_{target} = r + \gamma \text{máx}_{a'} Q(s',a';\theta)$;
                Actualizamos θ: $d\theta \leftarrow d\theta + \frac{\partial J(Q,Q_target)}{\partial \theta}$;
            end
        end
        $step \leftarrow step + 1$;
    end
    Cada n episodios actualizamos θ^-: $\theta^- \leftarrow theta$;
    $epoch \leftarrow epoch + 1$;
end

4.6 Ley de recompensa

4.6.1 La ley de recompensa como modulación del objetivo

La ley de recompensa se conoce como fundamental de todo algoritmo de Reinforcement Learning, pues supone la forma que se tiene de modelar el comportamiento objetivo del agente: una ley de recompensa orientada a cubrir toda la superficie del lago Ypacaraí tendrá que bonificar el descubrimiento de casillas no visitadas (muy deseable) pero también el visitar casillas que hace cierto tiempo que no se han visitado (deseable). Además, esta regla penalizará con dureza cualquier acción ilegal que pueda llegar a querer cometer el agente; esto es, intentar cualquier acción que pudiera llevarlo a una casilla no navegable o visitar una casilla recién visitada de la que se tiene información reciente (redundancia inútil).

Así, se busca una ley de recompensa que mapea el estado y la acción realizada en un valor real positivo si la acción beneficia al objetivo y negativo si no contribuye al objetivo.

$$R_e : (s,a) \rightarrow \mathcal{R} \tag{4.16}$$

La ley de recompensa normalmente se escoge mediante un criterio heurístico, es decir, que se diseña con unas normas razonables basadas en el objetivo final pero de lógica generalmente difusa. A la hora de cuantizar qué se entiende por acción buena o mala se tiene en cuenta más que el valor absoluto, el valor relativo respecto de otras acciones. En el caso de la exploración del lago Ypacaraí, se puede proponer que una acción ilegal es 5 veces peor que descubrir una casilla nueva pero solo 3 veces peor que visitar una casilla recién visitada.

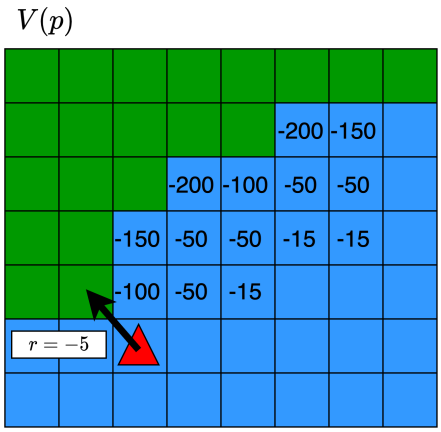


Figura 4.14 Representación de la función de valor para casillas en las fronteras..

Penalizar excesivamente ciertas acciones en esta clase de algoritmos hace que

se desarrolle cierta *fobia* por realizar esas acciones en ese estado, incluso llegando el caso de evitar a toda costa caer en estados cercanos. Esto ocurre por el mismo funcionamiento del algoritmo, que varía su función de valor $V(s)$ en función de la recompensa. Un estado muy penalizado puede llegar a hacer que situarnos en los bordes del mapa (en zonas donde muchas acciones son ilegales y llevan una penalización alta asociada; véase imagen 4.14) se evite constantemente, lo que realmente a nivel de *coverage* no es adecuado.

De esta forma, los valores de penalización se establecen a veces mediante ensayo y error y se corrigen en pruebas sucesivas.

4.6.2 Recompensa por matriz de interés ponderada

Es el primer objetivo de este trabajo el desarrollar un algoritmo capaz de cubrir el lago lo más uniformemente posible sin salirse de sus zonas navegables y revisitando las casillas que hace más tiempo que no se visitan (*patrolling problem*).

Surge la idea, como se expone en [8], de que existen zonas del entorno que en principio pudieran tener más interés que otras (por cuestiones biológicas u operativas cuando una zona del lago no es visitable o se conoce menos contaminada). De esta forma, se definirá la matriz de interés absoluta $\mathbf{R}_{abs}(i,j)$ como un campo escalar que para cada (i,j) posee el valor de lo importante que es esa zona a nivel estático, es decir, el interés de esa zona independientemente del tiempo.

Un valor de $\mathbf{R}_{abs}(i,j)$ constante indica que todas las zonas tienen el mismo interés y que el objetivo es recorrer todas las zonas por igual. La matriz $\mathbf{R}_{abs}(i,j)$ tiene sus valores entre 0 y 255 para poder ser representada como una matriz de niveles de gris o mapa de calor en el algoritmo.

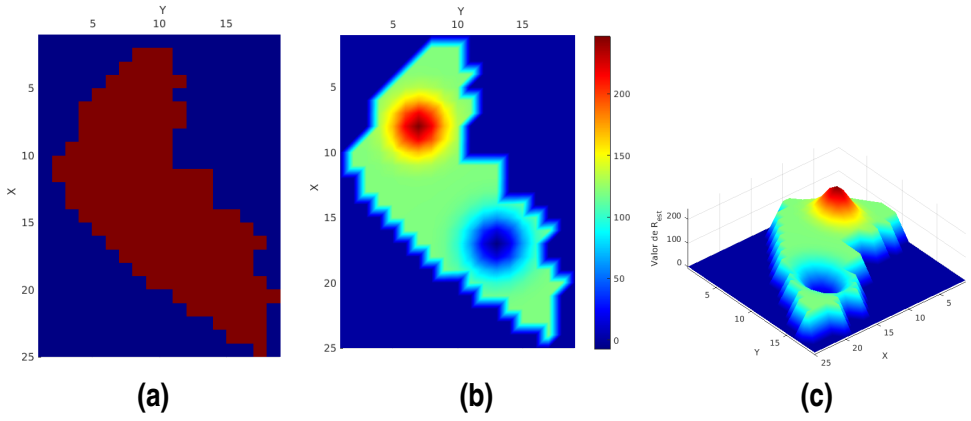


Figura 4.15 Distintas representaciones de la matriz estática con (a) constante y (b)(c) variable..

Ahora supongamos una matriz de ponderación $\mathbf{S} \in \mathcal{R}^{H \times W}$ con valores de 0 a 1.

Esta matriz ponderará cada posición de $\mathbf{R}_{abs}$ dependiendo de cuánto hace que ha sido visitada. De esta forma, $\mathbf{S}$ evoluciona según la siguiente fórmula:

$$\mathbf{S} = \begin{cases} \mathbf{S}(i,j)^{t+1} = min[\mathbf{S}(i,j)^t + \delta, 1] & \text{if } [x,y]_{agent} \neq [i,j] \\ \mathbf{S}(i,j)^{t+1} = 0 & \text{if } [x,y]_{agent} = [i,j] \end{cases} \tag{4.17}$$

La matriz $\mathbf{R}$ de ponderación relativa será el producto de Hadamard (producto punto, *element-wise product*) de ambas matrices:

$$\mathbf{R} = \mathbf{S} \circ \mathbf{R}_{abs} \tag{4.18}$$

Así, una celda recientemente visitada tendrá su interés ponderado con un valor pequeño por muy interesante que sea esa casilla a nivel absoluto. Las casillas van regenerando su importancia a lo largo del tiempo. De nuevo, la recompensa se define como el gradiente de interés entre una casilla y otra:

$$\begin{cases} \rho(s') = \nabla \mathbf{R} = \mathbf{R}(s') - \mathbf{R}(s) & \text{if } s' \neq ilegal \\ \rho(s') = -C_{illegal} & \text{if } s' ilegal \end{cases} \tag{4.19}$$

Se puede ver cómo esta ley de recompensa recompensa el movimiento hacia zonas menos exploradas pero siempre teniendo en cuenta que es mejor visitar un área ya explorada que salirse de las fronteras y dentro de esto, es mejor visitar casillas abandonadas hace más tiempo.

Como el algoritmo está diseñado para ser ejecutado durante un tiempo grande, tiene sentido incorporar un factor de desgaste en el mecanismo de recompensa. Siguiendo la ecuación de $\mathbf{S}$, se tiene que pasado un número $1/\delta$ de instantes, una casilla recupera su valor máximo en la matriz $\mathbf{R}$ y el algoritmo puede llegar a entender que es mejor visitar una casilla que se visita hace mucho tiempo que una completamente nueva. Este comportamiento indeseado se puede intentar enmendar imponiendo que cada visita a una casilla resta un interés estático fijo sobre la matriz $\mathbf{R}_{abs}$. Se ha impuesto en este algoritmo que cada visita sobre una casilla reste un 20 % de su valor original. De esta forma, aunque una casilla tenga un valor de interés absoluto de 255 (Valor máximo) al visitarla 5 veces ya no tiene ningún interés respecto a otras.

Se ha comprobado el comportamiento de esta ley de recompensa cuando el agente se mueve en modo manual (se ha desarrollado un programa para probar la ley de recompensa offline). Haciendo un símil con un mapa de calor, la recompensa se otorga cuando se pasa por zonas frías (zonas que tienen mucho interés, azules en la imagen 4.16). El interés máximo entonces estará cuando se maximiza el gradiente de calor al pasar de zonas muy cálidas (recién visitadas, poco importantes o ambas, cálidas) a zonas frías (poco visitadas, poco interesantes o ambas). Un movimiento de la casilla recién visitada (posición actual) a una casilla ya caliente, genera un gradiente de interés bajo, por ejemplo y viceversa.

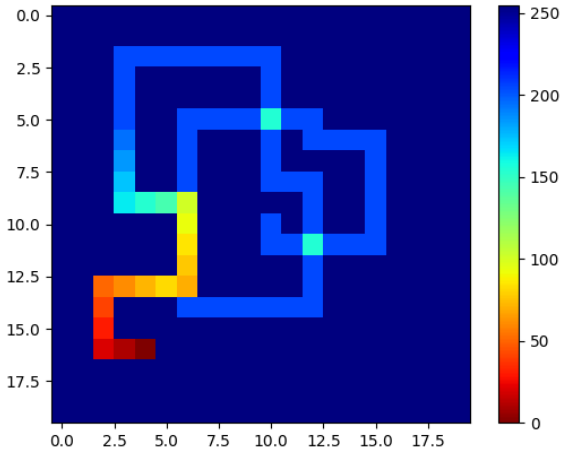


Figura 4.16 Representación de la matriz $\mathbf{R}$ como un mapa de calor..

Asimismo, en los algoritmos de RL, la recompensa suele estar acotada a unos niveles. Se debe definir las recompensas máximas y mínimas que se otorgan por cada acción de cada tipo. Por ejemplo, descubrir una casilla nueva genera un gradiente de interés de 255 (según se ha definido en 4.19), conlleva una recompensa neta de 1 por ejemplo, mientras que un gradiente de 0 conlleva una penalización de -0.5. Estos valores, al igual que el valor de $-C_{ilegal}$, van a condicionar muchísimo el alcance del algoritmo, por lo que valdrá la pena obtener un perfil de mejora variándolos en unos rangos. En el capítulo dedicado a los resultados se comprobará este efecto y se seleccionará el valor que mejor prestaciones dé.

En la tabla 4.1 se tiene un resumen de la ley de recompensa en la que ya se ajusta ρ para los valores máximos y mínimos al visitar una casilla nueva y una recién visitada respectivamente.

Tabla 4.1 Resumen de la ley de recompensa.

Ley de recompensa

$$\mathbf{S} = \begin{cases} \mathbf{S}(i,j)^{t+1} = min[\mathbf{S}(i,j)^t + \delta, 1] & \text{if } [x,y]_{agent} \neq [i,j] \\ \mathbf{S}(i,j)^{t+1} = 0 & \text{if } [x,y]_{agent} = [i,j] \end{cases}$$

$$\mathbf{R} = \mathbf{S} \circ \mathbf{R}_{abs}$$

Movimiento	Recompensa
A casilla ilegal	$-C_{ilegal}$
A casilla nunca visitada antes	r_{max}
A casilla recién visitada	r_{min}
De posición p a p' cualquiera	$\rho = \frac{r_{max} - r_{min}}{R_{max} - R_{min}} [\nabla \mathbf{R}(p,p') - R_{min}] + R_{max}$

Capítulo 5: Resultados

En este capítulo se expondrán la implementación de la solución del problema de exploración del lago Ypacaraí y los resultados obtenidos en los distintos entrenamientos. El contenido de este capítulo se organiza de la siguiente forma:

- Implementación del simulador con el que se llevará a cabo el entrenamiento.
- Explicación de las métricas y los hiperparámetros del entrenamiento.
- Resultados del entrenamiento con Deep Q-Learning para los problemas de cobertura total y parcial, con énfasis en la calibración de la ley de recompensa.
- Resultados del entrenamiento con Double Deep Q-Learning para los problemas de cobertura total y parcial.
- Análisis de sensibilidad del entrenamiento a los hiperparámetros.
- Comparación del desempeño de ambos algoritmos frente a otros algoritmos *naive*.

El código y resultados pueden encontrarse en el repositorio de *GitHub* *https://github.com/derpberk/YpacaraiReinforcementLearning*.

5.1 Simulador

5.1.1 Definición del escenario

El entorno es una parte fundamental del desarrollo de un algoritmo de aprendizaje, ya que es el medio en el que actúa el agente y el que determina lo bien o mal que realiza su misión. El escenario está compuesto por los siguientes elementos (necesarios para la implementación de un algoritmo de *Reinforcement Learning*):

1. **Tipo del mapa**: El mapa consistirá en un *grid map* en el que cada celda cuadrada hace referencia a una porción de espacio discretizado del entorno. Así, distinguiremos entre celda ocupable si está dentro del dominio navegable del lago o celda no ocupable si es un espacio de la tierra. El tamaño de estas celdas determinarán lo preciso que se describe el movimiento: un tamaño grande servirá para computar rápidamente el entorno a costa de perder la caracterización realista del espacio.

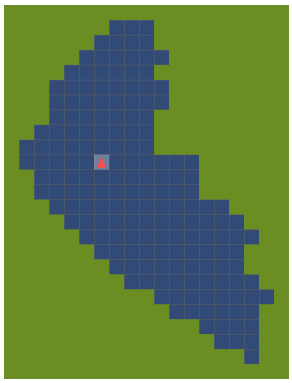


Figura 5.1 Gridmap del escenario..

Esta descripción discreta del entorno permitirá simplificar significativamente la carga computacional del entrenamiento: tenemos que tener en cuenta que cada paso del entrenamiento consiste en el avance del agente sobre una celda del mapa y en esta iteración habrán de recalcularse la posición, verificar si la acción ha sido legal, procesar el estado (sea cual fuere), actualizar las matrices de ponderación, etc. Un mapa excesivamente grande ralentizará sobremanera el entrenamiento, cosa que se ha querido evitar a toda costa.

2. **Interfaz de acción agente-entorno**: La interfaz entre el agente y el entorno comienza con las acciones que puede realizar el agente sobre el entorno para cambiar el estado y poder obtener una recompensa positiva. Las acciones son los movimientos posibles que realizará el barco dentro del lago, esto es, los movimientos a las 8 casillas adyacentes en cada instante (siguiendo los puntos cardinales: N,S,E,O y NE,NO,SE,SO, según la figura 5.2).

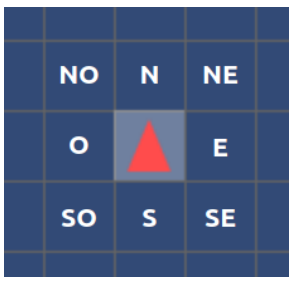


Figura 5.2 Convención de signos para los movimientos..

Esta descripción de cómo se puede mover el agente permite reflejar de forma más o menos realista cómo es el movimiento del ASV real: como se explica en el capítulo 1, el algoritmo de entrenamiento se centra en encontrar los *waypoints* que conformarán la trayectoria final. Realmente, el módulo de bajo nivel traduce entre movimientos entre celdas a velocidad, orientación y demás, así que tiene sentido incluir la posibilidad de realizar movimientos diagonales aunque su coste de movimiento pudiera ser mayor y se ignora la orientación por no importar en el problema realmente.

3. **Recompensa**: La recompensa es la respuesta del medio a la acción realizada por el agente. La ley de recompensa estará diseñada para bonificar comportamientos deseados y penalizar comportamientos ilegales o poco adecuados según se especifica en la fórmula 4.19. Para el diseño se ha tenido en cuenta una norma heurística bastante simple: con exploración eficiente entenderemos un algoritmo que permita recorrer el área máximo de superficie posible, atendiendo a un criterio de redundancia ponderado: visitar una zona del lago visitada recientemente es **peor** que visitar una zona que no hemos visitado antes (**mucho mejor**) o que visitamos hace tiempo (**no tan malo**).

4. **Interfaz de observación**: La interfaz de observación es la forma que tiene el agente de entender el estado del entorno. El estado (entendido dentro de un modelo de *Proceso de Decisión de Markov*, podrá (y tendrá) distintas formas dependiendo del algoritmo: posición en el mapa en forma de vector, la propia matriz del mapa, etc. De esta forma, se ha posibilitado obtener el estado de distintas formas para estudiar cómo afecta al proceso de entrenamiento el tener más información de su entorno: no será lo mismo poder observar solo la posición del agente que una imagen RGB que de una idea de todos los lugares visitados y con qué frecuencia han sido visitados. En este trabajo se han utilizado ambas representaciones para verificar cuál del ellas devuelve un resultado mejor.

Finalmente, se tiene un programa (*YpacarayMap.py*) que contiene un objeto tipo *Environment*. Este objeto posee distintos métodos para inicializar el escenario (*Environment.reset()*), para tomar una acción (*Environment.step(action)*), o para renderizar el estado del mapa (*Environment.render()*) etc. El escenario devolverá a cada acción un diccionario con la observación del estado (posición del agente, matrices de ponderación,...), la recompensa que ha generado esa acción y si el episodio ha terminado (*done*).

Este escenario posee todos los métodos fundamentales de cualquier escenario de la biblioteca de Python: *Gym* de OpenAI. Esto facilita la integración de este escenario con cualquier método de Reinforcement Learning desarrollado para Gym que un investigador o investigadora quisiera poner a prueba en el contexto del lago Ypacaraí.

5.1.2 Generación del gridmap

Para generar el gridmap se ha creado un programa en Python que, con ayuda de la biblioteca *OpenCV* permitirá tomar de un mapa la forma de una masa de agua y mediante una ajuste manual tomar solo la superficie navegable discretizada según una relación de resolución espacial configurable.

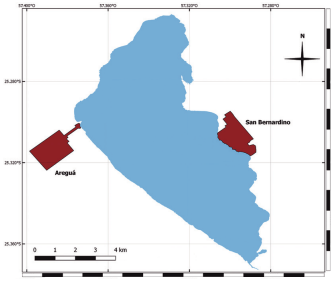


Figura 5.3 Mapa del lago Ypacaraí..

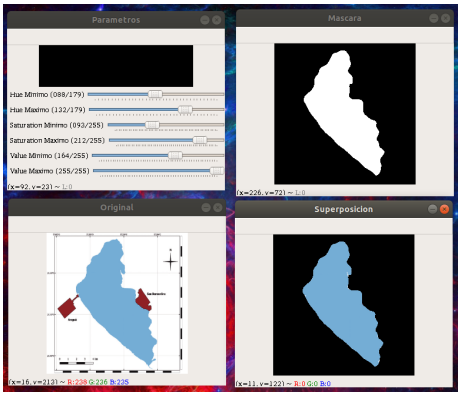


Figura 5.4 Interfaz del programa *map-adquisition.py*.

El programa realiza un filtrado por umbral de la imagen de forma que nos quedemos solo con el tono, color y brillo concreto (valor plano) . Posteriormente, el programa realiza un filtrado de mediana para poder rellenar huecos en la máscara que pudieran surgir por ruido (véase la figura 5.4). Al final, aplica un escalado siguiendo una fórmula lineal con unas constantes de equivalencia entre ancho de celda y valor en metros impuestos por el usuario.

$$width[px] = round\left(\frac{width\ map\ [Km]}{(px/Km)}\right) \tag{5.1}$$

$$height[px] = round\left(\frac{height\ map\ [Km]}{(px/Km)}\right) \tag{5.2}$$

Para comenzar, se ha tomado que 1 píxel de en el mapa escalado se corresponda a 250 metros para tener aproximadamente 20 píxeles de ancho de mapa.

El resultado se almacenará como un fichero CSV de ceros y unos. Para evitar tener que calibrar la imagen 5.3 siempre, el programa dispone de una opción *-d* (default) para poder usar los parámetros predeterminados. El resultado será como el de la figura 5.5.

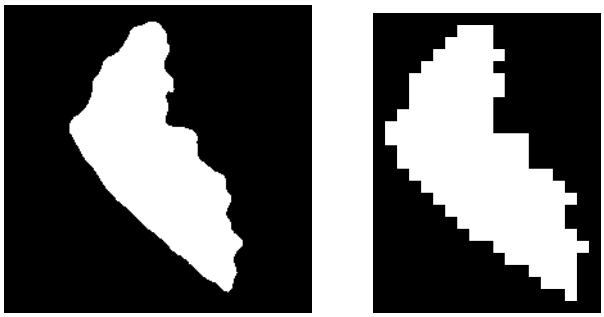


Figura 5.5 Máscara binaria (izquierda) y grid map resultante (derecha).

Esta herramienta resulta útil para discretizar no solo la imagen del lago Ypacaraí sino para obtener el mapa de muchas otras masas de agua u otro tipo de extensiones en mapas. En la figura 5.6 se puede ver un ejemplo con el lago Sanabria (Zamora, España).

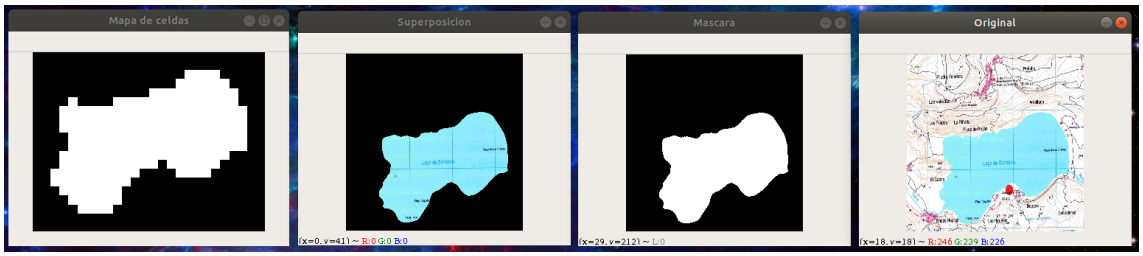


Figura 5.6 Proceso de calibración aplicado al lago de Sanabria (Zamora)..

Para probar el funcionamiento, se prueba el escenario con un *testbench* de 300 movimientos aleatorios y se renderiza la imagen RGB resultante tras terminar el episodio como se muestra en la figura 5.7.

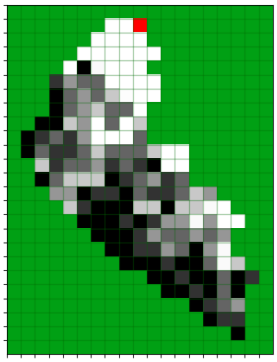


Figura 5.7 Render RGB tras 300 movimientos aleatorios con el estado completo observado.

5.2 Métricas de funcionamiento y parámetros de ajuste

Se ha realizado un amplio conjunto de experimentos con los algoritmos desarrollados para poder observar cómo afecta cada hiperparámetro de aprendizaje al resultado final. A continuación expondremos las distintas variantes de cada algoritmo implementado (DQN, DDQN, ...) y se analizarán las connotaciones de cada parámetro.

En el Reinforcement Learning (sobretodo cuando tenemos *Deep Reinforcement Learning*) los hiperparámetros van a influir significativamente en el desempeño final. Esto hace casi obligado, para poder obtener un resultado bueno, buscar de alguna forma aquellos valores que mejoren el entrenamiento. Se ha realizado en este trabajo una variación paramétrica de un valor cada respecto de un valor nominal para comprobar la mejoría (o desmejoría) del cambio. El valor nominal se ha obtenido con valores típicos utilizados en tareas semejantes en la literatura (valores utilizados en [8] y [19].

5.2.1 Parámetros de entrenamiento

Los parámetros de calibración considerados más importantes y los que se han variado son:

- *Learning Rate*: Valor de aprendizaje. Se corresponde con la cantidad del gradiente que se incorpora a los pesos de la red neuronal (α en la ecuación 4.9).
- *Batch size*: número de transiciones de $s \rightarrow s'$ que se extraen del *memory replay* para entrenar la red.
- ϵ-*decay*: cantidad en la que se reduce ϵ con cada episodio. Un valor alto implica que exploramos durante menos tiempo y explotamos durante más tiempo. Un valor pequeño indica lo contrario.

- *Número de neuronas*: cantidad de filtros convolucionales y neuronas de las estructuras de la red neuronal implementada para estimar las funciones Q del modelo y del *target*.

- *Parámetros de la recompensa*: valores de $C_{illegal}$, $max(\rho)$ y $min(\rho)$. Indican el valor de penalización por una acción ilegal, por seguir un gradiente de interés bajo y visitar una casilla inexplorada, respectivamente.

Los valores nominales de estos parámetros (que servirán de *baseline* para comparar cada cambio) serán:

Tabla 5.1 Hiperparámetros nominales.

Hiperparámetro	Componente
Replay Memory Size	10.000
Memory Batch	250
ϵ-decay	0.0007
Learning Rate	$1e^{-3}$
γ	0.99
Número de episodios	1.500
Movimientos por episodio	300

5.2.2 Métricas de funcionamiento

Figuras de mérito

Se definen 3 figuras de mérito que caracterizarán la bondad del resultado obtenido:

1. **Cobertura**: La cobertura definida como el número de casillas nuevas visitadas respecto del número de casillas visitables en el mapa. Una cobertura del 100 % consigue el objetivo del *Complete Coverage*.

$$\kappa = \frac{\sum_{\forall(i,j)visited}[\mathbf{R}_{abs}(i,j)]}{\sum \mathbf{R}_{abs}} \tag{5.3}$$

   Siendo κ la cobertura en tanto por uno. En el caso de cobertura homogénea (en el que cada posición de $\mathbf{R}_{abs}$ vale lo mismo, la fórmula 5.3 se reduce a la ecuación

$$\kappa = \frac{m}{N} \tag{5.4}$$

   con m el número de casillas visitadas y N el número de casillas totales.

2. **Tiempo medio efectivo de visita**: Es el tiempo medio que tarda cada casilla

en recibir una visita y ponderado en relación al área cubierta. Esto se hace así para poder introducir la importancia de la cobertura en la figura de mérito puesto que cuando se cubre pocas zonas, el tiempo de visita de las mismas es menor porque se alterna la posición del agente entre unas pocas casillas.

Se define $\tau_i^{(j,k)}$ como el tiempo entre visitas en la casilla *(i,j)* en el instante k:

$$\tau_k^{(i,j)} = t_k^{(i,j)} - t_{k-1}^{(i,j)} \tag{5.5}$$

El tiempo medio de visita de cada casilla será la media de tiempos de espera en esa posición:

$$\mu_\tau^{(i,j)} = \frac{1}{n^{(i,j)}} \sum_{k=0}^{n} \tau_k^{(i,j)} \tag{5.6}$$

con $n^{(i,j)}$ como el número de visitas de esa casilla.

Finalmente, el tiempo medio efectivo será la media de tiempos medios de espera de cada casilla visitada, ponderado por su nivel de importancia (si la importancia es igual para todas las zonas los tiempos de espera importarán igual) y evaluado en función de su área cubierta:

$$T_{mean} = \frac{1}{m} \sum_{\forall(i,j)} \left[\mu_\tau^{(i,j)} \frac{R_{abs}(i,j)}{max R_{abs}} \right] \tag{5.7}$$

quedando definido T_{mean}^{eff}:

$$T_{mean}^{eff} = \frac{T_{mean}}{\kappa} \tag{5.8}$$

3. **Homogeneidad efectiva de visita**: Define la homogeneidad del tiempo de visitas de cada casilla. Visitar una zona mucho y otra poca implica que la desviación típica de $T_{\text{entre visitas}}$ aumente mucho lo que no es conveniente para el objetivo del algoritmo. Del mismo modo que el tiempo medio efectivo, se define:

$$H_{mean} = \sqrt{\frac{1}{m} \sum_{\forall(i,j)} \left[\mu_\tau^{(i,j)} - T_{mean} \frac{R_{abs}(i,j)}{max R_{abs}} \right]^2} \tag{5.9}$$

y finalmente:

$$H_{mean}^{eff} = \frac{H_{mean}}{\kappa} \tag{5.10}$$

5.3 Resultados con Deep Q-Learning con importancia homogénea

El objetivo es entrenar al agente para cubrir el máximo área posible y que la cobertura de las celdas, en media de tiempo entre visita y visita, sea lo menor posible. Existen muchas formas de entrenar a la red igual que existen muchos valores de hiperparámetros posibles que intervendrán en el desempeño final, así como formas de entender el estado del escenario. Con la cobertura homogénea tendremos que la matriz de importancia absoluta es constante (como se ve en la figura 5.8). Esto quiere decir que un buen resultado estará en una cobertura máxima con valores mínimos de las figuras de mérito T_{eff} y H_{eff}.

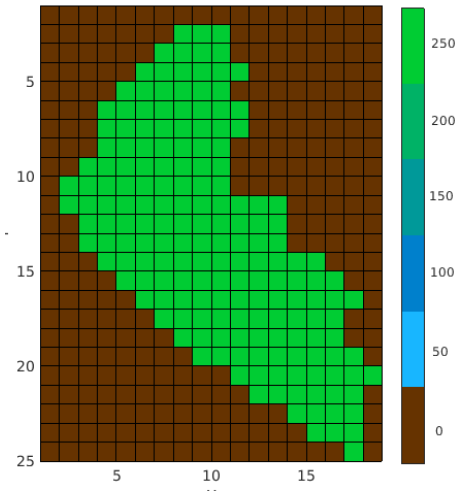


Figura 5.8 Mapa de importancia absoluta en el problema de cobertura homogénea.

5.3.1 Resultados para distintas formas de estado

El estado, como se menciona en el capítulo 2, supone cómo observa el agente el entorno. En un principio, se definen 3 estados posibles para este problema, cada uno con un nivel mayor de información que el anterior (basado en distintas técnicas utilizadas en la bibliografía estudiada en el capítulo 2). Se demostrará que cuanta más información es capaz de obtener el estado, mejor desempeño final:

- s es la posición del agente como el vector (i,j) dentro del mapa.
- s es una imagen RGB del mapa en el que se distingue celda no visitada (negro), celda visitada (blanco), posición del agente (rojo) y terreno no transitable

(verde).

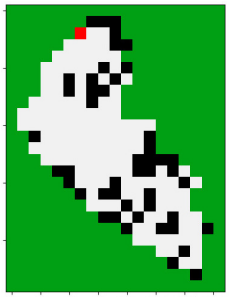


Figura 5.9 Imagen RGB del estado considerando solo las visitas.

- s es una imagen RGB igual que en el punto anterior pero cada celda visitada tiene como valor el de su correspondiente en la matriz de ponderación relativa $\mathbf{R}$.

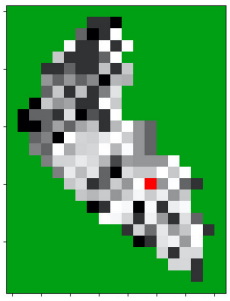


Figura 5.10 Imagen RGB del estado considerando la matriz $\mathbf{R}$.

Entrenamiento con s como posición

La red neuronal utilizada para estimar tanto Q como Q_{target} será:

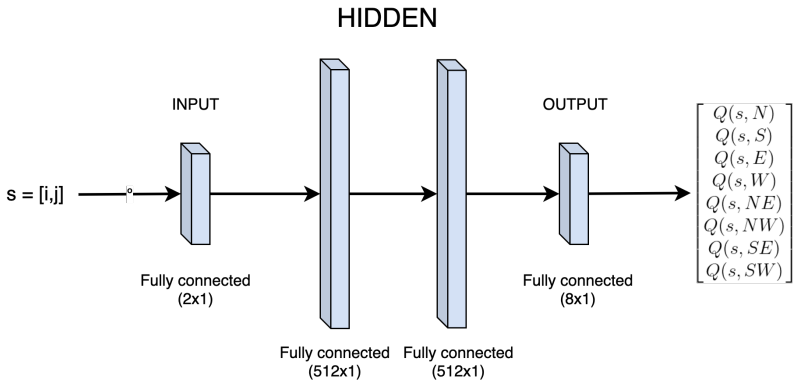


Figura 5.11 Red neuronal con posición del agente como entrada..

Se puede comprobar en la figura 5.12 que el entrenamiento no es satisfactorio puesto que se consiguen en las épocas finales un rendimiento bajo. Las recompensas máximas se sitúan por debajo de 100 y se visitan como máximo 60 casillas nuevas. El resultado de ejecutar un ciclo de 300 movimientos para probar cómo se explora el lago a lo largo del tiempo devuelve resultados mediocres pero esperables: la cobertura media para 10 partidas no llega al 37 % del área total y la redundancia es muy alta (se visitan continuamente las mismas casillas). Este resultado devuelve también la aparición de bucles triviales en el movimiento: el efecto de la falta de información es generar bucles que maximicen la recompensa y estos bucles suelen tender a cerrarse en cuanto el tamaño del bucle es lo suficientemente grande como para que el interés relativo de **r** se haya regenerado.

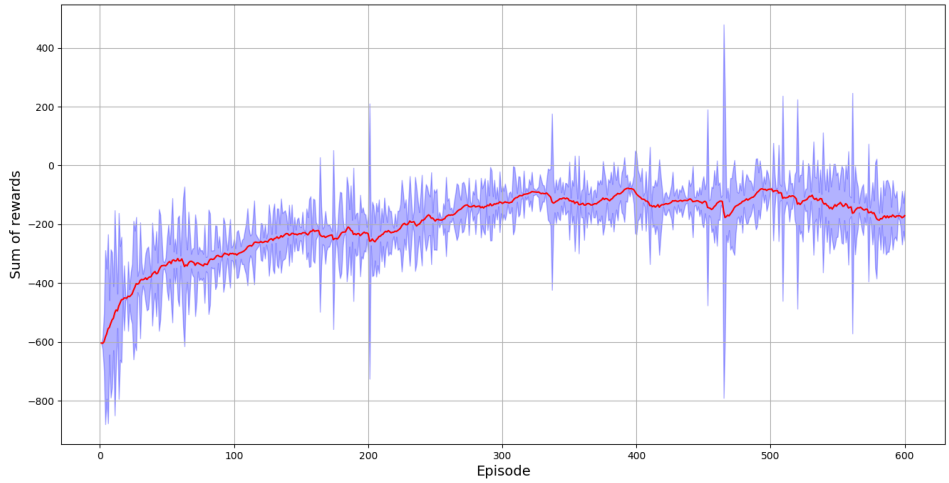


Figura 5.12 Resultado del entrenamiento para el caso 1.

Mapeando en cada posición la dirección preferida se puede ver que el resultado es un mapa vectorial que genera tendencia a realizar bucles cerrados y con poca tendencia a la exploración total.

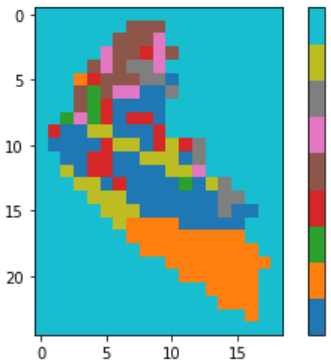


Figura 5.13 Mapa de direcciones resultantes del entrenamiento.

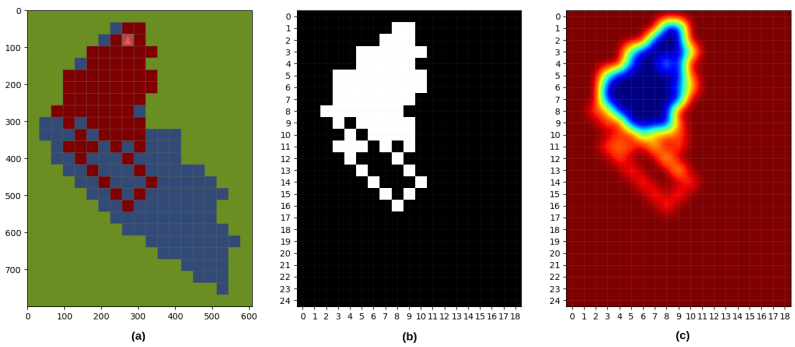


Figura 5.14 Resultado de una prueba de 250 movimientos con el entrenamiento 1. En (b) se tiene las casillas visitadas en blanco y en (c) se tiene la matriz $\mathbf{R}$ en forma de mapa de calor interpolado..

Entrenamiento con s como RGB de celdas visitadas

En este entrenamiento se transforma el programa para que admita como estado una imagen RGB (3 canales) en la que se tiene 4 tipos de casillas: verde para zonas no transitables, rojo para la posición del agente, blanco cuando se visita esa casilla o negro cuando no se han visitado:

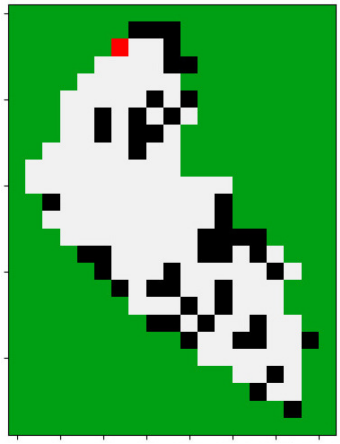


Figura 5.15 Imagen RGB que se le pasa al algoritmo Deep Q-Learning en su 2ª forma del estado.

Para transformar el programa y que la red neuronal admita una matriz RGB de 3 canales se ha la red convolucional de la imagen 5.16, que será la nominal en la comparación y está sacada de [8].

Se puede ver en la imagen 5.17 que el resultado del entrenamiento es mucho mejor, puesto que el crecimiento se mantiene constante a lo largo de casi toda la duración para establecerse al final. El valor de recompensa máximo llega a ser de 114 (un 190 % más que con el anterior método) y al realizar 50 partidas de 300 movimientos,

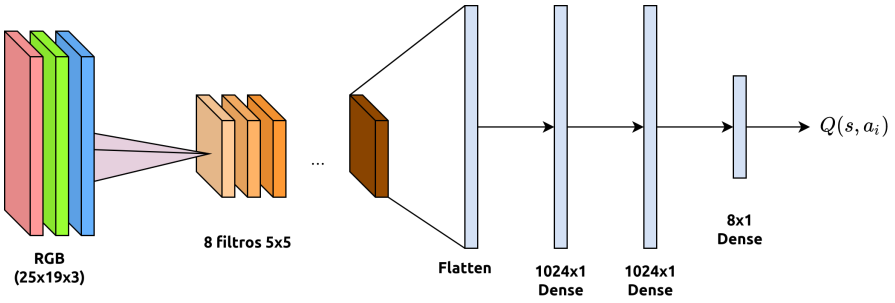


Figura 5.16 Red neuronal convolucional con imagen RGB del escenario como entrada..

se llega a cubrir (haciendo la media) un 61 % del área total.

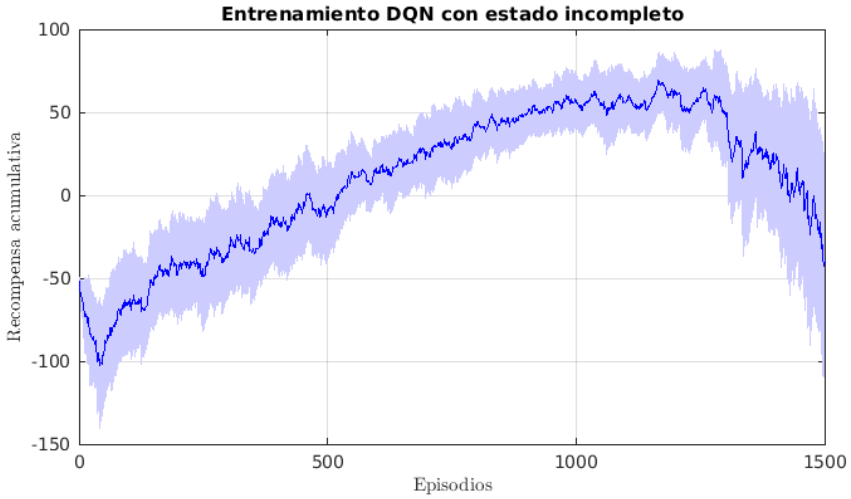


Figura 5.17 Resultado del entrenamiento para el caso 2..

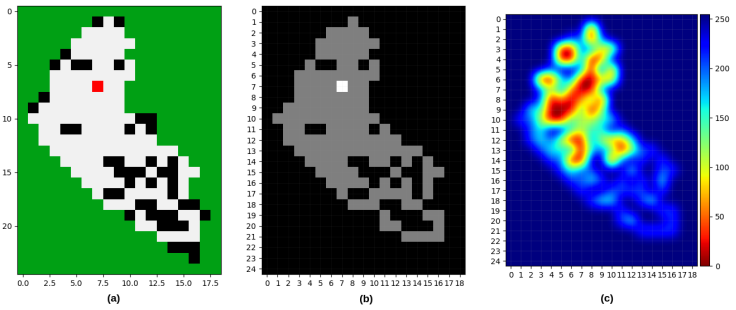


Figura 5.18 Resultado de una prueba de 300 movimientos tras el entrenamiento 2. En (c) las zonas rojas se corresponden a zonas más y recientemente visitadas y las más azules las menos visitadas o que hace tiempo que no se visitan.

Se ve en la figura 5.17 que el resultado del entrenamiento es **significativamente mejor** que en el caso anterior. La cobertura mejora sensiblemente aunque aún se sigue observando que a pesar de que las recompensa aumenta, la redundancia inútil

sigue siendo alta y se visitan las mismas zonas demasiadas veces mientras otras quedan más olvidadas.

En el mapa de calor de la figura 5.18 se tiene que la cobertura ha mejorado pero que el mapa de calor no es muy uniforme y sigue habiendo mucha información no cubierta mientras que en algunos puntos la información se ha agotado por haber pasado numerosas veces por esa zona. Es necesario aportar cierta información al agente del recorrido previo para que tenga en cuenta el interés a la hora de seleccionar el siguiente movimiento.

Entrenamiento con s como RGB completo

Para el último experimento se provee al algoritmo de toda la información posible disponible en el escenario. El estado s que recibirá el agente será una imagen RGB como la anterior pero con cada casilla visitada con un color en escala de grises que indique su valor de interés relativo, es decir, $\mathbf{R}(i,j)$. Así, se tiene que el estado tendrá la siguiente forma:

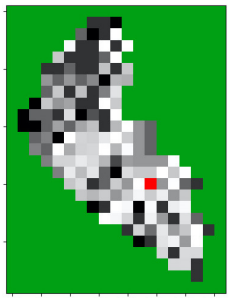


Figura 5.19 Imagen RGB del estado completo..

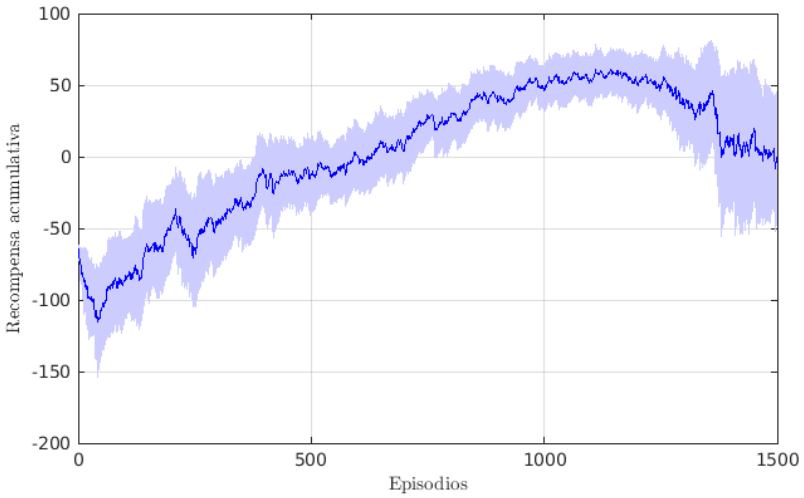


Figura 5.20 Resultado del entrenamiento 3..

El resultado del entrenamiento puede verse en la figura 5.20. Se observa cómo la tendencia es creciente a lo largo de todo el entrenamiento hasta decaer al final. Se ve claramente que la recompensa es mayor respecto del caso anterior y que el crecimiento es casi lineal. Nutrir de esta nueva información al algoritmo es evidente que mejora las prestaciones del aprendizaje.

Si se analiza el resultado de un ciclo de 300 movimientos posibles con el modelo de entrenamiento resultante (la red que devuelve la mayor recompensa, no necesariamente la última pero cercano a los últimos episodios), se tiene que la cobertura del lago Ypacaraí es casi total:

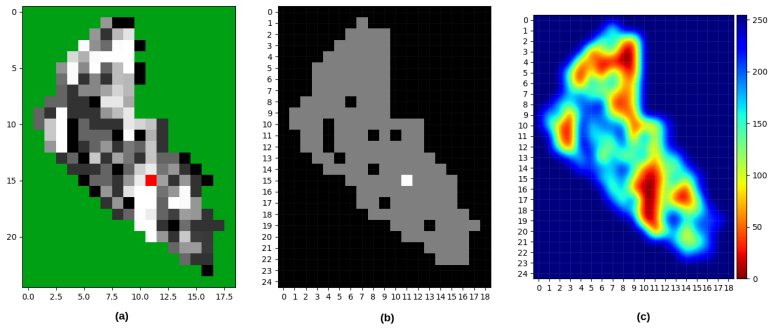


Figura 5.21 Resultado de una prueba de 300 movimientos tras el entrenamiento 3. En (c) las zonas rojas se corresponden a zonas más visitadas y las más azules las menos visitadas..

Se puede ver en la 5.21(b) (resultado de la mejor partida) que se visita un 87.7 % del área total. Asimismo, se tiene que la visita a las casillas de lago es significativamente más uniforme que en el caso anterior, cosa lógica al tener en cuenta que el algoritmo posee ahora un nivel de conocimiento del entorno mucho más amplio y es capaz de estimar mejor el valor óptimo de Q, es decir, una política $\pi(s)$ más cercana a la óptima.

Comparativa entre métodos de observación del estado

Se ha podido observar cómo a medida que el algoritmo recibe más información útil en el estado mejor es el desempeño final de su tarea. En la gráfica 5.57 se puede comprobar la comparativa de recompensas.

Esta imagen hay que analizarla con cautela. El resultado del aprendizaje es parecido en ambos casos, pero cuando se tiene más información de los caminos recorridos previamente cuando se usa el estado completo del mapa se puede mejorar también la uniformidad de la cobertura.

Una de las principales razones de que ocurra esto es debido a que el algoritmo del Deep Q-Learning tal y como se plantea tiene problemas al lidiar con las dependencias temporales cuando el estado proporcionado no es *completamente* identificativo del

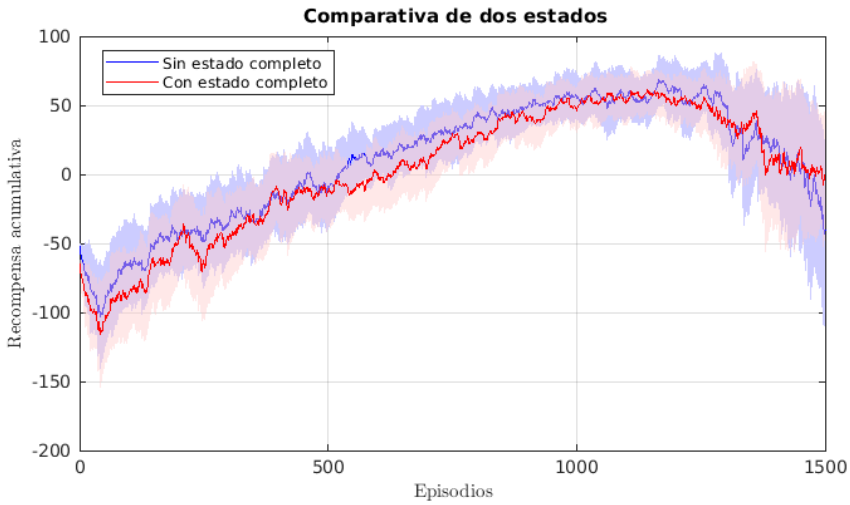


Figura 5.22 Comparativa de aprendizaje del método 2 y el 3..

estado real del entorno. Con la posición solo, de una evaluación de la red neuronal a otra dada una posición, el algoritmo encuentra imposible *recordar* las acciones pasadas. Esto viene a incidir en el hecho de que cuanto más información aporte el estado, mejor. A fin de cuentas, la resolución de problemas mediante Q-Learning está bajo la hipótesis de tener un MPD completamente observable, cosa que no ocurre con las dos primeras formas de estado que aquí se han probado-

Así, se puede ver en la figura 5.57 que el entrenamiento es más efectivo en el caso del método 3 puesto que alcanza el establecimiento más tarde que en el caso 2, por lo que será candidato para una las siguientes pruebas (hay que tener en cuenta que las pruebas pueden llegar a tener una duración de horas y en las primeras fases de experimentación esta criba inicial puede servirá para no realizar pruebas largas y poco provechosas).

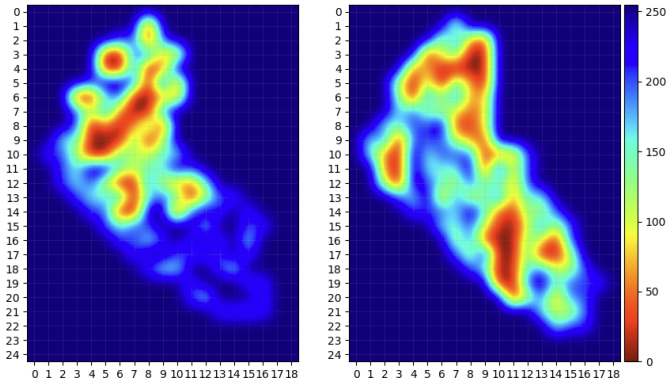


Figura 5.23 Comparativa de la cobertura del método 2 y el 3 con un episodio de 300 pasos..

En la 5.23 se ve claramente la diferencia entre el método 2 y el 3: no solo mejora

el objetivo del *coverage* sino el de la uniformidad de la cobertura ya que se tiene que es igual de importante visitar toda la extensión de lago como que se cubra el lago de forma uniforme y no muchas veces unos sitios y pocas veces otros. Queda claro que el método 3 es el candidato a mejorarse y a entrenarse durante mucho más tiempo.

Queda claro que aportar el estado completo al algoritmo es la mejor forma de propiciar el aprendizaje. En las figuras 5.24, 5.25 y 5.26 se ve la mejoría en las figuras de mérito tras realizar 20 test para cada algoritmo.

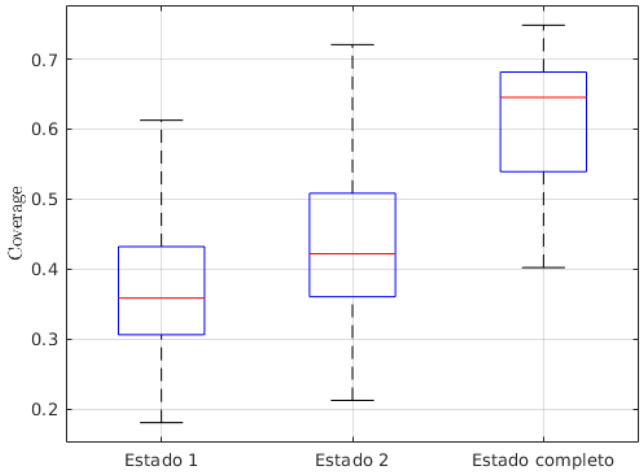


Figura 5.24 Diagrama de cajas para la cobertura con los tres tipos de estados.

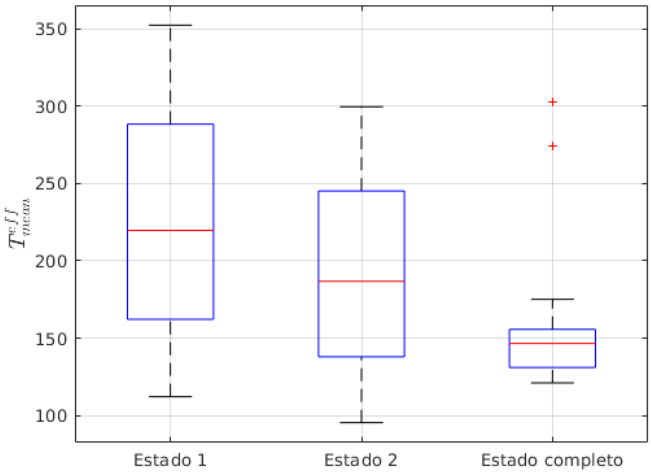


Figura 5.25 Diagrama de cajas para el periodo de visita con los tres tipos de estados.

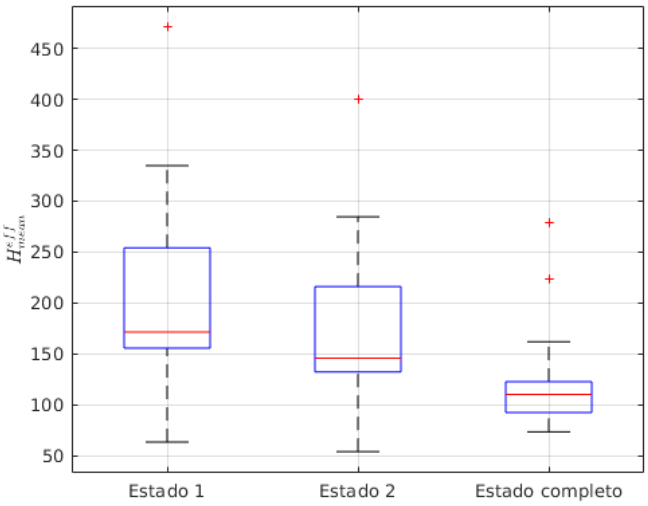


Figura 5.26 Diagrama de cajas para la homogeneidad con los tres tipos de estados.

5.3.2 Estudio del efecto de la ley de recompensa

Es conocido que la ley de recompensa viene a influir en el resultado del entrenamiento del agente, tanto a nivel de estabilidad, alcance y objetivos. Es por ello que para encontrar resultados destacables se suele llevar a cabo un *tunning* más o menos intensivo de los parámetros y en especial de los concernientes a la ley de recompensa.

En este trabajo, que lejos de pretender encontrar un algoritmo inmejorable, intenta demostrar que es posible resolver el problema inicial desde la perspectiva del RL, se han realizado una serie de experimentos con distintos parámetros de la recompensa (para casilla recién visitada, casilla nueva y movimiento ilegal). Una vez estudiado el resultado de entrenar con las distintas tríadas de valores, se escogerá para mejorar el que de mejores índices en las métricas.

Experimentos para distintas recompensas

Se prueba el algoritmo DQN con distintos valores de recompensa. Los valores de recompensa van variando en el sentido de penalizar más o menos las acciones ilegales (para tener menos movimientos ilegales), penalizar más la redundancia inútil (para cubrir casillas más *frescas*) o bonificar más visitar nuevas casillas (para aprender a cubrir el mapa mejor).

Las distintas recompensas que se han probado son:

Nº	Ilegal	Reciente	Nueva
[1]	-5	-0.5	+1
[2]	-10	-0.5	+1
[3]	-5	-1.5	+1
[4]	-5	-0.5	+1.5
[5]	-10	-0.5	+5
[6]	-0.5	-0.5	+1

Se ha ejecutado el algoritmo de entrenamiento de DQN con el estado RGB completo y se han realizado 20 pruebas. Al final, se procesan las 3 figuras de mérito para representarlo en la imagen siguiente:

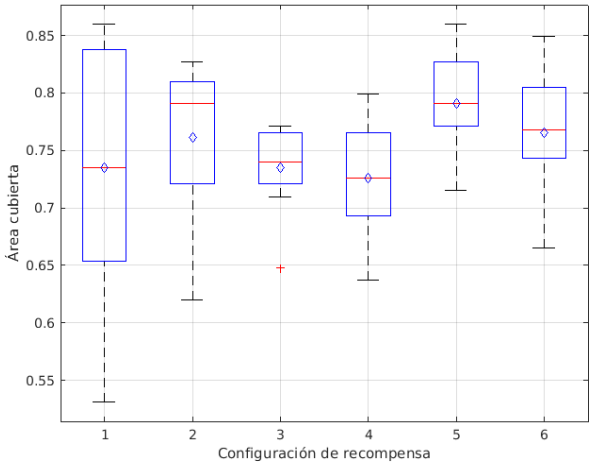


Figura 5.27 Diagrama de cajas para la cobertura..

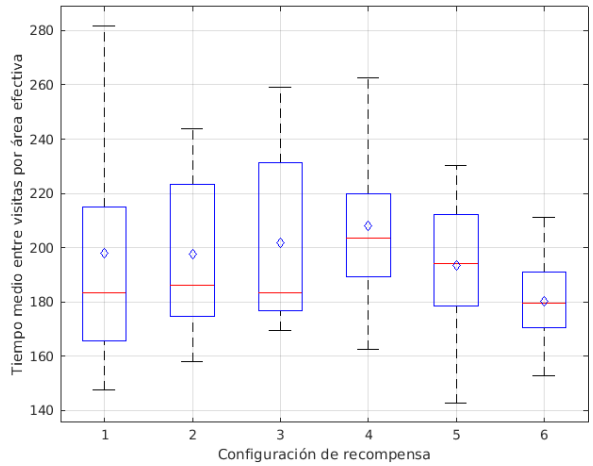


Figura 5.28 Diagrama de cajas para T^{eff}_{mean}..

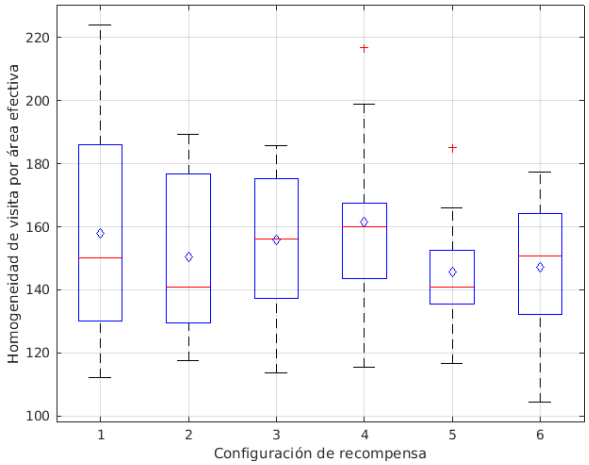


Figura 5.29 Diagrama de cajas para H^{eff}_{mean}..

Se puede observar de los diagramas de cajas que los mejores resultados están en los parámetros 5 y 6. La cobertura y la homogeneidad es mejor en el caso 5, mientras que existe más dispersión en el tiempo medio de visita a favor del caso 6. Se escogerá el caso 5 para continuar mejorando el algoritmo porque además de dar buenos resultados, al penalizar más las acciones ilegales proporciona un comportamiento menos errático.

5.3.3 Ventajas y desventajas del Deep Q-Learning

Se ha podido observar que con el algoritmo de Deep Q-Learning se obtienen unos resultados positivos en cuanto al aprendizaje del agente para cubrir regularmente todo el espacio visitable. El agente, episodio tras episodio, aprende a tomar acciones que maximizan la recompensa. No obstante, se puede observar un fenómeno frecuente en los procesos de Deep Learning: el *catastrophic forgetting*.

Este fenómeno puede verse, por ejemplo, en la imagen 5.20: a partir de cierto punto, se olvida lo aprendido, la red comienza a desentrenarse y la media de recompensa por episodio cae rápidamente. Este fenómeno impide que la red pueda entrenarse durante demasiado tiempo con resultados cada vez mejores, puesto que tarde o temprano desaprenderá lo que tanto le costó aprender.

Este fenómeno suele ocurrir en el entrenamiento de las redes neuronales (y en el Machine Learning en general) debido a la plasticidad de las redes de estructura fija como son las redes utilizadas. El mecanismo de *backpropagation* utilizado para entrenar la red puede generar cambios significativos en los pesos de las neuronas ante cualquier cambio en las entradas, llevando el aprendizaje a la inestabilidad [20]. Esta característica, sumada con la aproximación de la función Q mediante *bootstrapping* (utilización de la propia función Q estimada para estimar Q_{target}) en el algoritmo DQN y que estamos tratando con algoritmo *off-policy* (porque utilizamos una política

de comportamiento para seleccionar la acción con $Q(s,a)$ y otra política *greedy* para realizar el *bootstrapping* con $max_a Q(s',a)$). Esto conforma una *tríada mortal* [7] que introducirá divergencias e inestabilidades en el entrenamiento.

5.4 Resultados con Deep Q-Learning con importancia no homogénea

Partiendo de los resultados del estudio de la sección 5.3.2 y habiendo escogido la ley de recompensa más prometedora de entre todas las probadas y proporcionando el estado RGB completo, se procede a comprobar los resultados del aprendizaje para el caso de cobertura de importancia homogénea con el algoritmo de Double Deep Q-Learning.

Del trabajo de [14] se ha creado un mapa de interés estático $\mathbf{R}_{est}$ que se incorporará a la ley de recompensa con distintos niveles de interés dependiendo de la zona geográfica y de contaminación (véase figura 5.30).

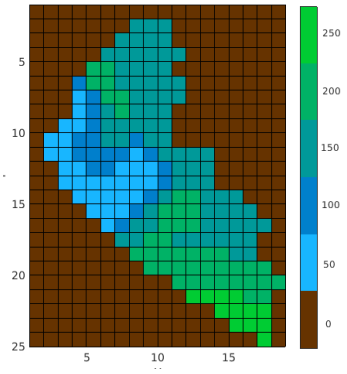


Figura 5.30 Matriz $\mathbf{R}_{est}$ señalando las distintas zonas de interés por contaminación..

En el entrenamiento de esta red puede observarse mucho más intensamente el fenómeno de la inestabilidad del entrenamiento y el *catastrophic forgetting*. Este fenómeno hace que el resultado final para el caso no homogéneo sea muy pobre, e invita a probar con el resto de algoritmos.

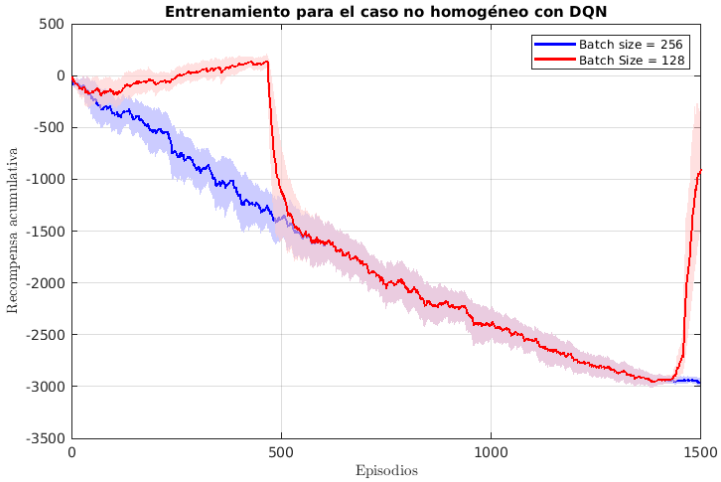


Figura 5.31 Resultado del entrenamiento para DQN en el caso no homogéneo.

En la figura 5.31 se puede comprobar también cómo afecta el tamaño del batch al aprendizaje. Esto demuestra la dependencia del desempeño final respecto de los parámetros de entrenamiento. La estabilidad del entrenamiento será un compromiso entre lo bueno que sea el algoritmo implementado (DQN, DDQN, ...) y los hiperparámetros escogidos junto con la arquitectura de la red neuronal. Como ya se dijo antes, se demostrará que con el algoritmo de Double DQN, este fenómeno queda mitigado para ambos casos de entrenamiento.

5.5 Resultado con Double Deep Q-Learning con importancia homogénea

Se ejecuta bajo las mismas condiciones nominales y usando la ley de recompensa que mejor resultados da de la sección 5.3.2 el algoritmo Double DQN para el caso homogéneo. Se comprueba en la imagen 5.32 que se contrasta la hipótesis enunciada: la duplicación de la red para la función objetivo mejora la estabilidad del entrenamiento y proporciona mejores resultados en un caso general.

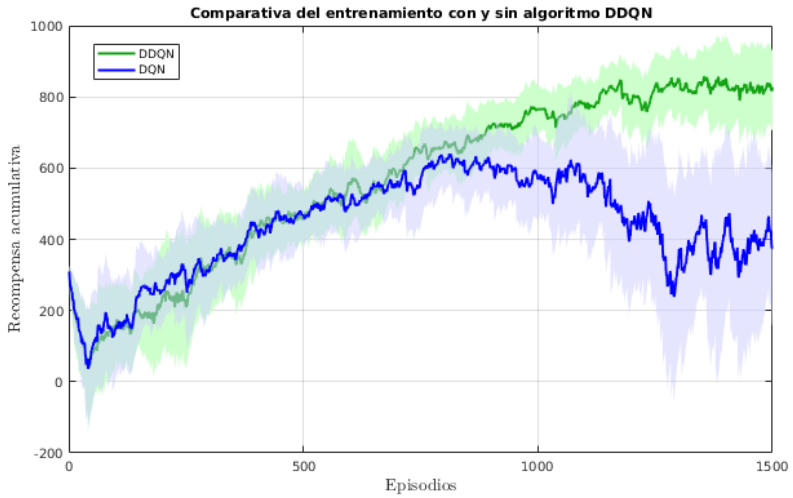


Figura 5.32 Comparativa del aprendizaje entre el algoritmo DQN y DDQN en el caso homogéneo.

La recompensa media en el caso nominal se sitúa en el entorno de 800 en las etapas finales frente a la puntuación de 600 en el caso del algoritmo de DQN simple. Este algoritmo (Double DQN) será candidato a un entrenamiento de más larga duración teniendo en cuenta esta mejoría, aún cuando el ajuste de hiperparámetros pudiera devolver recompensas mayores.

Se puede ver en la figura 5.33 la comparación del comportamiento del agente con DQN y Double-DQN para un episodio de 300 movimientos.

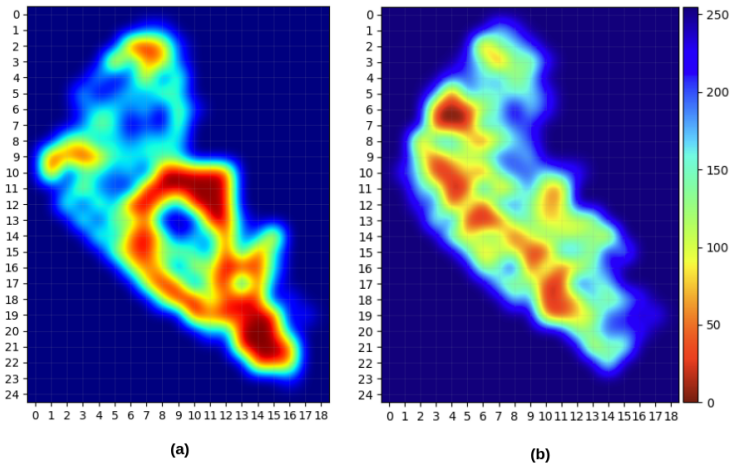


Figura 5.33 Comparativa en un test de 300 movimientos entre el agloritmo DQN y DDQN..

La mejoría es visible y se comprueba que el agente cubre más área en el segundo caso y que distribuye mejor la atención a lo largo de todo el mapa de importancia equitativa. Asimismo, tenemos que se visitan más eficientemente todas las zonas de misma importante. Este será un principio importante para poder entrenar la red en entornos variables. A continuación se exponen las figuras de mérito resultado de cada entrenamiento, comparadas con el resultado de una exploración aleatoria:

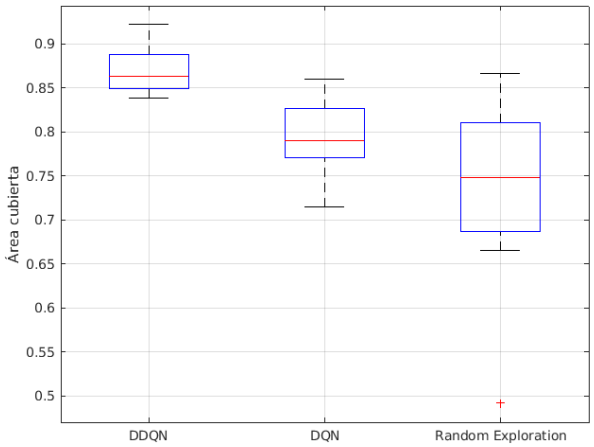


Figura 5.34 Comparativa del *coverage* tras 20 tests..

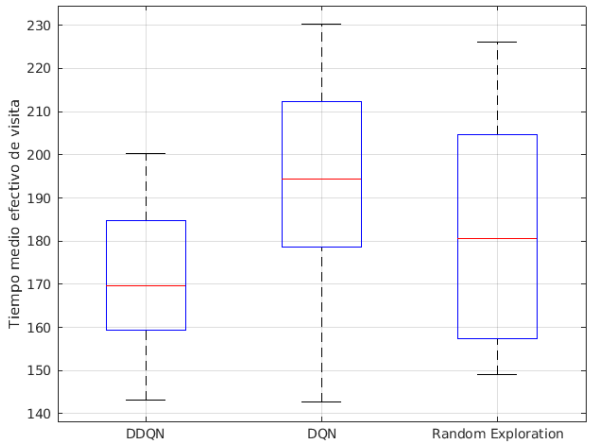


Figura 5.35 Comparativa del T^{eff}_{mean} tras 20 tests..

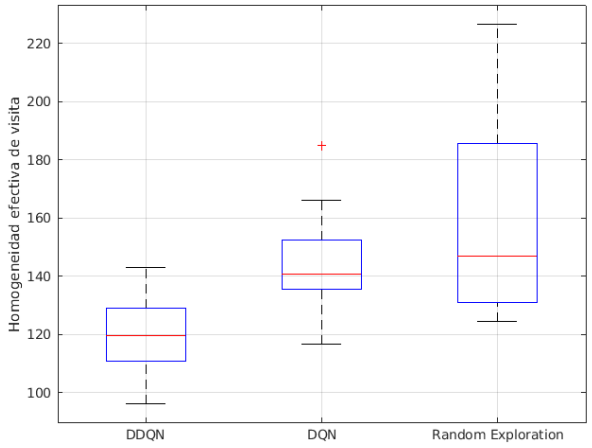


Figura 5.36 Comparativa de H^{eff}_{mean} tras 20 tests..

Se puede ver que el algoritmo entrenado en última instancia devuelve mejores resultados que con una red DQN simple y con una exploración aleatoria de 300 pasos. La cobertura alcanza máximos del 96 % en el mejor de los casos y se consigue reducir el tiempo de cobertura máximo por debajo de los 200 movimientos. La homogeneidad de visita se reduce también significativamente, lo que refrenda la bondad del resultado al reducir las diferencias de tiempo de visita de una celda a otra.

5.6 Resultado con Double Deep Q-Learning con importancia no homogénea

De la misma forma que antes, ejecutaremos el algoritmo Double DQN en las condiciones nominales. El resultado devuelve un entrenamiento exitoso muy distinto del que se obtuvo usando solo DQN.

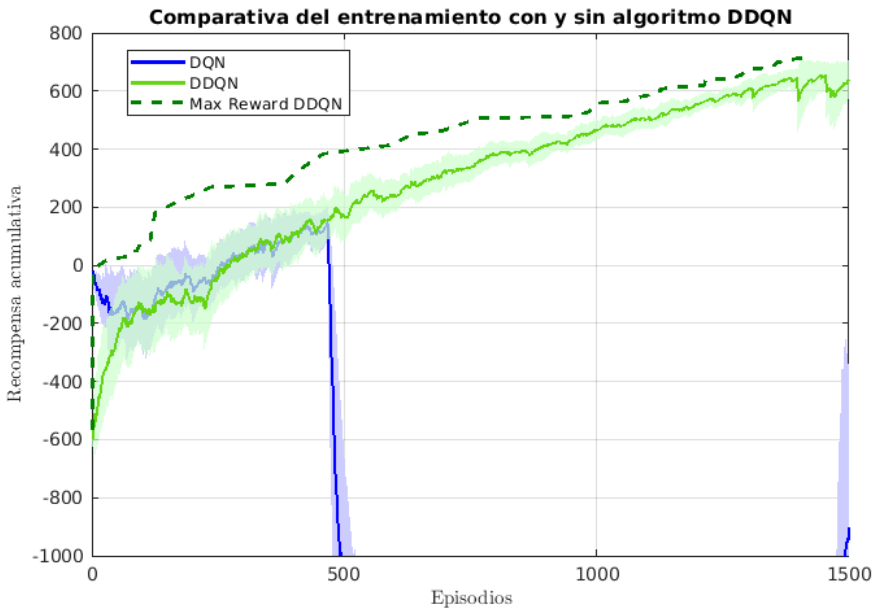


Figura 5.37 Resultado del entrenamiento DDQN para $\mathbf{R}_{est}$ variable.

Tenemos que fijarnos en que las recompensas máximas obtenidas son inferiores a las del caso anterior porque ahora las zonas de máximo interés en el mapa son muy pocas.

A continuación se expone el mapa de frecuencia de visitas y el mapa de $\mathbf{R}_{est}$.

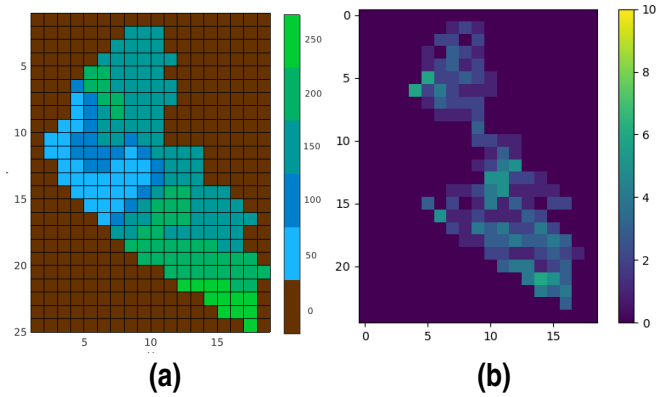


Figura 5.38 Mapa de interés estático (a) y mapa de número de visitas (b).

Se tiene ahora que el agente es capaz de operar tal que las zonas más importantes son más frecuentemente visitadas (figura 5.38(b)).

Se puede observar cómo las zonas de mayor importancia en el mapa reciben una mayor cantidad de visitas que las de importancia menor. También surge un comportamiento aprendido en el que se tiende a ignorar el sector de zonas menos importantes. Este comportamiento se entiende puesto que el algoritmo encuentra una solución aceptable en revisitar mucho zonas de alta importancia más que visitar alguna vez zonas sin casi importancia.

Este comportamiento asociado podría resultar adecuado cuando hay zonas del mapa que no se quieren visitar por conocerse ya estudiadas exhaustivamente o poco contaminadas.

5.6.1 Ventajas y desventajas del Double Deep Q-Learning

Se ha visto que el algoritmo de Double-DQN es capaz de hacer desaparecer el *catastrophic forgetting* del entrenamiento en ambos casos de exploración del lago Ypacaraí. Esta ventaja es fundamental para encontrar la eficiencia en el aprendizaje, puesto que con 1500 episodios se consiguen resultados buenos sin perder lo aprendido anteriormente. De otra forma, tendríamos que entrenar durante mucho más tiempo esperando a que el comportamiento "a tirones" del algoritmo DQN obtuviese recompensas mayores.

El coste de implementar el algoritmo de Double DQN es muy pequeño: solo es necesaria la duplicación de la red Q del modelo en una red de *target*. Esta última red, como se vio en el algoritmo 4, no se entrena de ninguna forma, sino que recibe cada cierto número de episodios los pesos entrenados, por lo que no conlleva más esfuerzo computacional sino algo más de memoria (lo que ocupe la red neuronal en RAM).

Por ello, Double-DQN se convierte en una buena solución para ambos modos de exploración del lago Ypacaraí y se demuestra como un algoritmo robusto en los términos nominales en los que se ha entrenado la red.

5.7 Análisis del efecto de los hiperparámetros

Se ha dicho en varias ocasiones que el valor de los hiperparámetros de entrenamiento así como la estructura de la red neuronal de cada algoritmo de Deep RL influirá cierto grado en el desempeño final del aprendizaje.

Para simplificar el análisis, nos limitaremos a verificar el efecto de los parámetros en el segundo caso, con una matriz no homogénea, que es el caso más general de ambos.

Tamaño del batch

Se ha entrenado la red, partiendo de los valores nominales e imponiendo distintos tamaños de batch: 32, 64, 128, 250 (nominal) y 512.

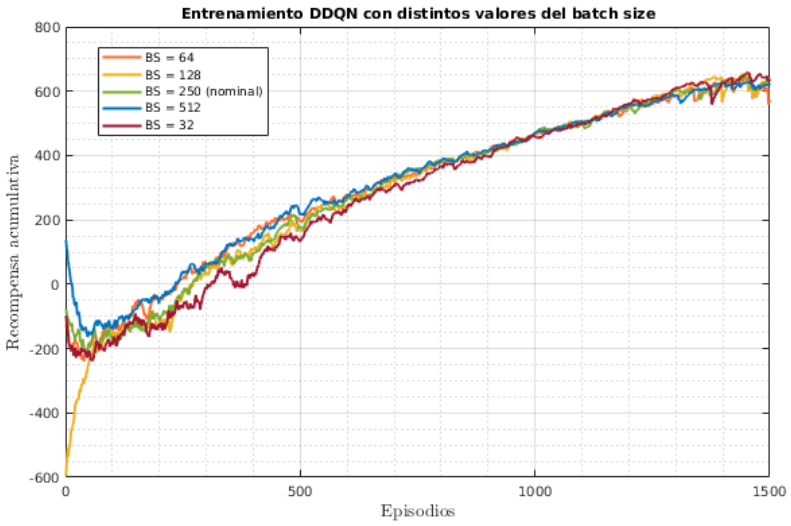


Figura 5.39 Resultado del entrenamiento para varios valores del *batch size*.

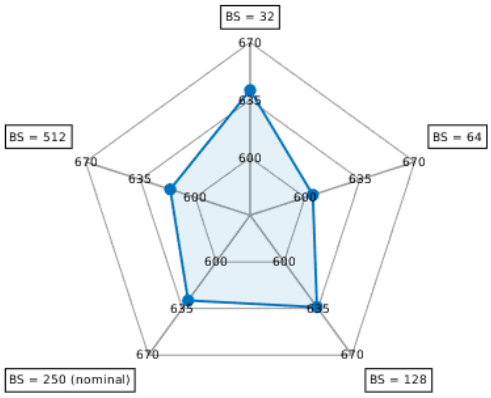


Figura 5.40 Media de recompensas de 50 episodios para cada *batch size*.

El resultado viene a contrastar las investigaciones de [21], en el que, a diferencia de lo que pueda ser intuitivo, un batch mayor no implica un mayor aprendizaje del agente. El aprendizaje se ve influenciado por la relación entre el número de experiencias muestreadas en cada paso de entrenamiento (el tamaño del batch) y el tamaño del *memory replay*. Según las conclusiones de [21]. Una relación alta nos acerca a un método más del tipo *on policy* y esto incide en una menor capacidad de entrenamiento. Además de un coste computacional mayor. Por contra, una relación excesivamente baja hace que el entrenamiento vea pocas asociaciones (s,a,s',r) cada vez resultando en un desempeño también pobre.

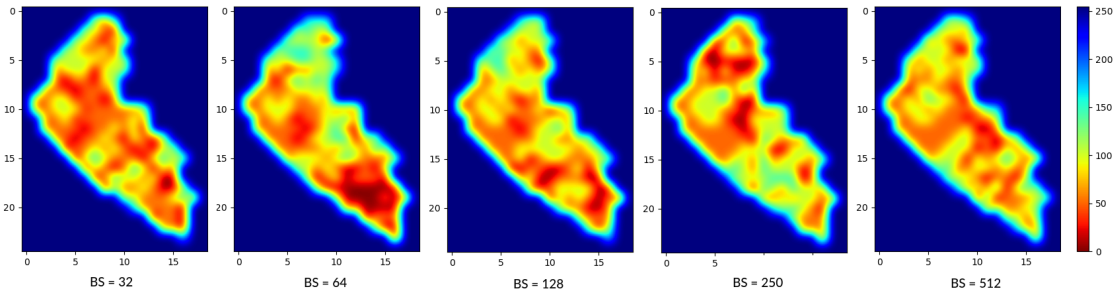


Figura 5.41 Mapa de calor para distintos valores del *batch size*.

Learning rate

El *learning rate* influirá en lo rápido que se incorpora el gradiente de la función de coste L a cada parámetro de la red neuronal. Si este parámetro es muy grande, puede causar inestabilidades en el entrenamiento y si es muy pequeño, la convergencia será excesivamente lenta y puede que se llegue a una solución sub-óptima. Aquí se han probado 4 valores de este parámetro: $1E-1$, $1E-2$, $1E-3$ (nominal) y $1E-4$:

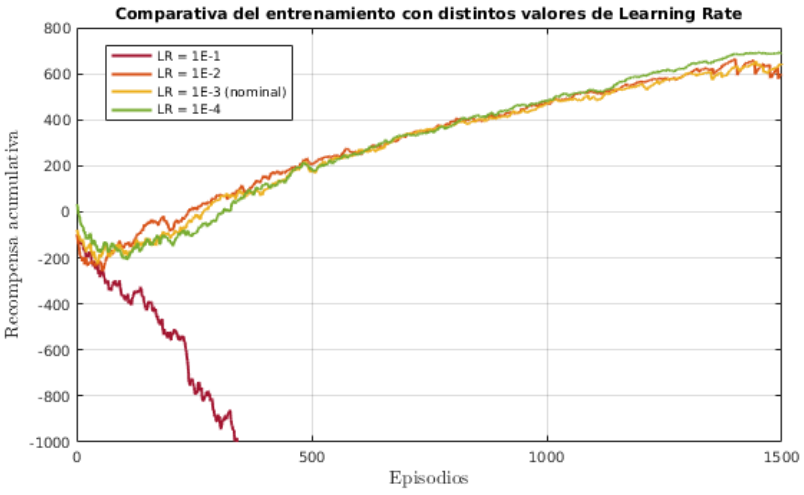


Figura 5.42 Resultado del entrenamiento para varios valores del *learning rate*.

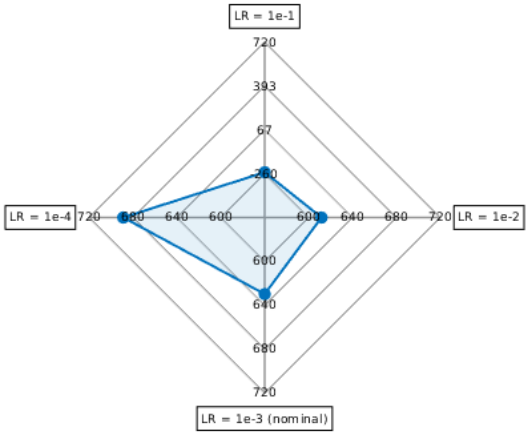


Figura 5.43 Media de recompensas de 50 episodios para cada *learning rate*.

Se observa que para un valor del *learning rate* demasiado grande, el entrenamiento se vuelve inestable, por lo que debe mantenerse en un valor por debajo del nominal.

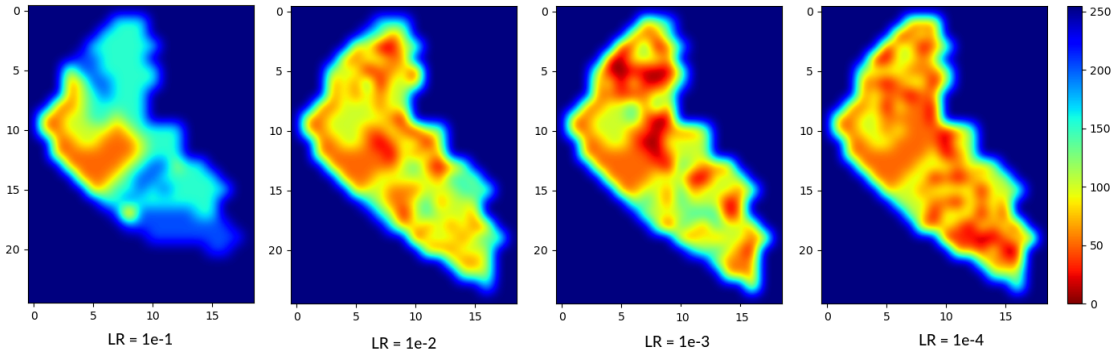


Figura 5.44 Mapa de calor para distintos valores del *learning rate*.

Épsilon-decay

El valor del decaimiento de ϵ determina el nivel de avariciosidad de la política de comportamiento según el algoritmo $epsilon - greedy$ explicado en el capítulo 4. Se han probado 3 valores del decaimiento de ϵ, de modo que cada uno haga de ϵ su valor mínimo (0.01) en un número de episodio completo. Estos tres valores son: 0.002 (ϵ llega a su mínimo en el episodio 500), 0.001 (ϵ llega a su mínimo en el episodio 1000) y 0.0007 (ϵ llega a su mínimo en el episodio 1400).

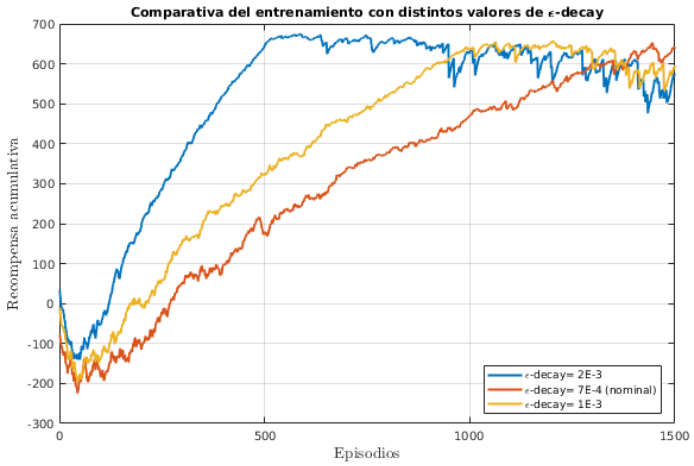


Figura 5.45 Resultado del entrenamiento para varios valores del $\epsilon - decay$.

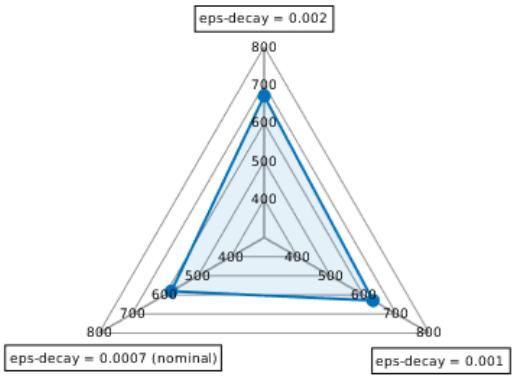


Figura 5.46 Media de recompensas de 50 episodios para cada valor de $\epsilon - decay$.

Se puede ver que cuanto más pequeño sea el valor de $\epsilon - decay$, antes llegaremos al establecimiento de la curva. Un valor muy alto podría resultar en una carencia exploratoria luego en un fracaso a la hora de generalizar el aprendizaje por falta de experiencias previas. Un valor muy pequeño causa un entrenamiento muy lento pero más estable, puesto que se exploran muchas más acciones en busca de la política Q óptima. Se tiene que el entrenamiento nominal puede mejorarse (en el sentido de la rapidez y de el desempeño) si se aumenta el valor de $\epsilon - decay$.

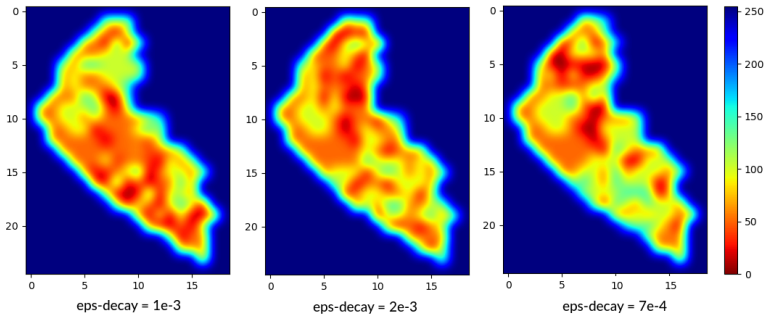


Figura 5.47 Mapa de calor para distintos valores del $\epsilon - decay$.

Tamaño de la red

Se ha probado a aumentar y disminuir la red neuronal de tamaño, manteniendo su estructura diseñada para comprobar cómo afecta la cantidad de neuronas y filtros convolucionales al entrenamiento. Una red excesivamente grande tendrá muchas de sus neuronas con unos pesos que tienden a 0 y una red muy pequeña carece de plasticidad para generalizar la función Q óptima, por lo que deberá ajustarse a un valor intermedio.

Se han probado las siguientes configuraciones, partiendo de la estructura nominal descrita en el apartado 5.3:

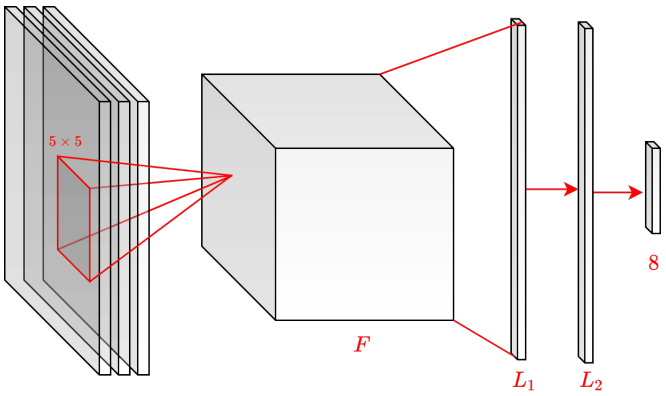


Figura 5.48 Estructura de la red neuronal según los parámetros de tamaño (F,L_1,L_2).

Tabla 5.2 Distintos tamaños de la red neuronal Q.

Tamaño	Número de filtros F	Neuronas capa lineal L_1	Neuronas capa lineal L_2
x2	16	2048	2048
x1 (nom.)	8	1024	1024
x0.5	4	512	512
x0.25	2	256	256
x0.125	1	128	128

El resultado del entrenamiento es:

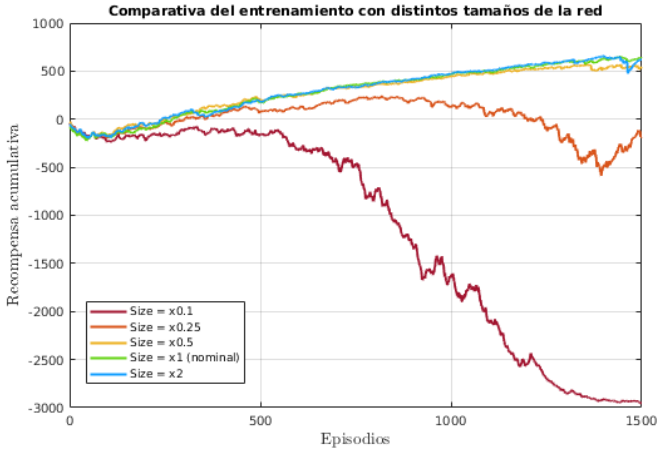


Figura 5.49 Resultado del entrenamiento para varios valores del tamaño de la red.

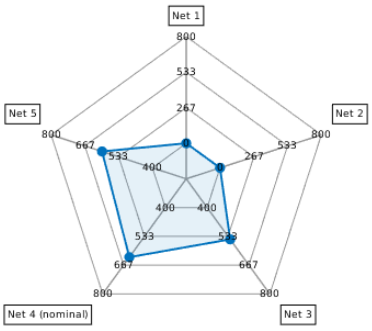


Figura 5.50 Media de recompensas de 50 episodios para cada tamaño de red.

Se observa cómo con el tamaño de la red nominal se tienen buenos resultados. Con un tamaño menor el entrenamiento de la red es inestable y con un tamaño mayor no conseguimos resultados mejores, por lo que se establece que la estructura inicial planteada es la que mejores resultados da.

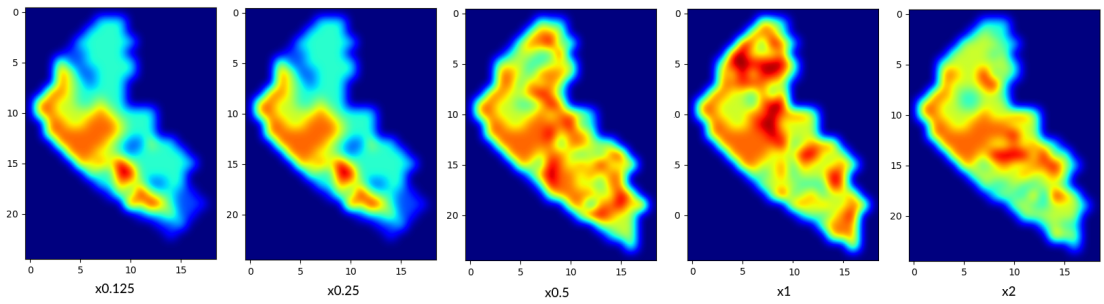


Figura 5.51 Mapa de calor para distintos valores del tamaño de la red.

5.8 Comparación con otros métodos

Una vez establecida la incidencia de los hiperparámetros en el entrenamiento se escoge los valores que mejores resultados proporcionan. La tabla de hiperparámetros sintonizada será:

Tabla 5.3 Hiperparámetros sintonizados.

Hiperparámetro	Componente
Replay Memory Size	10.000
Memory Batch	64
ϵ-decay	0.002
Learning Rate	$1e^{-4}$
γ	0.99
Número de episodios	1.500
Movimientos por episodio	300

Para comprobar que los algoritmos devuelven resultados positivos comparados con otros algoritmos, realizaremos una comparación con un método típico de *path-planning* de tipo barrido (*lawn mower* - corta-césped). El algoritmo de tipo barrido garantiza llegar a todas las zonas del mapa alguna vez de forma regular, pero no tiene en cuenta la importancia de las zonas ni la redundancia útil:

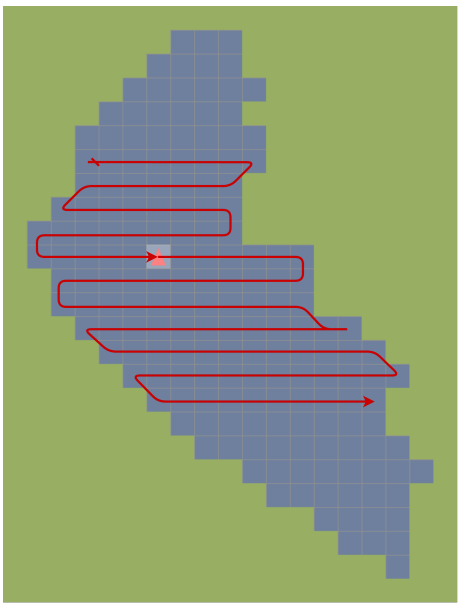


Figura 5.52 Trayectoria de barrido horizontal..

5.8.1 Caso de cobertura homogénea

El resultado del entrenamiento con los mejores valores de los hiperparámetros se puede ver en la figura 5.54.

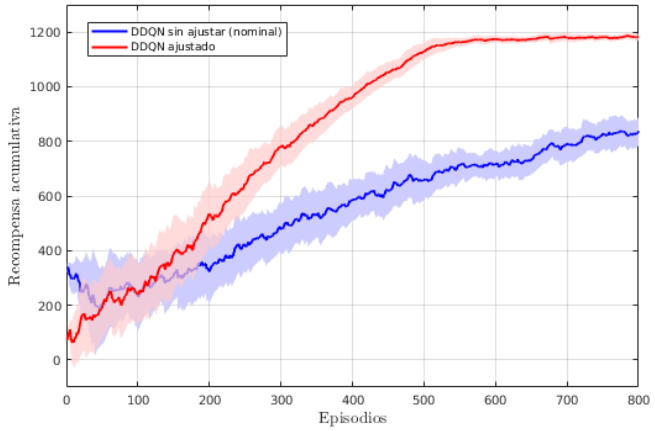


Figura 5.53 Resultado del entrenamiento con los parámetros ajustados para el caso homogéneo.

Se puede ver en la imagen 5.54 que la mejoría es significativa. El cambio de parámetros permite aumentar la recompensa máxima obtenida en unos 400 puntos lo que evidencia la sensibilidad del entrenamiento a los parámetros.

Comparando ahora el algoritmo con el método del cortacésped:

La mejoría puede verse no solo en la homogeneidad sino en la cobertura, que es casi total en el algoritmo propuesto. En la tabla 5.4 se pueden ver las figuras de mérito del algoritmo propuesto respecto del algoritmo del cortacésped:

Tabla 5.4 Figuras de mérito para los algoritmos propuestos del caso homogéneo.

	Cortacésped		Tunned DDQN		Nominal DDQN	
	Mean	Std Dev.	Mean	Std Dev.	Mean	Std Dev.
Coverage	1	-	0.93	0.02	0.91	0.01
T_{eff}	178.5	-	115.35	4.47	122.59	4.48
H_{eff}	25.41	-	59.91	2.29	64.00	3.03

Se obtiene una disminución (de media) del periodo de espera de cada casilla a ser visitada del orden del 34% respecto del valor obtenido con el algoritmo del cortacésped.

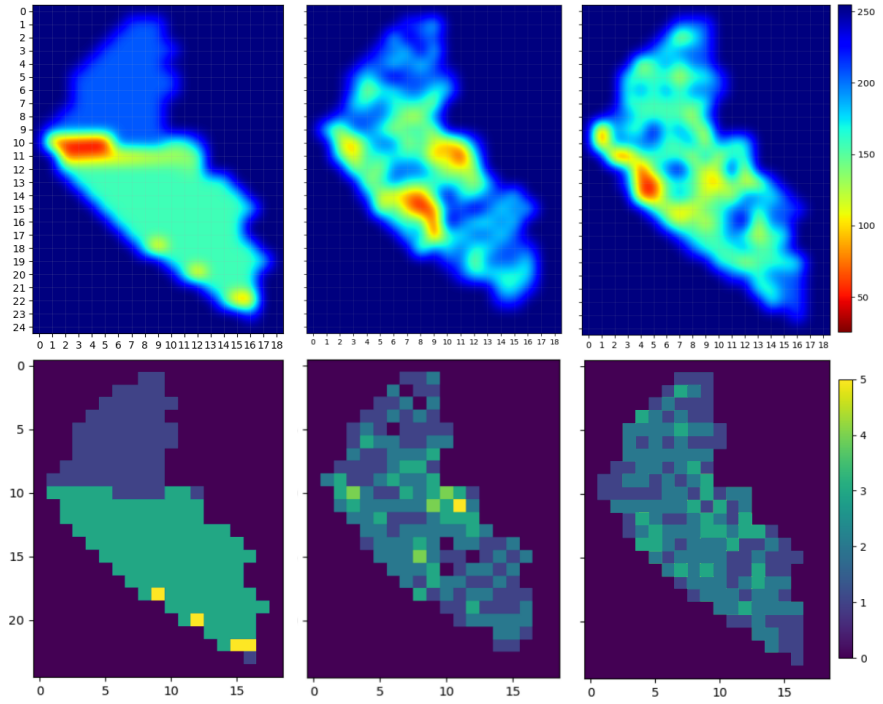


Figura 5.54 Comparación de las matrices de frecuencia de visitas (abajo) y de importancia relativa (arriba) para el algoritmo de corta-césped (izquierda), algoritmo con hiperparámetros sin sintonizar (centro) y sintonizado (derecha).

5.8.2 Caso de cobertura no homogénea

El resultado del entrenamiento con los mejores valores de los hiperparámetros es:

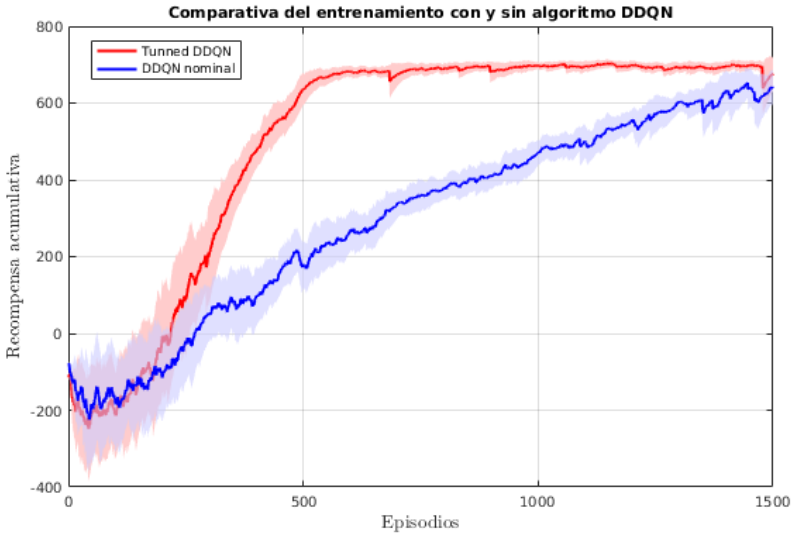


Figura 5.55 Resultado del entrenamiento con los parámetros ajustados para el caso no homogéneo.

Se observa que la distribución, tras 300 pasos es mucho más homogénea en el algoritmo propuesto

Se comprueba que no solo ganamos en performance y estabilidad, sino en velocidad a la hora de llegar al punto de entrenamiento final (sobretodo por el aumento del ϵ-decay). Además, se observa en la figura 5.55 que la desviación típica de la recompensa por partida es mucho menor en el caso sintonizado (sombra roja) que en el caso sin sincronizar (sombra azul). Esto viene a indicar que en distintos episodios el comportamiento es similar y más adecuado cuanta menor desviación tenga (las recompensas se distribuyen más concentradas en torno al óptimo).

Se compara el resultado de la recompensa a lo largo de un episodio con distintos algoritmos: cortacésped, DDQN sin sintonizar, DDQN propuesto y exploración aleatoria en la figura 5.56.

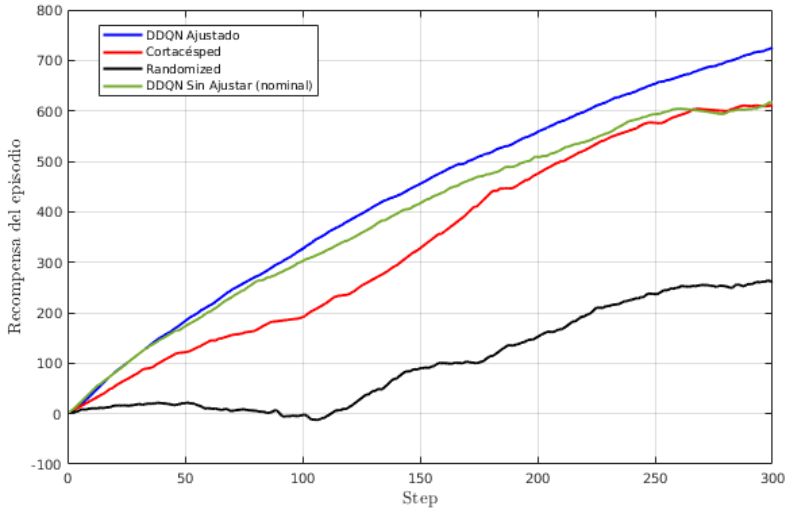


Figura 5.56 Recompensa a lo largo de un episodio para distintos algoritmos en el caso no homogéneo..

Se tiene que la mejora de la cobertura respecto del algoritmo de cortacésped es de más del 19 %. Tiene sentido este resultado puesto que el algoritmo propuesto es capaz de discriminar las zonas por importancia y ajustarse a su huella temporal de visitas y a la máscara de importancia que le proporciona el estado.

El resultado de las matrices de visitas y matriz de importancia relativa para cada caso es el se puede ver en la figura 5.57

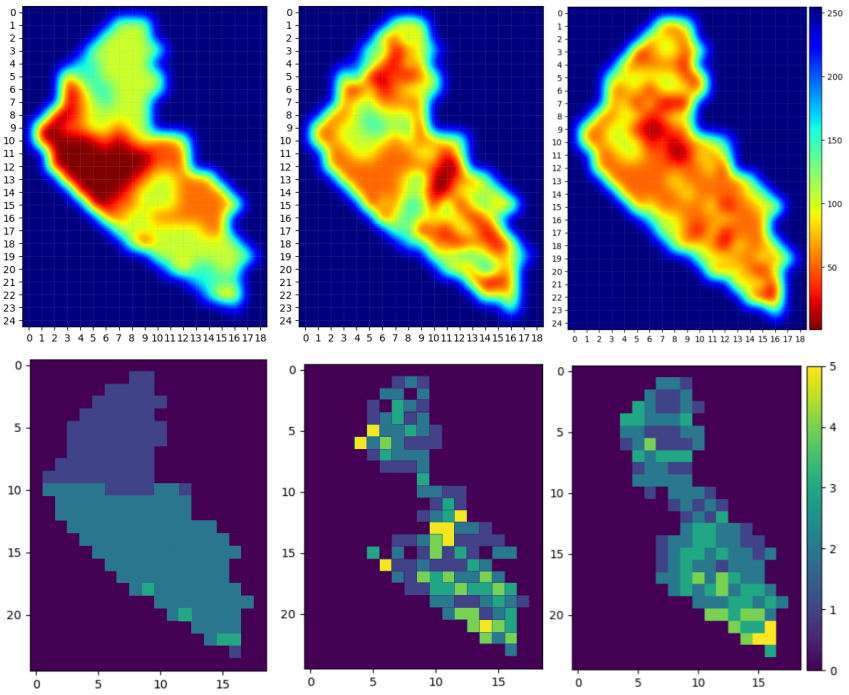


Figura 5.57 Comparación de las matrices de frecuencia de visitas (abajo) y de importancia relativa (arriba) para el algoritmo de corta-césped (izquierda), algoritmo con hiperparámetros sin sintonizar (centro) y sintonizado (derecha).

Está claro que el algoritmo propuesto es capaz de cubrir más eficientemente el lago con distribución no homogénea. En las matrices de importancia relativa se puede observar que el agente cubre deforma ponderada las zonas (visita más veces las zonas interesante y desprecia las que no son interesantes o ya han sido visitadas). Esto es un buen principio para el planteamiento multi-agente, donde se comparte el conocimiento de la cobertura del mapa.

Se ha realizado una comparativa de las figuras de mérito respecto del algoritmo de cortacésped. Los resultados figuran en la tabla 5.5.

Tabla 5.5 Figuras de mérito para los algoritmos propuestos del caso no homogéneo.

	Cortacésped		DDQN propuesto	
	Mean	Std Dev.	Mean	Std Dev.
Coverage	1	-	0.88	0.02
T_{eff}	130.38	-	100.69	3.29
H_{eff}	54	-	43.02	3.42

Capítulo 6: Conclusiones y líneas futuras

6.1 Conclusiones

Se ha podido observar que la metodología del Reinforcement Learning, más concretamente el Deep Reinforcement Learning, es cada vez más utilizada para la resolución de problemas complejos. Su versatilidad permite el entrenamiento en tareas que, hasta ahora, solo podían ser resueltas con un modelo complejo y mediante fuerza computacional. Las bondades en estas técnicas salen a relucir también en la robótica donde existen tareas de soluciones complejas con espacio de acciones grandes y estados prácticamente infinitos: el algoritmo del Deep Q-Learning es capaz de trabajar directamente con la experiencia previa sin necesidad de modelar el entorno. Además, al ser un algoritmo de tipo *off-policy* el algoritmo es capaz de encontrar la estrategia óptima sin importar qué política de comportamiento tenga para explorar el espacio de la dupla estado-acción.

Para el caso de la exploración del lago Ypacaraí, se ha demostrado que el Deep Reinforcement Learning es capaz de devolver resultados positivos tanto en la cobertura homogénea de la superficie navegable del lago (con mejoras del 30 % en el tiempo medio de espera de cada zona a ser visitada) respecto de otros métodos clásicos como el algoritmo de cortacésped o de búsqueda aleatoria. Para el caso de la exploración y cobertura dada una importancia no homogénea del lago, tenemos que se mejora la recompensa obtenida del orden del 20 % respecto del *pathplanning* de tipo cortacésped. Además de los resultados positivos en las figuras de mérito, se tiene un algoritmo capaz de aprender para *cualquier* entorno lacustre independientemente de su morfología, lo que supone un punto a favor de las técnicas empleadas.

Por otra parte, se ha demostrado también que los algoritmos de DQN y DDQN dependen mucho del valor de los hiperparámetros empleados. En este sentido, al haber realizado una breve sintonización del algoritmo DDQN, se han alcanzado resultados significativamente mejores, lo que obliga en investigaciones futuras a

dedicar tiempo a encontrar una familia de hiperparámetros adecuados para cada caso.

6.2 Líneas futuras

Este trabajo fue diseñado como planteamiento inicial de un trabajo mayor en forma de tesis doctoral y sus resultados permiten pronosticar toda una línea de investigación nueva en el que los algoritmos de Reinforcement Learning tengan un protagonismo especial. Así, se abren nuevos caminos que explorar en este ámbito.

Una línea importante está en redefinir el tipo de algoritmo de aprendizaje adaptándolo a los últimos avances en Reinforcement Learning. De entre toda la miríada de algoritmos RL, destacamos la utilización de los de tipo *Policy Optimization*, que se centran en encontrar la política óptima más que en el valor de la función Q óptimo, con referentes como *Advantage Actors Critic - A2C* y *Asynchronous Advantage Actors Critic - A3C*. Estos dos algoritmos han devuelto, con diferencia, los mejores resultados en múltiples tareas. Además, su capacidad de entrenamiento paralelo permite unos tiempos de aprendizaje mucho menores. En estos algoritmos está la verdadera eficiencia a la hora de encontrar soluciones óptimas a las tareas planteadas. Esto cobra especial importancia a la hora de introducir el paradigma multiagente, en el que la exploración será conjunta y habrá de tenerse en cuenta que los agentes pueden colisionar.

Otra línea importante que se abre es la inclusión de redes neuronales concurrentes para mejorar el desempeño final. Las redes neuronales concurrentes permiten realimentar la salida de las neuronas para detectar e integrar en la estimación las dependencias temporales de un movimiento a otro (la figura 6.1 es un ejemplo de neurona concurrente). Si bien en este trabajo se ha considerado que no existen dependencias temporales (o al menos no significativas debido a que la observación del estado es completa), en un futuro caso en el que el problema se reformule como un Proceso de Decisión de Markov no completamente observable (solo tenemos una parte del estado)

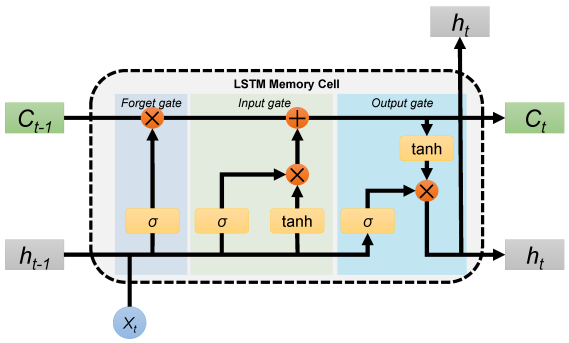


Figura 6.1 Neurona concurrente de tipo LSTM.

En un futuro paso de investigación se ha considerado también el incluir niveles de incertidumbre con origen en los sensores dentro del aprendizaje. Como el interés de la zona y la recompensa están asociados a tomar información de unos sensores físicos, al tratar un caso más realista tendremos que incluir incertidumbres de medidas lo que debe verse reflejado en una nueva forma del mapa. Las incertidumbres del escenario convierten necesariamente el problema en un proceso de decisión no determinista con dependencias temporales de observación.

Finalmente, el siguiente paso es integrar los dos algoritmos (el de cobertura homogénea y el de cobertura no homogénea) en el ASV. Se plantea un esquema de control (véase figura -) en el que, partiendo del desconocimiento del estado del mapa, se ejecute el primer algoritmo. Con el mapa de estado del lago construido (gracias a los sensores y al algoritmo) se genera un mapa de importancia relativa (donde las zonas de mayor contaminación tienen mayor importancia). En la siguiente iteración de exploración se utiliza el algoritmo de planificación no homogénea. Así, se podrán monitorizar los cambios del estado del lago poniendo énfasis en las zonas de mayor importancia detectadas en cada búsqueda.

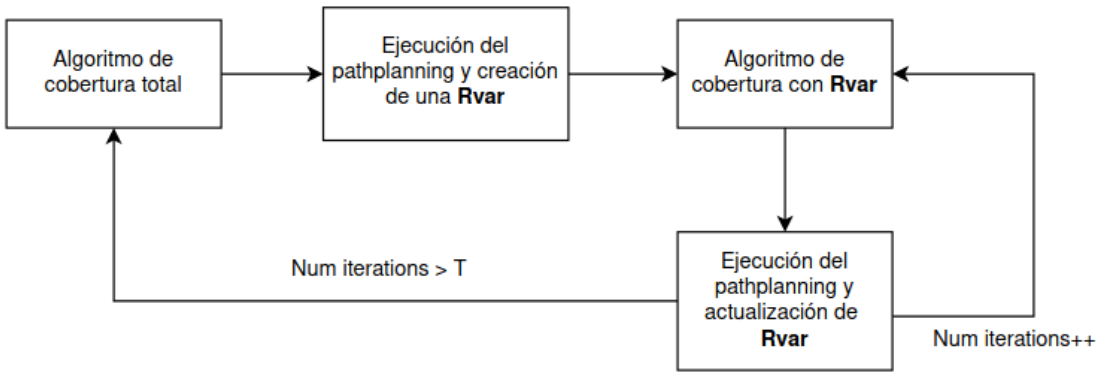


Figura 6.2 Algoritmo propuesto para implementación..

Apéndice A

Interfaz Gráfica

Para poder facilitar que el usuario pueda acceder a los algoritmos de entrenamiento, se ha desarrollado una *GUI* en Python mediante la librería *PySimpleGUI*. Con esta librería se puede construir muy fácilmente aplicaciones con ventanas compatibles con *GNOME* (Ubuntu) o Windows.

La aplicación permite seleccionar el mapa de ponderación en formato CSV (si no está en ese formato, el programa lanza un aviso y no permite continuar):

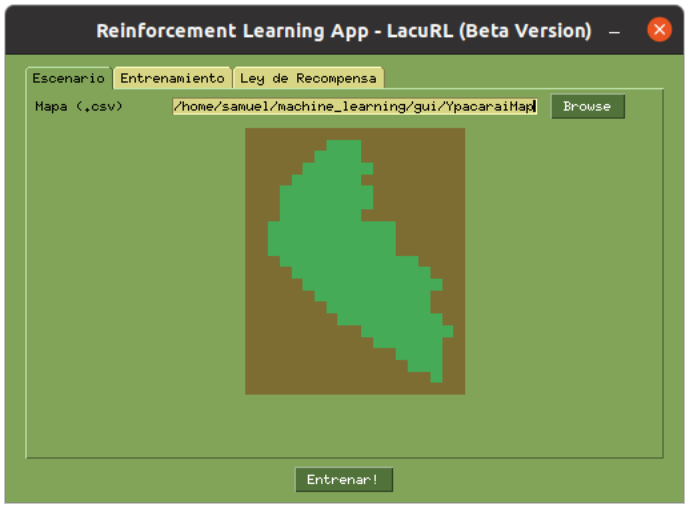


Figura A.1 Selección del mapa.

La aplicación permite modificar los hiperparámetros de entrenamiento (tamaño del *batch*, episodios, etc), así como el algoritmo que se desea utilizar para entrenar. Un panel de *debuggin* muestra la información del entrenamiento (como complemento a la gráfica que genera el propio programa):

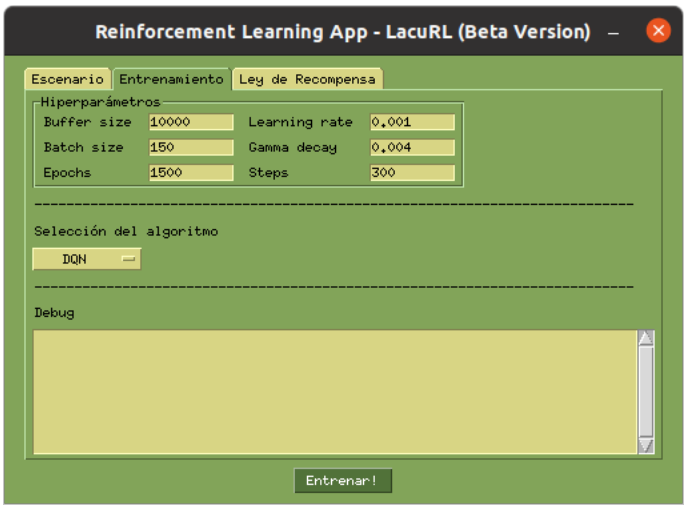


Figura A.2 Selección del mapa.

Para evitar errores, se lanza un aviso en el caso de que algún hiperparámetro no tenga el formato correcto o no tenga un valor válido:

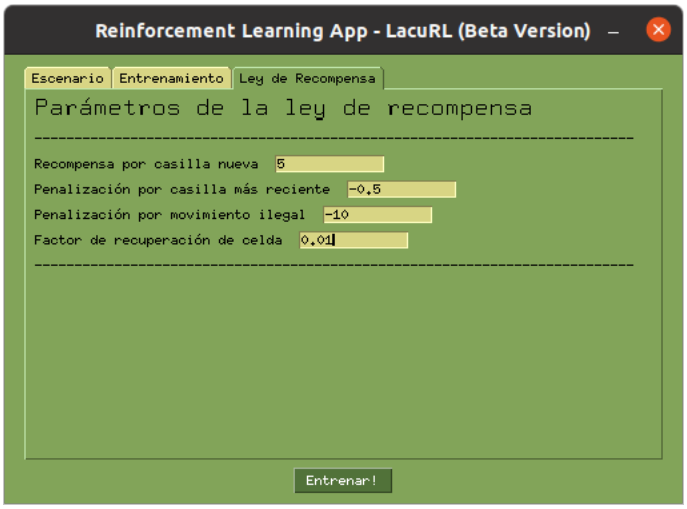


Figura A.3 Selección de hiperparámetros.

Se puede modificar también la ley de recompensa, imponiendo valores distintos de los nominales a las penalizaciones por movimiento ilegal y demás parámetros:

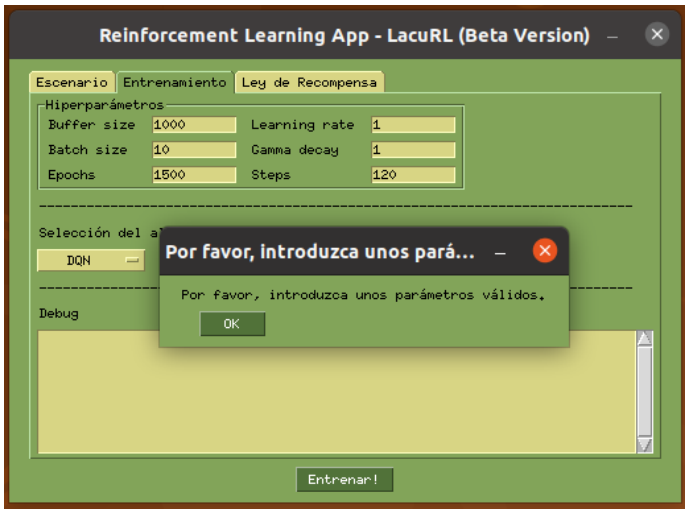


Figura A.4 Selección de la ley de recompensa.

Finalmente, al ejecutar el algoritmo, se genera un *.txt* con el resumen del entrenamiento. Al finalizar el entrenamiento, se tendrá en formato *.pt* los pesos entrenados de la red en el episodio de mayor recompensa (*best*) y en el último episodio entrenado (*last*), ya que no tienen por qué coincidir:

Figura A.5 Confirmación del entrenamiento.

Índice de Figuras

Índice de Tablas

Bibliografía

[1] J. C. Ho and A. M. Michalak, "Challenges in tracking harmful algal blooms: A synthesis of evidence from Lake Erie," *Journal of Great Lakes Research*, vol. 41, no. 2, pp. 317–325, 2015. [Online]. Available: http://dx.doi.org/10.1016/j.jglr.2015.01.001

[2] A. Rodríguez, "Estudio de la contaminación del lago Ypacaraí e introducción de un dron acuático para el monitoreo de la calidad del agua Autor," pp. 1–93, 2019.

[3] E. M. López Arzamendia, S. L. Toral Marín, and D. Gutiérrez Reina, "Reactive Evolutionary Path Planning For Autonomous Surface Vehicles in Lake Environments," no. December, 2018.

[4] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-Learning," *30th AAAI Conference on Artificial Intelligence, AAAI 2016*, pp. 2094–2100, 2016.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015. [Online]. Available: http://dx.doi.org/10.1038/nature14236

[6] A. Krishna Lakshmanan, R. Elara Mohan, B. Ramalingam, A. Vu Le, P. Veerajagadeshwar, K. Tiwari, and M. Ilyas, "Complete coverage path planning using reinforcement learning for Tetromino based cleaning and maintenance robot," *Automation in Construction*, vol. 112, no. May 2019, p. 103078, 2020. [Online]. Available: https://doi.org/10.1016/j.autcon.2020.103078

[7] M. Theile, H. Bayerlein, R. Nai, D. Gesbert, and M. Caccamo, "UAV Coverage Path Planning under Varying Power Constraints using Deep Reinforcement Learning," 2020. [Online]. Available: http://arxiv.org/abs/2003.02609

[8] C. Piciarelli and G. L. Foresti, "Drone patrolling with reinforcement learning," *ACM International Conference Proceeding Series*, no. 1, pp. 1–6, 2019.

[9] R. Shah, Y. Jiang, J. Hart, and P. Stone, "Deep R-Learning for Continual Area Sweeping," 2020. [Online]. Available: http://arxiv.org/abs/2006.00589

[10] F. Niroui, K. Zhang, Z. Kashino, and G. Nejat, "Deep Reinforcement Learning Robot for Search and Rescue Applications: Exploration in Unknown Cluttered Environments," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 610–617, 2019.

[11] J. Xiao, G. Wang, Y. Zhang, and L. Cheng, "A Distributed Multi-Agent Dynamic Area Coverage Algorithm Based on Reinforcement Learning," *IEEE Access*, vol. 8, pp. 33 511–33 521, 2020.

[12] S. Ravichandiran, *Hands-On Reinforcement.* Packt, 2018.

[13] A. A. Adepegba, S. Miah, and D. Spinello, "Multi-agent area coverage control using reinforcement learning," *Proceedings of the 29th International Florida Artificial Intelligence Research Society Conference, FLAIRS 2016*, pp. 368–373, 2016.

[14] L. C. Oporto, D. A. Ramírez, J. D. Varela, and C. E. Schaerer, "Analysis of Contaminant Transport under Wind Conditions on the Surface of a Shallow Lake," *Mecánica Computacional*, vol. 34, no. 31, pp. 2155–2163, 2016. [Online]. Available: http://www.cimapy.org/images/descargas/cabibeskry/Oporto{_}ENIEF{_}Analysis{_}of{_}contaminant{_}transport.pdf

[15] M. M. Drugan, "Reinforcement learning versus evolutionary computation: A survey on hybrid algorithms," *Swarm and Evolutionary Computation*, vol. 44, no. August 2017, pp. 228–246, 2019. [Online]. Available: https://doi.org/10.1016/j.swevo.2018.03.011

[16] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, "Dueling Network Architectures for Deep Reinforcement Learning," *33rd International Conference on Machine Learning, ICML 2016*, vol. 4, no. 9, pp. 2939–2947, 2016.

[17] S. Bhatnagar, R. Sutton, M. Ghavamzadeh, M. Lee, S. Bhatnagar, R. Sutton, M. Ghavamzadeh, S. Bhatnagar, R. S. Sutton, and M. Ghavamzadeh, "Natural Actor – Critic Algorithms," vol. 45, no. 11, 2013.

[18] M. Lones, *Sean Luke: essentials of metaheuristics*, 2nd ed. Lulu, 2011, vol. 12, no. 3. [Online]. Available: http://cs.gmu.edu/{\protect\T1\textdollar}{%}5Csim{\protect\T1\textdollar}sean/book/metaheuristics/{%}7D{%}7D

[19] B. Liu, Y. Zhang, S. Fu, and X. Liu, "Reduce UAV coverage energy consumption through actor-critic algorithm," *Proceedings - 2019 15th International Conference*

on Mobile Ad-Hoc and Sensor Networks, MSN 2019, pp. 332–337, 2019.

[20] A. Cahill, "Catastrophic Forgetting in Reinforcement-Learning Environments," Ph.D. dissertation, University of Otago, 2010.

[21] W. Fedus, P. Ramachandran, R. Agarwal, Y. Bengio, H. Larochelle, M. Rowland, and W. Dabney, "Revisiting Fundamentals of Experience Replay," *Proceedings of the 36th International Conference on Machine Learning*, 2019.

[22] V. Mnih, A. P. Badia, L. Mirza, A. Graves, T. Harley, T. P. Lillicrap, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," *33rd International Conference on Machine Learning, ICML 2016*, vol. 4, pp. 2850–2869, 2016.

[23] Z. Wang, V. Mnih, V. Bapst, R. Munos, N. Heess, K. Kavukcuoglu, and N. De Freitas, "Sample efficient actor-critic with experience replay," *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, no. 2016, 2019.