Universidad de Valladolid

FACULTAD DE CIENCIAS

TRABAJO FIN DE GRADO

GRADO EN ESTADÍSTICA

Aplicación de algoritmos de aprendizaje por refuerzo al control de un estimulador cerebral

Autor

D. Pablo Parte Renedo

Tutoras

D.ª Itziar Fernández Martínez

D.ª Yolanda Larriba González

Curso 2023-2024

Resumen

La estimulación cerebral profunda es una técnica empleada en el tratamiento de algunas enfermedades neurológicas que no se pueden tratar adecuadamente con medicación. Consiste en la implantación de electrodos en regiones específicas del cerebro, que envían impulsos eléctricos para modular la actividad cerebral de los pacientes. La selección de los parámetros óptimos para la estimulación es un problema abierto, que se puede abordar con diferentes estrategias.

En este trabajo se estudia la aplicación de una metodología de aprendizaje por refuerzo para la regulación automática de la amplitud de los pulsos eléctricos generados en la estimulación cerebral en función de las señales neurológicas observadas. Los algoritmos propuestos se aplican sobre un modelo neuronal simulado, en el que el objetivo del problema se caracteriza como la supresión de las oscilaciones colectivas de una población de neuronas sincronizadas.

Abstract

Deep brain stimulation is a technique used in the treatment of certain neurological diseases that cannot be adequately controlled with medications. It involves the implantation of electrodes in specific brain regions, which send electrical impulses to modulate the brain activity of patients. Optimal selection of parameters for the stimulation is an open problem, that can be tackled with different strategies.

In this work, we study the application of a reinforcement learning methodology to automatically regulate the amplitude of the generated electrical pulses based on observed neurological signals. The proposed algorithms are applied to a simulated neuronal model, where the objective of the problem is characterized as suppression of collective oscillations of a population of synchronized neurons.

Agradecimientos

A mis tutoras Itziar y Yolanda, por haber confiado en mí para desarrollar este proyecto y por todo el tiempo que han dedicado a supervisarlo. Sus propuestas, correcciones y sugerencias han tenido una contribución invaluable a la calidad del resultado final.

A mi familia, y en especial a mi madre, por todo el apoyo que me han proporcionado durante mi etapa universitaria.

Índice general

Índice de figuras

Índice de cuadros

1 Introducción

La estimulación cerebral profunda (DBS, del inglés *Deep Brain Stimulation*) es una técnica empleada para el tratamiento de varias enfermedades neurológicas, como la enfermedad de Párkinson o el temblor esencial [1]. Consiste en la implantación quirúrgica de electrodos que envían impulsos eléctricos de alta frecuencia dirigidos a zonas específicas del cerebro [1]. Los electrodos están conectados a un controlador programable que regula la amplitud, anchura y frecuencia de los pulsos eléctricos generados [1]. Si bien los mecanismos de acción de la DBS no se conocen con certeza, una hipótesis que se sostiene con fuerza es que las enfermedades mencionadas tienen su origen en determinados grupos de neuronas en estado patológico que emiten señales de forma sincronizada [2]. De esta forma, el objetivo de la DBS consiste en desincronizar estos grupos de neuronas sin suprimir la actividad oscilatoria de las neuronas individuales [2].

Los dispositivos DBS más extendidos en la actualidad emplean un controlador de ciclo abierto, es decir, sin retroalimentación, lo que significa que las acciones del controlador no tienen en cuenta el valor de las variables del proceso que está siendo controlado [3]. Un ejemplo es un sistema de calefacción controlado únicamente por un temporizador, que se enciende a una determinada temperatura durante el tiempo especificado por el usuario. En estos dispositivos, los parámetros de la estimulación son seleccionados por un médico en función de la evaluación de los síntomas del paciente y se mantienen constantes con independencia de las fluctuaciones en el estado neurológico del paciente [2]. Esta estrategia hace que el dispositivo suministre mayor cantidad de energía de la que sería estrictamente necesaria, lo que puede contribuir a la aparición de efectos secundarios y acelerar la descarga de la batería del dispositivo. Como los sistemas DBS más habituales no tienen baterías recargables, la reducción de la vida de la batería expone a los pacientes a los riesgos de una nueva intervención quirúrgica para reemplazar el dispositivo [4].

Ante estas dificultades, tiene interés implementar la DBS con un controlador de ciclo cerrado, que recibe una señal de retroalimentación del proceso que está controlando para conocer si las acciones están consiguiendo el propósito deseado [3]. Las acciones del controlador dependen de dicha señal de retroalimentación [3]. Volviendo al ejemplo anterior, un sistema de calefacción implementado con un controlador de ciclo cerrado incluiría un termostato para monitorizar la temperatura del edificio, y utilizaría esa información para regular su comportamiento y mantener la temperatura cerca de la temperatura deseada. En estos dispositivos, sería deseable que los parámetros de la estimulación se ajusten automáticamente en función de las señales eléctricas observadas en el cerebro del paciente en tiempo real [5]. Una aproximación a este problema puede plantearse como un problema de optimización cuyo objetivo es encontrar un algoritmo de control que maximice la supresión de la señal colectiva generada por el grupo de neuronas sincronizadas y minimice la cantidad de energía que el dispositivo DBS suministra al cerebro del paciente.

Un posible enfoque para resolver el problema anterior consiste en emplear un modelo que caracterice las dinámicas individuales y colectivas de las neuronas. En la literatura se han propuesto múltiples modelos biofísicos para ello, en los que las señales se describen habitualmente mediante ecuaciones diferenciales o mapas iterados [2]. Sin embargo, la obtención de una solución analítica para el problema de optimización a partir de estos modelos presenta una gran complejidad [6]. Por ello, una estrategia alternativa consiste en el uso de métodos de optimización que no requieran un conocimiento explícito de las dinámicas del sistema neuronal [6]. Las dos principales metodologías con estas características son los algoritmos genéticos y el aprendizaje por refuerzo [6].

Este trabajo se centra en el enfoque del aprendizaje por refuerzo, que ha experimentado un notable desarrollo en los últimos años gracias a los avances en el ámbito de las redes neuronales artificiales.

La traslación de este problema a la práctica clínica no es trivial. La experimentación en pacientes reales tiene importantes dificultades debido a una combinación de aspectos éticos, riesgos para la salud y limitaciones prácticas. En primer lugar, y puesto que se trata de un procedimiento invasivo que requiere cirugía, la DBS solo está indicada como tratamiento de último recurso para pacientes cuyos síntomas no pueden ser controlados adecuadamente con medicación o para aquellos que son intolerantes a la medicación [7]. Este hecho, unido a que la DBS es un procedimiento costoso, limita considerablemente el número de participantes en los estudios. Además, la estimulación eléctrica puede tener efectos secundarios, como cambios en el estado de ánimo, o disfunciones cognitivas o motoras [8]. También hay que tener en cuenta que en estos experimentos hay fuentes de variabilidad no controladas que pueden dificultar la comparación y reproducción de resultados.

Por estos motivos, resulta útil simular las dinámicas de grupos de neuronas en un estado patológico y el efecto que tiene la DBS sobre el sistema neuronal. El desarrollo de un algoritmo de control en un entorno simulado no es un mero ejercicio teórico, ya que existen estrategias para transferir el comportamiento aprendido en un entorno simulado a un entorno real [9]. De acuerdo con lo descrito en [2], este trabajo propone el uso de un simulador que emplea un modelo neuronal simplificado basado en osciladores de FitzHugh-Nagumo [2].

El objetivo de este trabajo es desarrollar un algoritmo de aprendizaje por refuerzo que resuelva el problema de control de la DBS en un entorno simulado. En concreto se propone el estudio de los algoritmos de aprendizaje por refuerzo *Deep Deterministic Policy Gradient* [10] y *Proximal Policy Optimization* [11], extendidos entre la comunidad científica. Para cada uno de ellos, se realiza un diseño experimental con el fin de encontrar la arquitectura de red y la combinación de hiperparámetros más adecuada para el problema. Finalmente, se compara el rendimiento de los dos algoritmos en diferentes configuraciones del simulador.

A continuación se enumeran las asignaturas del Grado que guardan mayor relación con este trabajo.

- **Técnicas de Aprendizaje Automático.** Esta asignatura ha resultado particularmente relevante, puesto que en ella se introducen las técnicas básicas de adquisición automática de conocimiento. Entre otras cosas, se estudia el funcionamiento de las redes neuronales y la optimización basada en descenso del gradiente, que son una pieza fundamental de los algoritmos estudiados en este trabajo. Además, el lenguaje de programación utilizado en dicha asignatura es Python, que también es el lenguaje en el que está implementada la interfaz del simulador y de todas las bibliotecas que se han utilizado en este trabajo.
- **Métodos Estadísticos de Computación Intensiva.** En esta asignatura se estudian con más detalle los fundamentos teóricos del descenso del gradiente.
- **Procesos Estocásticos.** Esta asignatura proporciona la base teórica de los procesos de Markov, que son uno de los pilares del aprendizaje por refuerzo.
- Asignaturas de programación: **Fundamentos de Programación**, **Paradigmas de Programación**, **Programación Orientada a Objetos** y **Computación Estadística**. Aunque en este trabajo no se ha desarrollado una gran cantidad de código, los conocimientos de

programación adquiridos en los primeros cursos han sido necesarios para llevar a cabo los diseños experimentales y elaborar los gráficos de resultados.

- Asignaturas básicas de probabilidad: **Modelos probabilísticos** y **Probabilidad**. No hay que olvidar que los conocimientos básicos adquiridos en estas asignaturas son necesarios para comprender el funcionamiento de los algoritmos que se han empleado

La organización del resto del trabajo es la siguiente. En el capítulo 2 se introducen los conceptos básicos de las redes neuronales, un tipo de aproximadores no lineales que son la base de gran parte de los avances recientes en el campo del aprendizaje automático, y en particular de los algoritmos estudiados en este trabajo. En el capítulo 3 se introduce el paradigma del aprendizaje por refuerzo y se detallan los algoritmos que se van a aplicar. En el capítulo 4 se explica el funcionamiento del simulador con el que se han realizado los experimentos. En el capítulo 5 se detallan los diseños experimentales realizados y en el capítulo 6 se exponen los resultados obtenidos. Por último, en el capítulo 7 se presentan las conclusiones extraídas en este trabajo y las líneas futuras de investigación.

2 Redes neuronales artificiales

Las redes neuronales son una categoría de modelos de aprendizaje automático inspirados en las conexiones neuronales del cerebro humano. Una red neuronal está compuesta por un conjunto de nodos conectados, denominados neuronas debido a que las conexiones entre ellos recuerdan a las sinapsis de las neuronas biológicas. Las neuronas reciben señales de otras neuronas como entradas, las procesan y generan una señal de salida que se transmite a otras neuronas. Estas señales son valores reales.

La salida de una neurona se calcula como una función no lineal de una combinación lineal de sus entradas. Además, es habitual añadir un parámetro de sesgo o *bias* que se suma a dicha combinación lineal.

$$y = f(\sum_j w_j x_j + b)$$

donde y es la salida de la neurona, x_j denota la componente j-ésima del vector de entradas de la neurona, f es una función real, w_j denota la componente j-ésima del vector de coeficientes de la combinación lineal y b es el parámetro de sesgo. La función f se conoce como función de activación y los coeficientes de la combinación lineal, w_j, se conocen como pesos. La magnitud de cada uno de estos pesos se puede interpretar como una medida de la fuerza de la correspondiente conexión entre dos neuronas. La inclusión del término de sesgo es equivalente a incorporar una entrada con valor constante 1 y asociarle un peso [12].

Habitualmente, las neuronas se estructuran en capas, de manera que las señales se transmiten desde una capa de entrada hasta una capa de salida, pasando por un número de capas intermedias, también llamadas capas ocultas. El número de capas de una red neuronal se conoce habitualmente como profundidad de la red.

2.1 Función de pérdida

El concepto de aprendizaje en las redes neuronales se fundamenta en la idea intuitiva de que, cuando la red responde de la forma deseada ante una cierta entrada, los parámetros, es decir, los pesos y los sesgos de las neuronas, deberían ajustarse para aumentar la probabilidad de una respuesta similar ante entradas similares en el futuro. Por el contrario, si la red responde de una forma no deseada, los parámetros deberían ajustarse para disminuir la probabilidad de una respuesta similar [12].

Para implementar esta idea, es necesario definir una medida del error que comete la red, que en el contexto del aprendizaje automático se conoce como función de pérdida y se denota por $\mathcal{L}$. Esta función debe cumplir dos condiciones: debe poderse expresar como un promedio de n funciones de error, correspondientes a cada una de las observaciones del conjunto de datos, y debe poderse expresar como una función de las salidas de la red [13].

El proceso de optimización de los parámetros de una red neuronal se conoce como entrenamiento de la red y su objetivo es encontrar el conjunto de parámetros que minimice la función de pérdida $\mathcal{L}$ sobre un cierto conjunto de datos, que se denomina conjunto de entrenamiento. Idealmente, con esos mismos parámetros se conseguirá también un error pequeño sobre nuevos datos que tengan una distribución similar a la de los datos de entrenamiento.

2.2 Descenso del gradiente

El entrenamiento de una red neuronal habitualmente se realiza empleando un método de optimización iterativo conocido como descenso del gradiente, que se utiliza para realizar una búsqueda en el espacio de parámetros de la red. La idea básica del descenso del gradiente es actualizar los parámetros en la dirección de máximo descenso local de la función $\mathcal{L}$. En cada iteración, se calcula el gradiente de la función de pérdida respecto a los parámetros, y estos incrementan en la dirección opuesta al gradiente.

$$\Delta w = -\eta \frac{\partial \mathcal{L}}{\partial w}$$

donde η es un hiperparámetro denominado tasa de aprendizaje, que controla la magnitud del cambio de los parámetros en cada iteración.

El descenso del gradiente clásico emplea todas las observaciones del conjunto de entrenamiento para calcular el gradiente de la función de pérdida en cada iteración. Cuando el conjunto de entrenamiento es muy grande y la red es compleja, esto puede suponer un coste computacional excesivo. Una alternativa es emplear el método conocido como descenso del gradiente estocástico, que consiste en aproximar el gradiente de la función de pérdida por el gradiente calculado con una sola instancia elegida aleatoriamente en cada iteración.

Habitualmente, se emplea el descenso del gradiente por lotes como solución de compromiso entre ambos métodos. En cada iteración, se selecciona aleatoriamente un conjunto de instancias, de tamaño prefijado, para calcular el gradiente. Cada uno de estos conjuntos se conoce como lote. El entrenamiento de la red se estructura por épocas, de manera que en cada época se realiza una pasada por todas las instancias del conjunto de entrenamiento, sin repetir ninguna. Para ello, al comienzo de cada época se hace una partición aleatoria del conjunto de entrenamiento, asignando cada instancia a un lote. Si el tamaño del conjunto de entrenamiento no es múltiplo del tamaño de lote, hay un lote de menor tamaño formado por las instancias restantes.

2.3 Perceptrón multicapa

El perceptrón multicapa (MLP, del inglés *mutilayer perceptron*) es una arquitectura de red utilizada en numerosas aplicaciones por su simplicidad y su amplia capacidad de representación. Se trata de una red completamente conexa, lo que significa que cada neurona está conectada con todas las neuronas de la siguiente capa (ver Figura 1).

Se puede demostrar que un MLP con una sola capa oculta es un aproximador universal de funciones, es decir, puede aproximar cualquier función continua con una precisión arbitraria si no hay restricción sobre el número de neuronas [14]. Si el número de neuronas de la capa oculta es menor que el número de grados de libertad del conjunto de datos de entrenamiento, la salida de la capa oculta se puede interpretar como una representación en componentes principales no lineal del espacio de entradas [12]. De esta manera, el MLP es capaz de filtrar el ruido presente en los datos de entrenamiento y modelizar la estructura del sistema generador subyacente. Esto es lo que proporciona la capacidad de generalización sobre nuevos datos.

Sin embargo, a menudo el aprendizaje de un concepto complejo se ve facilitado por el aumento

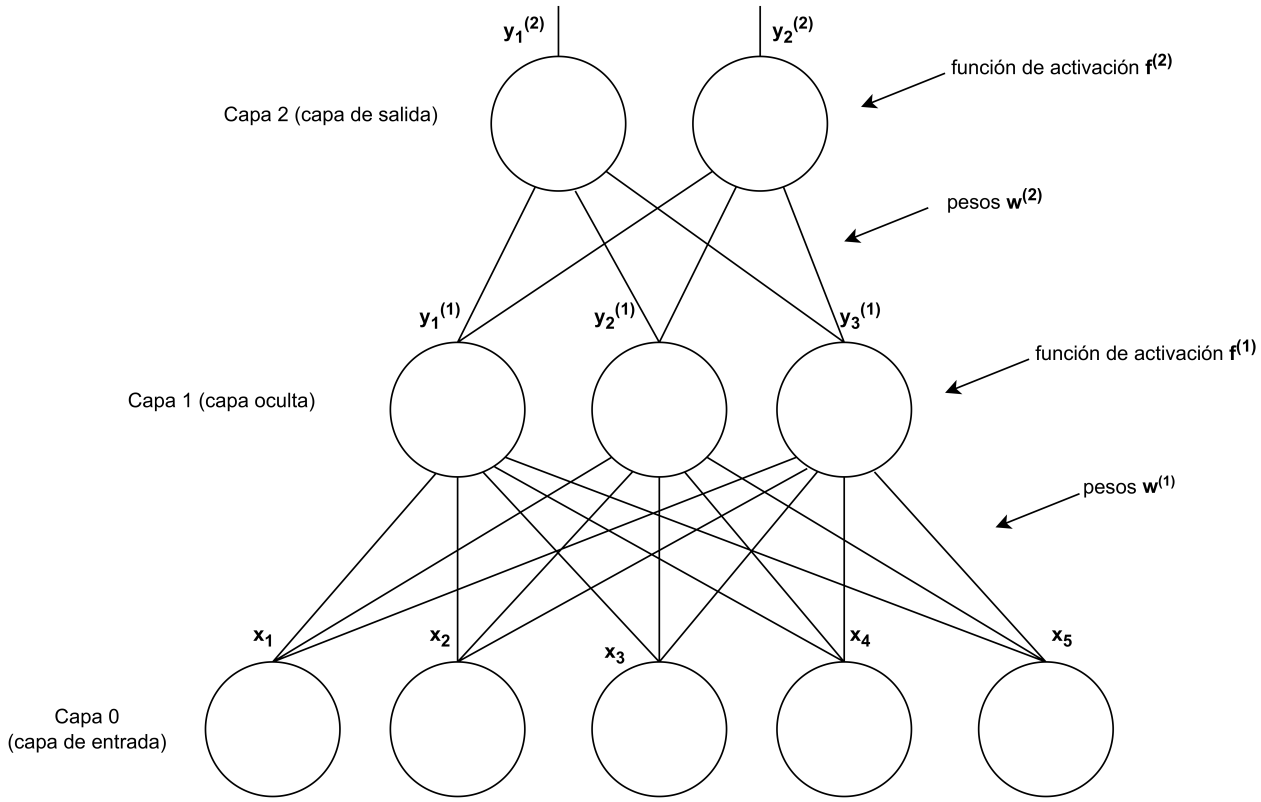

Figura 1: Esquema de un MLP con una capa oculta.

del número de capas del MLP. Las capas sucesivas producen representaciones con niveles de abstracción progresivamente mayores.

2.4 Funciones de activación

Un aspecto a tener en cuenta en el diseño de las redes neuronales es la elección de la función de activación de las capas ocultas. Tradicionalmente se han empleado funciones de la familia sigmoide, debido a la simplicidad computacional de su derivada.

Definición 2.1. Función sigmoide. Una función sigmoide es una función real, acotada y diferenciable, que está definida para cualquier valor real, y que tiene derivada no negativa en todos sus puntos.

Algunos ejemplos de funciones sigmoides son (ver Figura 2):

- Función sigmoide logística. Se define como $f(x) = \frac{1}{1+exp(-x)}$ y su rango es el intervalo $(0, 1)$.
- Función tangente hiperbólica. Se define como $f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ y su rango es el intervalo $(-1, 1)$.
- Función arcotangente $f(x) = arctan(x)$. Su rango es el intervalo $(-\frac{\pi}{2}, \frac{\pi}{2})$.

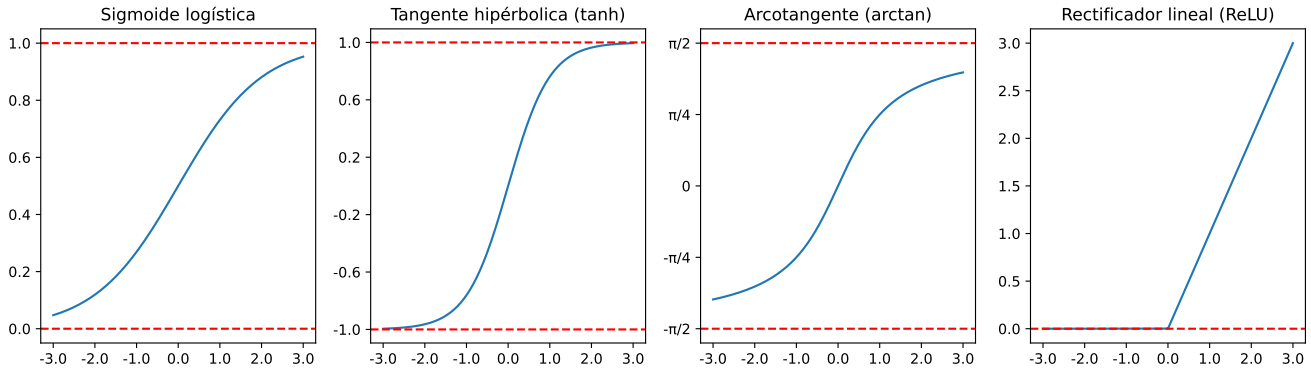

Figura 2: Representación gráfica de varias funciones de activación.

No obstante, en la última década se ha observado que las redes neuronales profundas, es decir, con un gran número de capas ocultas, funcionan mejor con la función de activación ReLU, descrita en [15]. La función ReLU (*rectifier linear unit*) o rectificador, definida como $f(x) = max(0, x)$, es una mejor aproximación a la función de activación de las neuronas biológicas que las funciones sigmoides, puesto que induce una representación dispersa de la información. Esto significa que las representaciones latentes de las entradas que se construyen en cada capa de la red tienen un gran número de componentes que valen 0. La función ReLU vale 0 para toda entrada negativa, por lo que, tras una inicialización aleatoria de los pesos de la red, aproximadamente la mitad de las neuronas están inactivas, es decir, tienen salida 0.

Desde un punto de vista computacional, la representación dispersa de la información introducida por la función ReLU proporciona varias ventajas, que se describen con más detalle en [15].

En primer lugar, facilita el desacoplamiento de las representaciones latentes generadas por la red para las diferentes instancias del conjunto de datos. En una representación no dispersa, casi todos los parámetros de la red intervienen en el cálculo de la salida para cualquier entrada y, por tanto, si se calcula la pérdida asociada a una entrada cualquiera y se aplica el descenso del gradiente se actualizarán casi todos los parámetros. En cambio, en una representación dispersa, para cada entrada hay un subconjunto de neuronas activas, es decir, que tienen salida distinta de 0, y en el resto de neuronas el gradiente vale 0. Por tanto, el subconjunto de parámetros que se actualizan cuando llega una determinada entrada puede ser disjunto del subconjunto de parámetros que se actualiza cuando llega otra entrada diferente.

Además, es frecuente que la cantidad de información de los datos sea variable, porque algunos valores de las entradas son mucho más frecuentes que otros. Por ello, es más eficiente representar esta información en una estructura de datos de tamaño variable que en una de tamaño fijo. La variación en el número de neuronas activas para cada entrada permite a la red controlar el tamaño de la representación, de forma que el número de neuronas que se activan para las entradas más frecuentes tenderá a ser menor que para las entradas menos frecuentes.

Por último, hay que tener en cuenta que el coste computacional asociado a la función ReLU es menor que el de otras funciones de activación, ya que para cada entrada, solo un subconjunto de las neuronas tienen salida distinta de 0, y en este subconjunto la salida es una función lineal de la entrada.

2.5 Propagación hacia atrás

Para calcular el gradiente en un MLP, se utiliza el algoritmo de propagación hacia atrás o *backpropagation*. Este algoritmo consiste en calcular el gradiente de la función de pérdida en cada capa a partir del valor del gradiente en la capa posterior. De esta manera, el cálculo del gradiente comienza en la última capa y se propaga hacia las capas anteriores.

Para simplificar la notación, en esta sección se utiliza w para referirse al conjunto de parámetros de la red, incluyendo pesos y sesgos. Sea $w^{(l)}$ la matriz de parámetros de las neuronas de la capa l, $f^{(l)}$ la función de activación de dicha capa y $z^{(l)} = w^{(l)}y^{(l-1)}$ la entrada de la capa l multiplicada por la matriz de parámetros. La salida de cada capa se puede expresar como $y^{(l)} = f^{(l)}(z^{(l)})$. Se define el error en la neurona k de la capa l como:

$$\delta_k^{(l)} = \frac{\partial \mathcal{L}}{\partial z_k^{(l)}} = \frac{\partial \mathcal{L}}{\partial y_k^{(l)}} \frac{\partial f^{(l)}(z_k^{(l)})}{\partial z_k^{(l)}}$$

En particular, en la última capa (L), los errores $\delta_k^{(L)}$ se calculan directamente. En el resto de capas, el cálculo de los errores se realiza a partir de los errores de la capa inmediatamente posterior:

$$\delta_k^{(l)} = \sum_j (w_{jk}^{(l+1)} \delta_j^{(l+1)}) \frac{\partial f^{(l)}(z_k^{(l)})}{\partial z_k^{(l)}}, l = 1, ..., L-1$$

donde $w_{jk}^{(l+1)}$ denota el peso de la conexión entre la neurona k de la capa l y la neurona j de la capa $l+1$.

De forma que el gradiente buscado se calcula como:

$$\frac{\partial \mathcal{L}}{\partial w_{jk}^{(l)}} = y_k^{(l-1)} \delta_j^{(l)}, l = 1, ..., L-1$$

teniendo en cuenta que en la primera capa la expresión es:

$$\frac{\partial \mathcal{L}}{\partial w_{jk}^{(l)}} = x_k \delta_j^{(l)}$$

Para una descripción más detallada, el lector puede consultar [13].

2.6 Optimizadores

La tasa de aprendizaje es uno de los hiperparámetros con mayor influencia en el rendimiento de una red neuronal [16]. En la explicación anterior sobre el descenso del gradiente se ha considerado la situación simplificada de tener una tasa de aprendizaje constante e igual para todos los parámetros. Sin embargo, como la magnitud del gradiente puede ser muy diferente para los diferentes parámetros, es más adecuado emplear una tasa de aprendizaje adaptativa, que toma un valor diferente para cada parámetro de la red [17].

Los parámetros asociados con características que ocurren de manera infrecuente en los datos tienen gradientes pequeños en la mayoría de las iteraciones y solo reciben actualizaciones significativas cuando entra una instancia que presenta esa característica [18]. Si se utiliza una única tasa de aprendizaje global, estos parámetros convergerán más lentamente que los que están asociados a características más comunes [18]. Por tanto, los parámetros que reciben menos actualizaciones deberían tener tasas de aprendizaje más grandes para acelerar la convergencia [19]. La tasa de aprendizaje de cada parámetro se obtiene multiplicando la tasa de aprendizaje global por un valor asociado a la magnitud del gradiente para cada parámetro.

Existen múltiples algoritmos de optimización que implementan esta idea de diferentes formas. Uno de los más populares actualmente es el optimizador Adam (*adaptive moment estimation*). Este algoritmo utiliza una estimación del primer momento del gradiente, m_{it}, y del segundo momento, q_{it}, en cada iteración it.

$$m_{it} = \mathbb{E}_{it}[g]$$
$$q_{it} = \mathbb{E}_{it}[g^2]$$

donde $g = \frac{\partial \mathcal{L}}{\partial w}$ es el gradiente de la función de pérdida con respecto a los parámetros de la red.

Las estimaciones del primer y del segundo momento se inicializan a 0 y en cada iteración se actualizan empleando medias móviles.

$$m_{it} = \beta_1 m_{it-1} + (1 - \beta_1) g_{it}$$
$$q_{it} = \beta_2 q_{it-1} + (1 - \beta_2) g_{it}^2$$

donde β_1 y β_2 son hiperparámetros con valores entre 0 y 1.

En la actualización de los parámetros se emplean las estimaciones corregidas $\hat{m_{it}}$ y $\hat{q_{it}}$ para corregir el sesgo provocado por la inicialización a 0. Estas estimaciones corregidas se calculan como:

$$\hat{m_{it}} = m_{it}/(1 - \beta_1^{it})$$
$$\hat{q_{it}} = q_{it}/(1 - \beta_2^{it})$$

donde β_1^{it} y β_2^{it} denotan β_1 y β_2 elevados a la potencia it.

Finalmente, la regla de actualización de los parámetros es:

$$w_{it} = w_{it-1} - \eta \hat{m_{it}}/\sqrt{\hat{q_{it}}}$$

El lector puede consultar [20] para una lectura más detallada.

3 Aprendizaje por refuerzo

El aprendizaje por refuerzo es un área del aprendizaje automático que estudia los problemas en los que un agente artificial debe aprender a realizar una tarea mediante la interacción con un entorno dinámico (E). Es un enfoque que se fundamenta en la idea del aprendizaje mediante prueba y error [21]. El agente es capaz de percibir parcialmente el estado (s) del entorno y utiliza esa información para decidir las acciones (a) que va a realizar [21]. Las acciones del agente modifican el entorno. Tras realizar una acción, el agente recibe una señal de recompensa (r) que le informa del valor del cambio que se acaba de producir en el entorno [21]. El agente debe generalizar sus experiencias para aprender un comportamiento que le permita obtener mayores recompensas en el futuro [22].

La principal ventaja del aprendizaje por refuerzo es que permite al agente aprender comportamientos complejos que resultarían difíciles de programar de manera explícita. El aprendizaje por refuerzo se ha aplicado con éxito a diversos tipos de problemas, entre los que destacan los siguientes:

- Robótica. Algunos ejemplos son el control de robots manipuladores de objetos [23] o el control de la locomoción de un robot cuadrúpedo [24].
- Conducción autónoma. El aprendizaje por refuerzo se ha utilizado para obtener algoritmos de control para vehículos autónomos en entornos simulados [25] [26].
- Videojuegos de estrategia. En 2019, un conjunto de agentes entrenados mediante aprendizaje por refuerzo se convirtió en el primer sistema de IA capaz de ganar a jugadores profesionales en el videojuego Dota 2 [27].

A continuación se describen varios conceptos fundamentales en el contexto del aprendizaje por refuerzo y de interés para nuestro trabajo.

3.1 Proceso de decisión de Markov

El esquema básico del entrenamiento de un agente de aprendizaje por refuerzo se describe formalmente de la siguiente forma. En cada instante de tiempo discreto t, el agente recibe como entrada una observación, s_t, del estado actual del entorno E y selecciona una acción, a_t, del conjunto de acciones posibles. La acción modifica el estado interno del entorno y el agente recibe la observación del nuevo estado s_{t+1} y una función de recompensa $r_{t+1} = f(s_{t+1}, a_t)$. La función de recompensa es un número real cuyo valor es mayor cuanto mayor es la valoración de la transición entre estados. En la Figura 3 se muestra un esquema de la interacción entre el agente y el entorno en un instante de tiempo t.

En general, el entorno puede ser estocástico, lo que supone que escoger la misma acción en el mismo estado en dos ocasiones diferentes puede resultar en transiciones a diferentes estados. Sin embargo, se asume que el entorno es estacionario, es decir, que las probabilidades de estas transiciones entre estados no cambian con el tiempo.

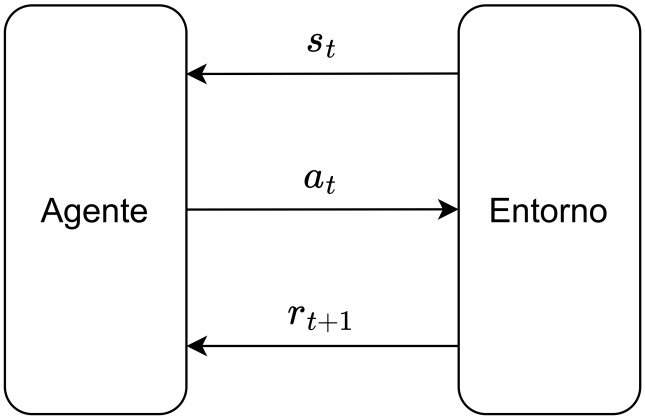

Figura 3: Esquema de la interacción del agente con el entorno en un instante de tiempo t.

Por tanto, se puede formular el problema como un proceso de decisión de Markov en tiempo discreto con un espacio de estados $\mathcal{S}$, un espacio de acciones $\mathcal{A}$, una distribución del estado inicial con densidad $p_1(s_1)$, una distribución de transiciones estacionaria con densidad condicional $p(s_{t+1}|s_t, a_t)$ y una función de recompensa $f : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ [10]. En general, tanto $\mathcal{S}$ como $\mathcal{A}$ pueden ser discretos o continuos. En el problema que se aborda en este trabajo, ambos espacios son continuos.

La distribución de las transiciones entre estados satisface la propiedad de Markov

$$p(s_{t+1}|s_t, a_t) = p(s_{t+1}|s_1, a_1, ..., s_t, a_t)$$

para cualquier secuencia de estados y acciones [10]. Esto significa que el nuevo estado del entorno solo depende del estado actual y de la última acción del agente, y no es necesario tener en cuenta lo que ha ocurrido anteriormente [21].

3.2 Políticas

La política concreta el comportamiento del agente y puede ser determinista o estocástica [21]. Una política determinista se puede expresar como una función del espacio de estados en el espacio de acciones, $\mathcal{S} \to \mathcal{A}$. Es decir, siempre que el agente observe el mismo estado, responderá con la misma acción. En cambio, una política estocástica relaciona estados observados con distribuciones de probablidad sobre el espacio de acciones, $\mathcal{S} \to P(\mathcal{A})$. Es decir, el agente puede escoger acciones distintas en respuesta al mismo estado.

Las políticas estocásticas son útiles durante el entrenamiento del agente, ya que permiten la exploración de nuevas regiones del espacio de acciones. En cambio, cuando el entrenamiento del agente ha concluido y va a ser aplicado en un entorno real, lo más razonable es utilizar una política determinista para minimizar la influencia del azar en el desempeño del agente. Para convertir una política estocástica en determinista, basta con escoger como acción la media de las distribuciones de probabilidad generadas.

3.2.1 Medida de rendimiento

El objetivo del agente es encontrar una política óptima que escoja aquellas acciones que maximicen el valor esperado de una cierta medida de rendimiento R_t. Habitualmente, la medida de rendimiento utilizada es la suma de recompensas futuras con descuento [10].

$$R_t = \sum_{i=t}^{\infty} \gamma^{i-t} r_{i+1}$$

donde γ es un factor de descuento entre 0 y 1. Cuanto mayor sea el valor de γ, mayor es la influencia de las recompensas futuras en la medida de rendimiento. Este modelo de optimalidad prioriza las recompensas a corto plazo mediante la aplicación de un descuento geométrico a las recompensas obtenidas en el futuro.

3.2.2 Funciones de valor

En este contexto resulta útil conocer el valor de un estado o de un par de estado-acción. El término valor se refiere al rendimiento esperado si se comienza en ese estado o par estado-acción, y luego se actúa de acuerdo con una política específica. Las funciones de valor son empleadas por la mayoría de los algoritmos de aprendizaje por refuerzo.

A continuación se definen las principales funciones de interés para nuestro trabajo, siguiendo la notación presentada en [21]:

Definición 3.1. Función estado-valor $V^\pi(s)$. El valor de un estado para una política π en un entorno E es el rendimiento esperado de un agente que comienza en el estado s y siempre actúa de acuerdo con la política π:

$$V^\pi(s) = \mathbb{E}_{\pi,E}[R_t | s_t = s]$$

Definición 3.2. Función acción-valor $Q^\pi(s,a)$. El valor de un par estado-acción para una política π en un entorno E es el rendimiento esperado de un agente que realiza la acción a en el estado s y a continuación actúa con la política π:

$$Q^\pi(s,a) = \mathbb{E}_{\pi,E}[R_t | s_t = s, a_t = a]$$

La diferencia con la función anterior es que la acción a puede ser diferente de la que marca la política π.

Definición 3.3. Función ventaja $A^\pi(s,a)$. La ventaja de un par estado-acción para una política π es la diferencia entre la función acción-valor y la función estado-valor:

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$$

Esta función representa la ventaja de seleccionar la acción a en el estado s frente a seleccionar la acción dada por la política π en ese estado, suponiendo que después el agente actúa siempre de acuerdo con la política π. Es decir, es una función que describe la ventaja que obtendría un agente que seleccione la acción a en el estado s, y a continuación siga la política π, con respecto a otro agente que actúe estrictamente de acuerdo con la política π. Como es lógico, una ventaja positiva indica que la acción a es mejor que la acción dada por la política π en el estado s, y una ventaja negativa indica que la acción a es peor.

3.2.3 Ecuaciones de Bellman

Las funciones de valor cumplen unas identidades recursivas conocidas como ecuaciones de Bellman, cuya descripción se puede consultar en [21]. Estas ecuaciones relacionan el valor de un estado s_t con el del siguiente estado, s_{t+1}. Se fundamentan en el hecho de que el rendimiento a partir de un instante t se descompone en la suma de la recompensa recibida en la siguiente transición entre estados más el rendimiento a partir del instante $t+1$.

$$R_t = r_{t+1} + R_{t+1}$$

Las ecuaciones de Bellman son relevantes porque permiten evaluar la consistencia de los estimadores de las funciones de valor, como se explica más adelante en uno de los algoritmos estudiados.

La ecuación de Bellman para la función estado-valor es:

$$V^\pi(s_t) = \mathbb{E}_{\pi,E}[r_{t+1} + \gamma V^\pi(s_{t+1})]$$

Esta ecuación refleja que el valor de un estado es la suma de la recompensa esperada de la siguiente transición entre estados más el valor esperado (con descuento γ) del nuevo estado alcanzado.

La ecuación de Bellman para la función acción-valor relaciona el valor de un par estado-acción, (s_t, a_t), con el valor del siguiente par estado-acción, (s_{t+1}, a_{t+1}).

$$Q^\pi(s_t, a_t) = \mathbb{E}_E[r_{t+1} + \gamma \mathbb{E}_\pi[Q^\pi(s_{t+1}, a_{t+1})]]$$

Si la política π es determinista, las acciones a_t se pueden describir como una función $a_t = \mu(s_t)$. En este caso, la ecuación de Bellman para la función acción-valor queda de la forma [28]:

$$Q^\mu(s_t, a_t) = \mathbb{E}_E[r_{t+1} + \gamma Q^\mu(s_{t+1}, \mu(s_{t+1}))]$$

3.3 Arquitectura actor-crítico

Existen diversos enfoques para optimizar la política del agente. El paradigma más habitual en los algoritmos de aprendizaje por refuerzo para espacios de acciones continuos es el uso de la arquitectura actor-crítico [29], que se introduce a continuación.

Dado el gran éxito de las redes neuronales como aproximadores de funciones no lineales, en los problemas con espacios de estados y de acciones continuos la política que sigue el agente se implementa con una red neuronal, denominada actor. Como su objetivo es maximizar el rendimiento esperado, para optimizar el actor es preciso definir una función de pérdida cuyo gradiente sea una buena aproximación del gradiente del rendimiento de la política. Este no es un problema trivial, ya que hay que tener en cuenta que algunos estados son más propensos a transicionar a estados favorables que otros, con independencia de la acción seleccionada. Es decir, el agente puede recibir una recompensa grande en un determinado instante a pesar de haber seleccionado la peor acción posible, y de la misma forma, puede recibir una recompensa pequeña a pesar de haber seleccionado la mejor acción posible.

Por este motivo, el cálculo de la función de pérdida del actor involucra alguna de las funciones de valor introducidas anteriormente, ya sea la función estado-valor, V^π, o la función acción-valor, Q^π.

El uso de estas funciones proporciona un punto de referencia para evaluar si la acción seleccionada por el actor en un determinado estado ha mejorado o empeorado el valor esperado para dicho estado. Para aproximar la función V^π o Q^π, se introduce el crítico, que se implementa también con una red neuronal [10].

El actor y el crítico se optimizan simultáneamente empleando los algoritmos de optimización habituales basados en descenso del gradiente por lotes [29]. En la siguiente sección se describe cómo se concretan estas ideas en cada uno de los algoritmos estudiados.

3.4 Algoritmos

En este trabajo se han estudiado dos algoritmos de aprendizaje por refuerzo que están entre los más populares para espacios continuos.

3.4.1 *Deep Deterministic Policy Gradient*

El primer algoritmo estudiado en este trabajo se conoce como *Deep Deterministic Policy Gradient* (DDPG) y fue propuesto por investigadores de Google DeepMind en 2016 [28]. Este algoritmo hace uso de la arquitectura actor-crítico descrita en la sección anterior.

Arquitectura El actor es una red neuronal que implementa una política determinista μ. Por tanto, recibe como entrada un estado y devuelve una acción. El actor se denota como μ_θ, donde θ es el conjunto de parámetros de la red.

El crítico es una red neuronal que aproxima la función acción-valor asociada a dicha política. Por tanto, recibe como entradas un estado y una acción, y devuelve un número real que representa el valor de ese par estado-acción [28]. Se utiliza la notación Q^μ para hacer referencia a la función acción-valor teórica asociada a la política μ, que no se conoce, y Q_ω para representar el crítico que aproxima dicha función, donde ω es el conjunto de parámetros de la red.

Esquema de entrenamiento Un obstáculo que surge cuando se aplican redes neuronales a un problema de aprendizaje por refuerzo es que la mayoría de algoritmos de optimización requieren que las muestras que se emplean para entrenar la red sean independientes. Esto no se cumple en el aprendizaje por refuerzo, ya que las observaciones forman una secuencia temporal [28]. El algoritmo DDPG resuelve este inconveniente empleando un *buffer* denominado *buffer* de reproducción. En él se almacenan las experiencias del agente en forma de tuplas $(s_t, a_t, r_{t+1}, s_{t+1})$. Es decir, cada entrada del *buffer* almacena un estado del entorno, la acción realizada por el agente en ese estado, la recompensa obtenida y el siguiente estado observado. El entrenamiento del modelo se realiza a partir de experiencias extraídas aleatoriamente del *buffer*. Así se eliminan las correlaciones entre observaciones sucesivas, y además, cada experiencia puede usarse en muchas iteraciones de aprendizaje, lo que supone un uso más eficiente de los datos [22].

Cada iteración de entrenamiento se compone de dos pasos. En primer lugar, el agente genera una nueva transición entre estados, que es añadida al *buffer*. Cuando el *buffer* se llena, se descartan

las experiencias más antiguas. En segundo lugar, se realiza una actualización de los parámetros del actor y el crítico, calculando sus respectivas funciones de pérdida a partir de un lote de experiencias extraído aleatoriamente del buffer [28].

Una estrategia habitual para acelerar la exploración al principio del entrenamiento es generar una secuencia de transiciones seleccionando las acciones de una distribución uniforme sobre el espacio de acciones, antes de empezar a ejecutar la política. Así se introduce en el *buffer* un conjunto inicial de experiencias con suficiente variedad para comenzar el entrenamiento.

Además, en algunos problemas es útil generar las transiciones empleando una política estocástica para aumentar la diversidad de las acciones exploradas [10]. La política estocástica se construye añadiendo ruido a las acciones seleccionadas por el actor, que puede ser Gaussiano o de cualquier otro tipo.

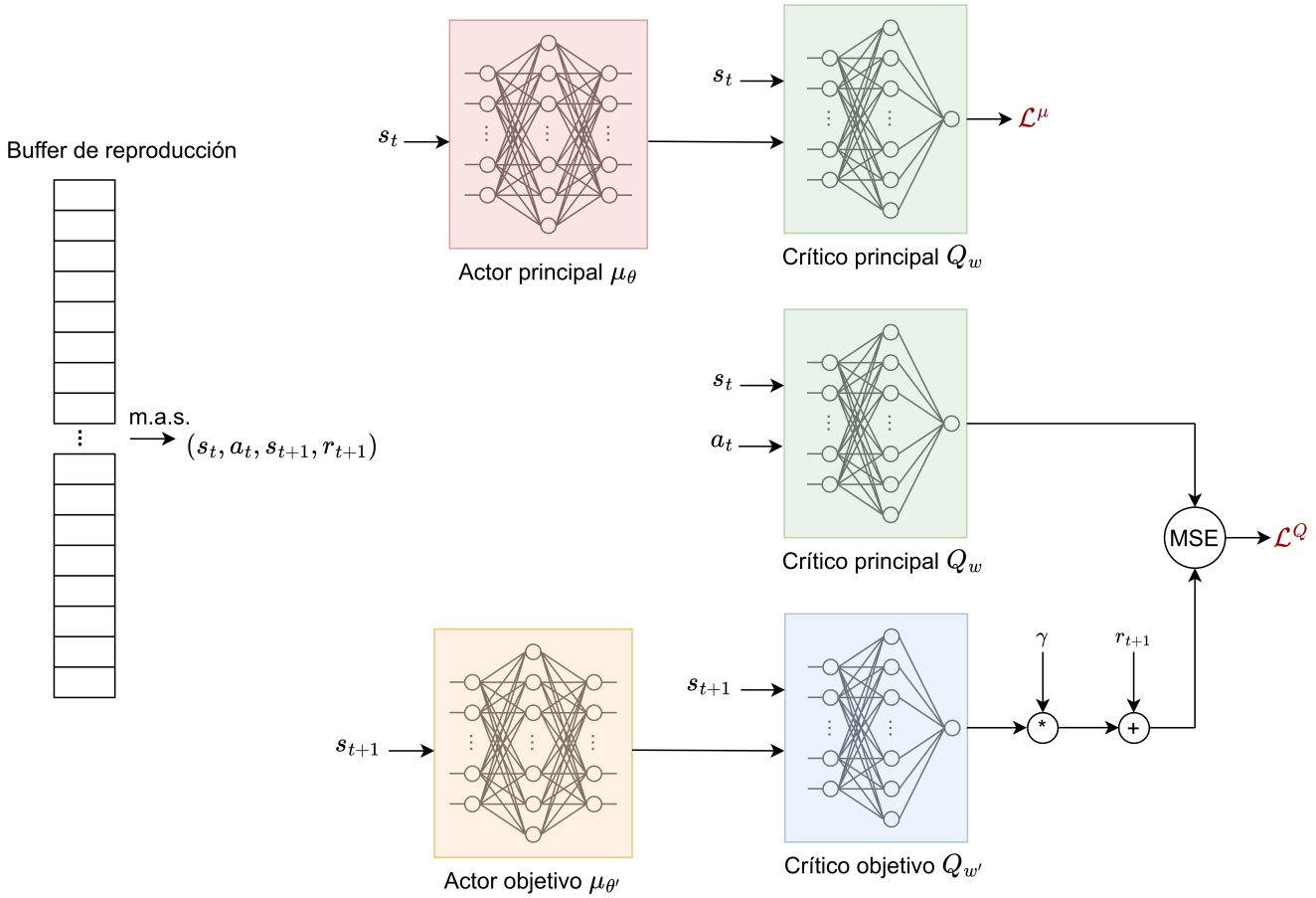

Figura 4: Cálculo de las pérdidas del actor y el crítico durante el entrenamiento en el algoritmo DDPG.

Funciones de pérdida A continuación se describe el procedimiento de optimización del actor y el crítico (ver Figura 4). El propósito del actor es proporcionar una acción a, para cada estado s, tal que se maximice el valor esperado $Q^\mu(s, a)$. Así, la función de pérdida del actor se calcula a

partir de las estimaciones proporcionadas por el crítico [21].

$$\mathcal{L}^{\mu} = -\frac{1}{|D|} \sum_{s_t \in D} Q_{\omega}(s_t, \mu_{\theta}(s_t))$$

donde D es un lote de experiencias extraído del *buffer* de reproducción.

La optimización del crítico se fundamenta en el uso de la ecuación de Bellman, descrita anteriormente. Como dicha ecuación se cumple para cualquier par estado-acción, con independencia de cómo se haya seleccionado dicha acción, se puede estimar la función Q^{μ} utilizando experiencias generadas por una política diferente de μ. En consecuencia, el *buffer* de reproducción puede contener experiencias antiguas, que se hayan obtenido con una política "obsoleta" de una etapa anterior del entrenamiento, y aun así, estas experiencias se pueden usar en el entrenamiento del crítico.

Sustituyendo en la ecuación de Bellman los valores reales $Q^{\mu}(s_t, a_t)$ por los valores estimados por el crítico, $Q_{\omega}(s_t, a_t)$, se obtiene un criterio para evaluar el desempeño del crítico. El crítico debe minimizar las diferencias entre los valores predichos $Q_{\omega}(s_t, a_t)$ y los valores objetivo dados por la ecuación de Bellman, $r_{t+1} + \gamma Q^{\mu}(s_{t+1}, \mu_{\theta}(s_{t+1}))$. Así, se define una función de pérdida empleando el error cuadrático medio [28].

$$\mathcal{L}^{Q} = \frac{1}{|D|} \sum_{(s_t, a_t, r_{t+1}, s_{t+1}) \in D} (Q_{\omega}(s_t, a_t) - r_{t+1} - \gamma Q^{\mu}(s_{t+1}, \mu(s_{t+1})))^2$$

La dificultad de este enfoque radica en la estimación de los valores objetivo. Si los valores objetivo se estimaran utilizando el mismo crítico, tendrían una fuerte correlación con los valores predichos $Q_w\omega(s_t, a_t)$, por depender de los mismos parámetros. Esta situación es indeseable porque introduce inestabilidad en el aprendizaje [22]. Para evitarlo, es necesario que la estimación de los valores objetivo sea independiente de los valores predichos por el crítico.

La solución propuesta en el algoritmo DDPG consiste en crear una copia del crítico, denominada crítico objetivo [28]. Los parámetros del crítico objetivo, ω', deben aproximarse a los del crítico principal, pero sin llegar a coincidir. Esto se logra actualizando los parámetros del crítico objetivo en cada iteración con una ponderación entre sus valores actuales y los valores de los parámetros del crítico principal [28]. Además, con el mismo propósito de mejorar la estabilidad de los valores objetivo, se introduce una copia del actor denominada actor objetivo [28]. Los parámetros del actor objetivo, θ', se actualizan también en cada iteración con una ponderación entre sus valores actuales y los valores de los parámetros del actor principal [28].

$$\omega'_t = \tau\omega_t + (1-\tau)\omega'_{t-1}$$
$$\theta'_t = \tau\theta_t + (1-\tau)\theta'_{t-1}$$

siendo τ un hiperparámetro entre 0 y 1, normalmente cercano a 0 [28].

Entonces, los valores objetivo empleados en el entrenamiento del crítico principal se calculan utilizando las estimaciones dadas por el actor y el crítico objetivo, de manera que la función de pérdida del crítico es [28]:

$$\mathcal{L}^{Q} = \frac{1}{|D|} \sum_{(s_t, a_t, r_{t+1}, s_{t+1}) \in D} (Q_{\omega}(s_t, a_t) - r_{t+1} - \gamma Q_{\omega'}(s_{t+1}, \mu_{\theta'}(s_{t+1})))^2$$

3.4.2 *Proximal Policy Optimization*

El segundo algoritmo que se ha aplicado en este trabajo se conoce como *Proximal Policy Optimization* (PPO) y fue propuesto en 2017 por investigadores de OpenAI [11].

Arquitectura Este algoritmo también utiliza una arquitectura actor-crítico, pero en este caso el actor representa una política estocástica π. Por tanto, recibe como entrada un estado y devuelve una distribución de probabilidad sobre el espacio de acciones. Las distribuciones utilizadas son normales multidimensionales, de dimensión igual a la del espacio de acciones, con matriz de covarianzas diagonal [30]. El actor se denota como π_θ, donde θ es el conjunto de sus parámetros. Se compone de dos elementos: una red neuronal que recibe como entrada un estado y devuelve la media de la distribución, y un vector de parámetros adicional que especifica la desviación típica. Es decir, la desviación típica de la distribución es independiente del estado observado. Los parámetros de la desviación típica se optimizan conjuntamente con los parámetros de la red neuronal.

Por su parte, el crítico es otra red neuronal que estima la función estado-valor asociada a la política π, V^π [11]. El crítico, denotado como V_ω, donde ω representa el conjunto de parámetros de la red, recibe un estado como entrada y devuelve un número real que representa el valor de dicho estado.

Esquema de entrenamiento Durante el entrenamiento, las acciones del agente se obtienen extrayendo una muestra de la distribución de probabilidad proporcionada por el actor a partir del estado observado en cada instante, $\pi_\theta(s_t)$. Por su parte, el crítico se implementa con una red neuronal que recibe como entrada un estado y devuelve un número real que representa el valor de dicho estado.

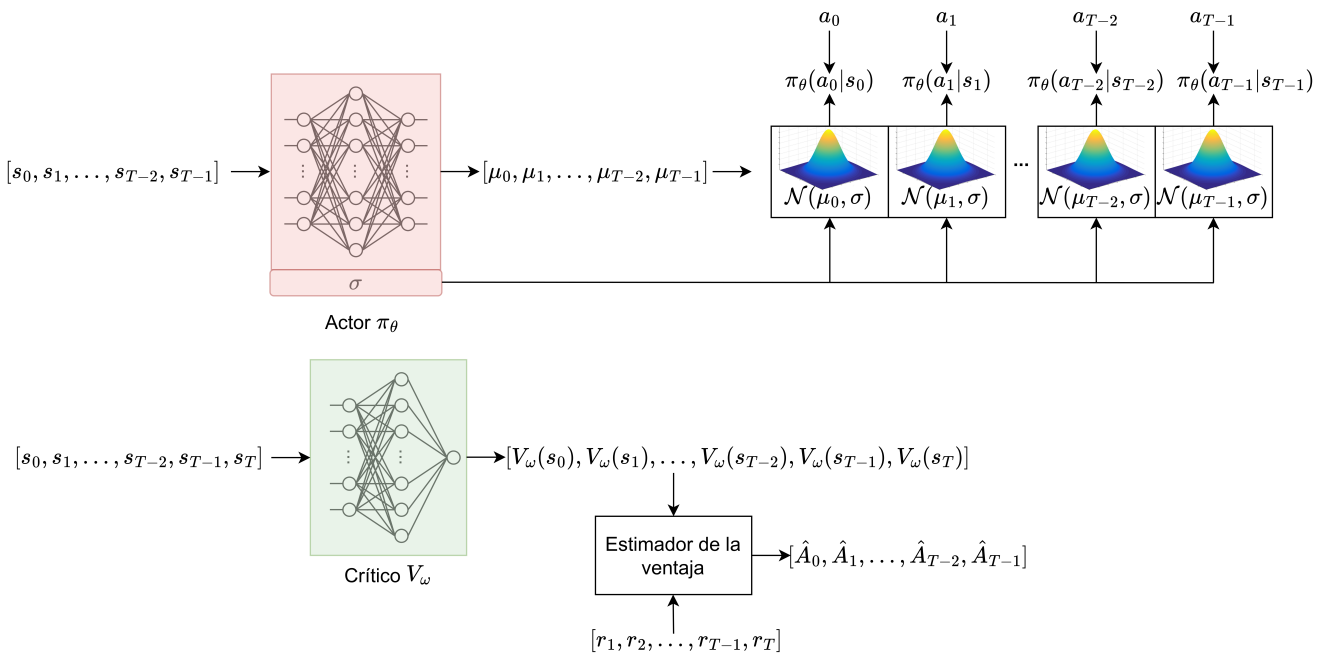

Figura 5: Estimaciones obtenidas a partir de una trayectoria en el algoritmo PPO al comienzo de una iteración de entrenamiento.

A diferencia del algoritmo DDPG, PPO no utiliza un *buffer* de reproducción. En su lugar, aplica el descenso del gradiente sobre experiencias generadas por la última versión de la política estocástica π_θ que se va a optimizar [21]. En cada iteración del bucle de entrenamiento, el agente ejecuta la política sobre el entorno durante T instantes de tiempo, generando una secuencia de longitud T de estados, acciones y recompensas, denominada trayectoria [11].

A continuación se calcula la probabilidad de cada acción a_t de la trayectoria en el correspondiente estado s_t bajo la política π, $\pi(a_t|s_t)$. Conviene aclarar que la notación $\pi(.|s)$ indica la distribución de probabilidad dada por la política estocástica π en el estado s. También se estima el valor de cada estado de la trayectoria, $V^\pi(s_t)$ y la ventaja de cada par estado-acción $A^\pi(s_t, a_t)$ (ver Figura 5). Para simplificar la notación en esta sección, las estimaciones $\hat{A}^\pi(s_t, a_t)$ se denotan como $\hat{A}_t$.

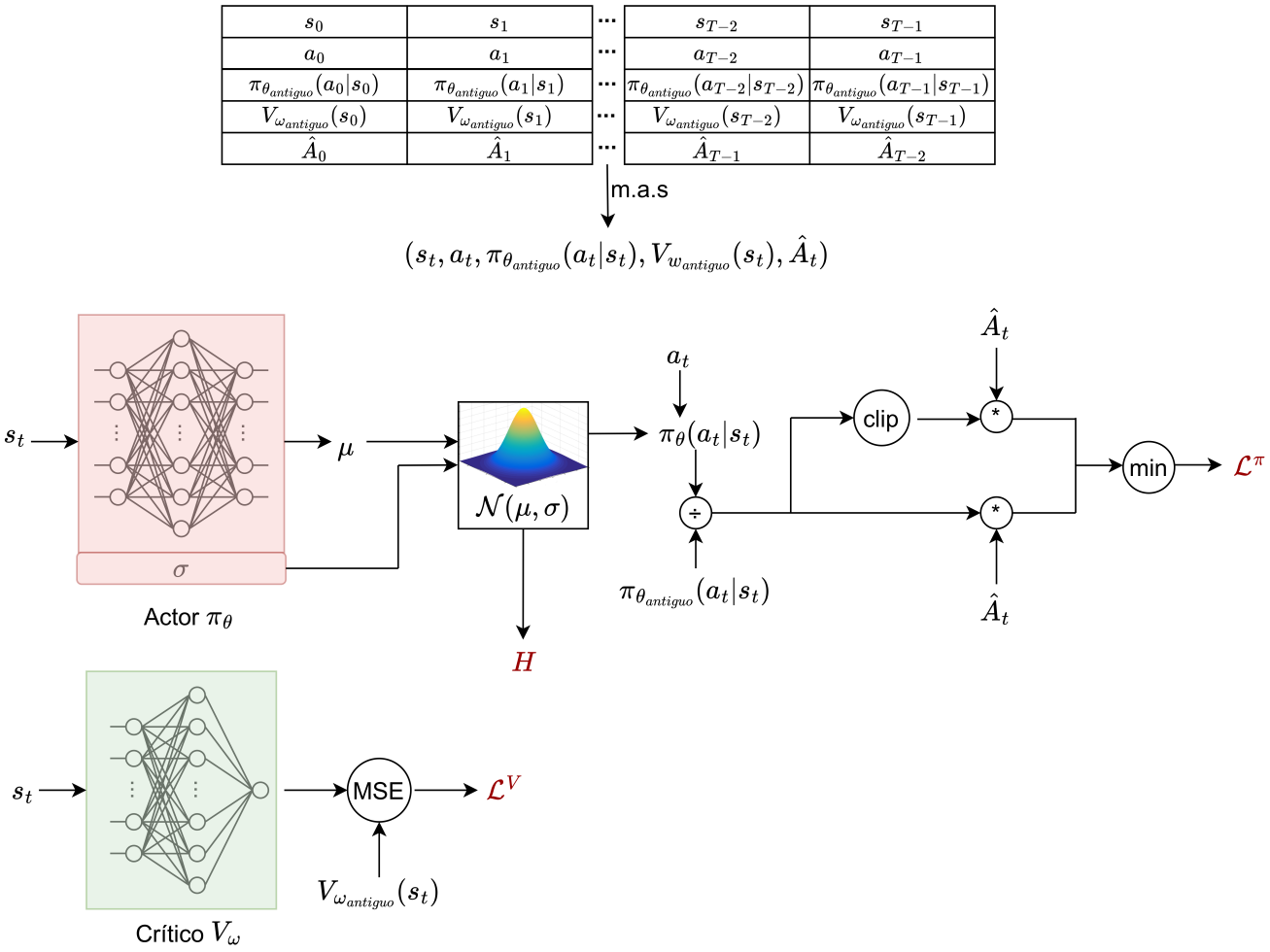

Figura 6: Cálculo de las pérdidas del actor y el crítico durante el entrenamiento en el algoritmo PPO.

Las estimaciones obtenidas, junto con las experiencias de la trayectoria, se utilizan para optimizar el actor y el crítico aplicando descenso de gradiente por lotes durante ζ épocas [11] (ver Figura 6). Hay que tener en cuenta que el orden de las experiencias de la trayectoria se aleatoriza antes de realizar la partición en lotes, para evitar el problema ya mencionado de las correlaciones entre observaciones consecutivas. Después de las ζ épocas la trayectoria se descarta y se recopila una nueva con la política actualizada.

Ventaja subrogada El actor se optimiza minimizando una función de pérdida compuesta por dos términos. El primer término, $\mathcal{L}^A$, aproxima el gradiente del rendimiento de la política mediante una estimación de la ventaja subrogada, una medida del rendimiento relativo de una política con respecto a otra.

Definición 3.4. Ventaja subrogada $A(\pi_0, \pi_1)$. La ventaja subrogada de una política estocástica π_1 con respecto a otra política estocástica π_0 es el valor esperado de la ventaja de un par estado-acción para la política π_0, multiplicada por el cociente de las densidades de probabilidad definidas por las políticas π_1 y π_0 para ese par estado-acción [11].

$$A(\pi_0, \pi_1) = \mathbb{E}_{\pi_0, E}[\frac{\pi_1(a_t|s_t)}{\pi_0(a_t|s_t)} A^{\pi_0}(s_t, a_t)]$$

El primer término de la función de pérdida del actor, $\mathcal{L}^A$, se define a partir de la ventaja subrogada de la política actual, π_θ, con respecto a la política con la que se ha generado la trayectoria, $\pi_{\theta_{antiguo}}$. Dada una estimación de la ventaja de cada par estado-acción de la trayectoria, $\hat{A}_t = \hat{A}^\pi(s_t, a_t)$, se puede calcular una estimación de la ventaja subrogada de la política actual con respecto a la política con la que se ha generado la trayectoria como:

$$\hat{A}(\pi_0, \pi_1) = \frac{1}{|D|} \sum_{t \in D} (\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{antiguo}}(a_t|s_t)} \hat{A}_t)$$

donde D es un conjunto de instantes de tiempo de la trayectoria seleccionado aleatoriamente.

Para evitar actualizaciones de la política excesivamente grandes, se establece una cota superior en el valor que puede tomar el cociente de probabilidades $\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{antiguo}}(a_t|s_t)}$ [11].

Definición 3.5. Ventaja subrogada acotada $A^{clip}(\pi_0, \pi_1)$. La ventaja subrogada acotada se obtiene acotando el valor del cociente $\frac{\pi_1(a_t|s_t)}{\pi_0(a_t|s_t)}$ en la definición 3.4 al intervalo $[1 - \epsilon_{clip}, 1 + \epsilon_{clip}]$, donde ϵ_{clip} es un hiperparámetro [11].

$$A^{clip}(\pi_0, \pi_1) = \mathbb{E}_{\pi_0, E}[clip(\frac{\pi_1(a_t|s_t)}{\pi_0(a_t|s_t)}, 1 - \epsilon_{clip}, 1 + \epsilon_{clip}) A^{\pi_0}(s_t, a_t)]$$

donde la función *clip* representa la operación de acotación:

$$clip(x, x_{min}, x_{max}) = min(x_{max}, max(x, x_{min}))$$

El estimador de $A^{clip}(\pi_0, \pi_1)$ se obtiene de forma análoga al de $A(\pi_0, \pi_1)$.

Finalmente, la expresión del primer término de la función de pérdida se obtiene tomando el mínimo entre la ventaja subrogada y la ventaja subrogada acotada [11].

$$\mathcal{L}^A = min(\hat{A}(\pi_0, \pi_1), \hat{A}^{clip}(\pi_0, \pi_1))$$

Para comprender la utilidad de esta función de pérdida, puede ser más sencillo analizar por separado lo que ocurre según el signo de la estimación de la ventaja, $\hat{A}_t$. Para los pares estado-acción

cuya ventaja es positiva, la expresión de la función de pérdida es [21]:

$$\mathcal{L}^A = min(\frac{\pi_1(a_t|s_t)}{\pi_0(a_t|s_t)}, 1 + \epsilon_{clip})\hat{A}_t$$

La ventaja subrogada es mayor cuanto mayor es el cociente de probabilidades $\frac{\pi_1(a_t|s_t)}{\pi_0(a_t|s_t)}$, pero este cociente está acotado superiormente por $1 + \epsilon_{clip}$. Por tanto, $\mathcal{L}^\pi$ está acotada superiormente por $(1 + \epsilon_{clip})\hat{A}_t$.

En cambio, si la ventaja es negativa, la expresión de la función de pérdida es [21]:

$$\mathcal{L}^A = max(\frac{\pi_1(a_t|s_t)}{\pi_0(a_t|s_t)}, 1 - \epsilon_{clip})\hat{A}_t$$

La ventaja subrogada es mayor cuanto menor es el cociente $\frac{\pi_1(a_t|s_t)}{\pi_0(a_t|s_t)}$, pero este cociente está acotado inferiormente por $1 - \epsilon_{clip}$. Por tanto, $\mathcal{L}^A$ está acotada superiormente por $(1 - \epsilon_{clip})\hat{A}_t$.

Como en ambos casos la ventaja subrogada está acotada superiormente, la función de pérdida utilizada es una estimación pesimista de la ventaja subrogada de la política actual respecto de la antigua [11]. Así se desincentiva que la nueva política se aleje demasiado de la antigua, consiguiendo una mayor estabilidad en el aprendizaje.

Estimación de las ventajas Un aspecto importante a considerar es la obtención de las estimaciones $\hat{A}_t$ que aparecen en las ecuaciones anteriores [11]. Recordando la definición que se introdujo anterioremente, la ventaja de un par estado-acción es la diferencia entre el valor de dicho par estado-acción, $Q^\pi(s_t, a_t)$ y el valor del estado, $V^\pi(s_t)$. Como el crítico es un estimador de la función estado-valor, el valor de cada estado de la trayectoria, $V^\pi(s_t)$, se estima mediante el valor proporcionado por el crítico, $V_\omega(s_t)$.

El valor del par (s_t, a_t), $Q^\pi_(s_t, a_t)$, se puede estimar a partir del rendimiento empírico observado en la sección de la trayectoria que va desde el instante t hasta $t + T$. Como se explicó al principio, el rendimiento se calcula como la suma de recompensas con factor de descuento γ. Solo hay que tener en cuenta que, como la trayectoria está truncada, en el último estado s_T es preciso utilizar el valor estimado por el crítico [11].

$$\hat{Q}(s_t, a_t) = \sum_{i=t}^{T-1} \gamma^{i-t} r_{i+1} + \gamma^T V_\omega(s_T)$$

De esta forma, la estimación de $A^\pi_(s_t, a_t)$ es [11]:

$$\hat{A}_t = \hat{Q}(s_t, a_t) - V_\omega(s_t)$$

Este estimador tiene la expresión equivalente [31]:

$$\hat{A}_t = \sum_{i=t}^{T-1} \gamma^{i-t}(r_{i+1} + \gamma V_\omega(s_{i+1}) - V_\omega(s_i)) \tag{1}$$

Para reducir la varianza de este estimador, se utiliza una versión generalizada cuya expresión es [31]:

$$\hat{A}_t = \sum_{i=t}^{T-1} (\gamma\lambda)^{i-t}(r_{i+1} + \gamma V_\omega(s_{i+1}) - V_\omega(s_i)) \tag{2}$$

El hiperparámetro λ controla el compromiso entre sesgo y varianza [31]. Si se toma $\lambda = 1$ se obtiene la expresión (1). Reduciendo el valor de λ se consigue reducir la varianza a costa de introducir un mayor sesgo en la estimación [31].

Durante el entrenamiento, las estimaciones de las ventajas, calculadas a partir de la expresión (2), se estandarizan en cada lote [32]. Nótese que las estimaciones son las mismas durante las ζ épocas que dura cada iteración de entrenamiento, ya que se calculan una sola vez para una trayectoria determinada.

Término de entropía Como se ha mencionado antes, la función de pérdida del actor incluye un término de entropía H, que es la entropía de Shannon de las distribuciones de probabilidad definidas por el actor [11]. Este término se añade para incentivar la exploración de nuevas regiones del espacio de acciones [11]. La entropía de una distribución normal con desviación típica σ se define como [33]:

$$H = \frac{1}{2} log(2\pi e\sigma^2)$$

Así, la entropía solo depende de la varianza de la distribución y es mayor cuanto mayor sea esta. Como la varianza de las distribuciones es independiente del estado observado, todas las distribuciones proporcionadas por el actor tienen la misma varianza, y por tanto, la misma entropía.

La expresión completa de la función de pérdida del actor es [11]:

$$\mathcal{L}^\pi = \mathcal{L}^A - c_H H(\pi_\theta)$$

donde la constante c_H controla la importancia relativa del término de entropía.

Función de pérdida del crítico Por otro lado, el crítico se optimiza minimizando el error cuadrático medio entre los valores $V_\omega(s_t)$ predichos por el crítico y unos valores objetivo, $\hat{V}^\pi(s_t)$ [11], que se obtienen sumando los valores estimados por el crítico tras generar la trayectoria y las ventajas estimadas [32]. La idea detrás de estos valores objetivo es que el nuevo crítico debería mejorar las estimaciones que proporcionó el antiguo crítico, teniendo en cuenta la información proporcionada por la trayectoria, que se resume en las ventajas estimadas.

$$\mathcal{L}^V = \frac{1}{|D|} \sum_{t \in D} (V_\omega(s_t) - \hat{V}^\pi(s_t))^2$$

donde

$$\hat{V}^\pi(s_t) = V_{\omega_{antiguo}}(s_t) + \hat{A}^{\pi_{\theta_{antiguo}}}(s_t, a_t)$$

Paralelización Un detalle relevante sobre la implementación de este algoritmo es que la recopilación de las experiencias del agente se puede paralelizar entre varios entornos para reducir el coste computacional del entrenamiento. En lugar de generar una trayectoria de longitud T interaccionando con un solo entorno, se recopilan N trayectorias de longitud T', con $T = T' * N$ [32]. Estas trayectorias provienen de N entornos independientes que se ejecutan simultáneamente.

4 Simulador

A diferencia de otros paradigmas de aprendizaje automático, el entrenamiento y evaluación de un modelo de aprendizaje por refuerzo requiere disponer de un entorno, ya sea real o simulado, con el que el agente pueda interactuar. En este trabajo se ha utilizado en todo momento un entorno simulado que emula el comportamiento de una población de neuronas sincronizada.

4.1 Modelo teórico

El modelo de Hodgkin-Huxley es el modelo básico que se utiliza en neurología para describir las dinámicas de una neurona. Fue propuesto en 1952 para explicar el funcionamiento del axón gigante del calamar. Se fundamenta en el hecho de que las neuronas, al igual que las demás células, están rodeadas por una membrana semipermeable que separa el interior de la célula del espacio extracelular. La concentración de iones es diferente a ambos lados de la membrana, lo que genera una diferencia de potencial. En el modelo de Hodgkin-Huxley, todos los elementos de la neurona se representan mediante componentes eléctricos. La membrana celular de la neurona funciona como un condensador, es decir, un dispositivo capaz de almacenar carga eléctrica. La circulación de corriente a través de la membrana se produce a través de tres canales iónicos: un canal asociado al movimiento de iones de sodio, otro de iones de potasio y un canal de fuga asociado principalmente al movimiento de iones de cloro. Los canales de sodio y de potasio pueden estar abiertos o bloqueados con una cierta probabilidad. Por tanto, la corriente externa que recibe la neurona se puede descomponer en cuatro componentes: una corriente capacitiva que aumenta la carga del condensador y otras tres corrientes que pasan a través de cada uno de los canales iónicos de la membrana.

A partir de esta caracterización, la evolución en el tiempo de las propiedades eléctricas de una neurona se define mediante un sistema de cuatro ecuaciones diferenciales. La formulación detallada de estas ecuaciones se sale de los objetivos de este trabajo, pero se proporciona una descripción de las propiedades básicas de las neuronas que se derivan de este modelo.

El comportamiento de una neurona pasa por tres fases. En el estado de reposo, la neurona se encuentra en equilibrio y no emite ningún tipo de señal. Cuando recibe una corriente de entrada superior a un cierto umbral, la neurona entra en un estado de excitación, emitiendo un pulso eléctrico de corta duración. Por último, la neurona atraviesa un periodo refractario durante el que no puede ser excitada de nuevo.

Así, la señal emitida por una neurona está formada por una secuencia de pulsos eléctricos de corta duración. Los pulsos siempre están separados entre sí por una distancia mínima, puesto que no es posible que se produzca un nuevo pulso mientras la neurona está en estado refractario. Todos los pulsos emitidos por una neurona tienen una forma similar y un pulso constituye la unidad básica de transmisión de información. La información transmitida en la señal depende del número de pulsos y la separación entre ellos.

Debido a la complejidad de un sistema de ecuaciones diferenciales con cuatro variables, para el propósito de la simulación interesa reducir el modelo de Hodgkin-Huxley a un modelo de dos variables. Este modelo simplificado se conoce como modelo de FitzHugh–Nagumo y es una

generalización del oscilador de Bonhoeffer-van der Pol, que fue propuesto originalmente para modelar las oscilaciones en circuitos electrónicos. Las ecuaciones que describen un oscilador de Bonhoeffer-van der Pol se obtienen a partir de las de un oscilador armónico amortiguado, sustituyendo el coeficiente de amortiguamiento por una función no lineal. De esta forma, se permite que el amortiguamiento pueda ser positivo o negativo, en función de los valores de las variables del sistema.

En el modelo de FitzHugh–Nagumo, la evolución en el tiempo de una neurona está gobernada por las siguientes ecuaciones diferenciales.

$$\begin{cases} \dfrac{\partial u}{\partial t} = u - u^3/3 - y + I \\ \dfrac{\partial v}{\partial t} = \alpha(u + b_0 - b_1 v) \end{cases}$$

donde u es el voltaje de la membrana, v es una variable de recuperación, I es la corriente externa que recibe la neurona, y α, b_0 y b_1 son parámetros del modelo.

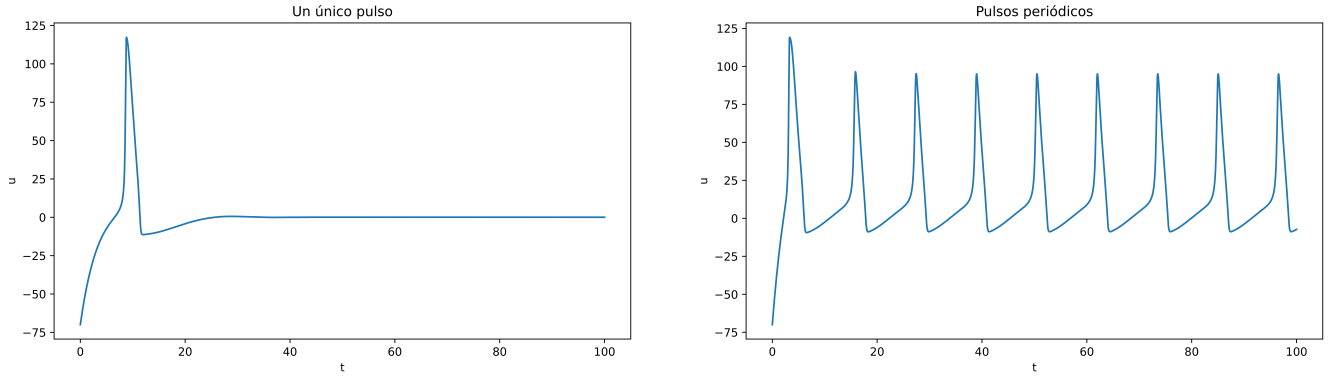

Figura 7: Evolución del potencial de una neurona con diferentes valores de la corriente externa aplicada. Izquierda: Un único pulso. Derecha: Secuencia de varios pulsos.

Dependiendo de la intensidad de la corriente externa aplicada, la neurona puede exhibir diferentes comportamientos: un único pulso seguido del regreso a un estado de reposo, o una secuencia de pulsos (ver Figura 7). La situación que resulta de interés para este trabajo es la aparición de una secuencia periódica de pulsos, es decir, un comportamiento oscilatorio.

Una descripción más detallada de estos modelos neuronales se puede encontrar en [34] y en [35].

El modelo empleado en el simulador, cuya descripción detallada se puede consultar en [2], se compone de una población de K neuronas de FitzHugh–Nagumo, cuyo acoplamiento viene dado por el valor del campo eléctrico global. Así, las ecuaciones del modelo quedan de la siguiente forma.

$$\begin{cases} \dfrac{\partial u_k}{\partial t} = u_k - u_k^3/3 - v_k + \xi_k + \epsilon U + a_t \\ \dfrac{\partial v_k}{\partial t} = \alpha(u_k + b_0 - b_1 v_k) \end{cases}$$

donde u_k es el potencial de la k-ésima neurona, v_k es la variable de recuperación de la k-ésima neurona, ξ_k es una variable de ruido Gaussiano, U es el valor del campo global, ϵ es el factor de acoplamiento y a_t es la corriente externa que reciben las neuronas en cada instante de tiempo.

El valor de U se calcula promediando los potenciales de las neuronas individuales.

$$U = \frac{1}{K} \sum_{k=1}^{K} u_k$$

La corriente externa aplicada es la misma para todas las neuronas y representa la corriente suministrada por el dispositivo DBS. Por este motivo, se denota por a_t, puesto que se trata de la acción en el problema de aprendizaje por refuerzo. Esta corriente tiene la forma de una secuencia de pulsos rectangulares de anchura constante, con una separación entre pulsos también constante. Por tanto, el espacio de acciones del problema es de dimensión 1 y los valores a_t están restringidos al intervalo $[-1, 1]$. En un dispositivo DBS real, la frecuencia de los pulsos producidos y la frecuencia de muestreo del valor de U están determinadas por el hardware. En el simulador, las ecuaciones diferenciales descritas antes se evalúan numéricamente mediante un método de Runge-Kutta, de manera que la frecuencia de los pulsos y la frecuencia de muestreo de las observaciones coinciden con el tamaño de paso seleccionado para la aproximación.

El factor de acoplamiento ϵ controla la fuerza del acoplamiento entre las neuronas, de manera que, a mayor valor de ϵ, mayor es la sincronización de la población de neuronas. Las variables ξ_k son independientes entre sí y siguen una distribución normal de media 0.6 y desviación típica 0.1. Estas variables de ruido se incluyen para evitar que el comportamiento de todas las neuronas sea idéntico. Los valores del resto de coeficientes se han fijado de la siguiente manera: $\alpha = 0.1$, $b_0 = 0.7$, $b_1 = 0.8$.

A semejanza de lo que ocurre en un experimento real, el agente no percibe el estado de las neuronas individuales, solo el valor del campo eléctrico global, U. Como se explicó en el capítulo anterior, un problema de aprendizaje por refuerzo se formula como un proceso de decisión de Markov, lo que significa que, dados el último estado, s_t, y la última acción realizada por el agente, a_t, el nuevo estado, s_{t+1}, es condicionalmente independiente de todo lo que haya ocurrido con anterioridad. Es evidente que, si se toma como observación del estado únicamente el valor de U en cada instante de tiempo, no se cumple la propiedad de Markov, puesto que un periodo de oscilación abarca múltiples instantes de tiempo. No obstante, el proceso se puede asemejar a un proceso de Markov tomando como observación s_t un vector formado por los M valores más recientes de U, donde M es un hiperparámetro cuyo valor tiene que ser suficientemente grande para permitir al agente predecir la evolución del sistema en el siguiente instante de tiempo.

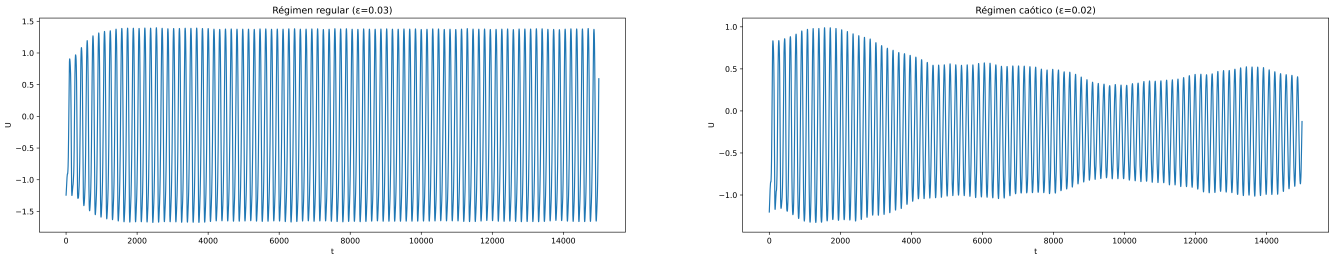

Figura 8: Evolución del valor del campo eléctrico global de una población de neuronas de FitzHugh–Nagumo para diferentes valores del parámetro de acoplamiento ϵ.

Cuando no se aplica ninguna corriente externa, el sistema puede presentar diferentes tipos de comportamiento, dependiendo del valor de ϵ. En la Figura 8 se muestra la evolución en el tiempo del valor de U para diferentes valores del parámetro ϵ. Para algunos valores de ϵ, el sistema

converge a un punto de equilibrio en que el valor del campo U oscila periódicamente en torno a un valor medio que no es conocido de antemano. Con otras configuraciones, el sistema no converge y presenta un comportamiento caótico.

Las oscilaciones que experimenta el valor de U se deben al efecto del acoplamiento entre las neuronas. Si el comportamiento de todas las neuronas individuales fuera independiente, el valor de U se mantendría cercano a un valor constante, con ligeras fluctuaciones aleatorias. Por tanto, el objetivo del problema se puede formular en términos de la supresión de las oscilaciones de U [2]. Así, la función de recompensa del problema se define como una ponderación entre dos términos [2]. El primer término, tal y como se observa en la ecuación (3), incentiva la convergencia del valor de U hacia la media de los M valores anteriores, o equivalentemente, penaliza las oscilaciones en el valor de U. El segundo término favorece las acciones a_t de magnitud pequeña, puesto que, como ya se ha mencionado, esto minimiza el riesgo de aparición de efectos secundarios en los pacientes y aumenta el tiempo de vida de la batería del dispositivo.

$$r_t = -(U_t - \frac{1}{M}\sum_{j=0}^{M-1} U_{t-j})^2 - \beta|a_t| \tag{3}$$

donde β es un hiperparámetro que controla la importancia relativa de los dos términos. El valor utilizado en los experimentos es $\beta = 2$ [2].

4.2 Implementación

El simulador está implementado en lenguaje C++ por razones de eficiencia computacional. Esto no supone ningún inconveniente para integrarlo con el código de Python, ya que Python permite importar módulos escritos en otros lenguajes, abstrayendo completamente los detalles de implementación.

En el código proporcionado por [2], se recurre a la biblioteca *Gymnasium* para encapsular el simulador como un entorno de aprendizaje por refuerzo. Esta es una bibilioteca de aprendizaje por refuerzo mantenida por la Fundación Farama. Las dos clases de esta biblioteca que se utilizan son la clase *Env* y la clase *Space*, cuya documentación se puede consultar en [36].

La clase *Space* sirve para describir conjuntos matemático y se utiliza para definir los espacios de estados y de acciones del problema. Esta clase tiene varias subclases, entre las que se encuentran las siguientes.

- *Box*: Representa un conjunto en $\mathbb{R}^n$.
- *Discrete*: Representa un conjunto finito de números enteros.
- *MultiBinary*: Representa el conjunto de arrays multidimensionales binarios de una dimensión fija.
- *MultiDiscrete*: Representa el producto cartesiano de un número arbitrario de conjuntos finitos de números enteros.

En el problema que se aborda en este trabajo tanto el espacio de estados como el de acciones son continuos, por tanto, se definen mediante la clase *Box*. El espacio de acciones es el intervalo $[-1, 1]$, mientras que el espacio de estados es el conjunto de vectores de dimensión M, con los valores de sus componentes en el intervalo $[-1.5, 1.5]$.

La clase *Env* conceptualiza un entorno de aprendizaje por refuerzo. Los principales atributos de esta clase son los siguientes.

- *action_space*: Objeto de clase *Space* que representa el espacio de acciones.
- *observation_space*: Objeto de clase *Space* que representa el espacio de estados.

Para implementar un entorno, es preciso extender la clase *Env* mediante la implementación de los métodos *step* y *reset*, que se describen a continuación.

El método *reset* se utiliza para inicializar el entorno. Permite como argumento opcional la semilla que se empleará en el generador de números aleatorios del entorno, para garantizar la reproducibilidad de los resultados. El método devuelve un elemento del espacio de estados, que es la observación inicial del entorno. Por tanto, al comienzo de la simulación se muestrea una secuencia de M valores como observación inicial.

El método *step* se utiliza para implementar una interacción del agente con el entorno. Toma como argumento una elemento del espacio de acciones, que es la acción realizada por el agente, y devuelve los siguientes valores:

- Un elemento del espacio de estados, que es la observación del siguiente estado del entorno.
- La recompensa obtenida en la interacción, que siempre es un número real.
- Un valor booleano que indica si se ha alcanzado un estado terminal. Este valor no se utiliza en nuestro problema, ya que no existen estados terminales.
- Un valor booleano que indica si se ha alcanzado una condición de truncamiento, típicamente un límite de tiempo que marca la duración máxima de un episodio.

En los experimentos, se ha establecido una duración de los episodios de 10000 iteraciones, lo que significa que, cuando se alcanza este límite, el simulador se reinicia y el agente recibe como nueva observación una secuencia de M valores que no guardan ninguna relación con los anteriores.

5 Metodología experimental

Para la aplicación de los dos algoritmos propuestos se ha utilizado la biblioteca *Stable-Baselines3* [37]. Se trata de una biblioteca de código abierto que implementa varios algoritmos de aprendizaje por refuerzo en *PyTorch*. Las clases de *Stable-Baselines3* están diseñadas para integrarse con los entornos de la biblioteca *Gymnasium*.

Teniendo en cuenta que los algoritmos estudiados tienen un gran número de hiperparámetros, un estudio exhaustivo de todas las posibles configuraciones e interacciones entre ellas requeriría realizar un número de experimentos enorme. Para evitar que los diseños experimentales tengan un coste computacional excesivo, se ha optado por realizar una selección previa de los hiperparámetros y decisiones de implementación que, a nuestro juicio y en base a la literatura, podrían ser más determinantes en el redimiento de cada algoritmo.

Para cada hiperparámetro, se ha seleccionado un conjunto de valores y se ha entrenado un agente con cada uno de ellos, fijando el resto de hiperparámetros con valores constantes. Después del entrenamiento, se ha medido el rendimiento del agente sobre un nuevo entorno con la misma configuración. Para ello, se aplica la política del agente durante 25000 instantes de tiempo del simulador y se utiliza como criterio la recompensa media obtenida en los últimos 15000 instantes. Las primeras 10000 transiciones se ignoran para permitir que el sistema se haya estabilizado.

Cada experimento se ha repetido cinco veces con diferentes semillas aleatorias para estudiar la variabilidad en los resultados. Los experimentos se han realizado con la configuración del simulador descrita en el Capítulo 4, fijando un valor del factor de acoplamiento $\epsilon = 0.03$. Este valor se ha elegido porque genera un patrón oscilatorio regular, que hemos considerado el más adecuado para este trabajo por su simplicidad. El diseño experimental se ha estructurado de manera secuencial, de forma que si se observa que un determinado valor de un hiperparámetro proporciona una mejora clara, ese valor se utiliza en los siguientes experimentos.

Por último, se evalúa el rendimiento de la mejor configuración de hiperparámetros encontrada para cada algoritmo considerando dos métricas que se describen en la siguiente sección. En esta fase de evaluación, se han entrenado agentes con dos valores distintos del factor de acoplamiento en el simulador: $\epsilon = 0.03$ y $\epsilon = 0.02$, y de nuevo, se han realizado cinco réplicas con cada una de las configuraciones.

5.1 Métricas de evaluación

Para cuantificar la eficacia de un algoritmo de control para un dispositivo DBS, se definen dos métricas que guardan una estrecha relación con los términos de la función de recompensa definida en la ecuación (3). La primera es el coeficiente de supresión, S, definido como el cociente de las desviaciones estándar (std) de los valores de U antes y después del inicio de la estimulación [2].

$$S = \frac{std(U_{antes})}{std(U_{después})}$$

Cuanto mayor sea el valor de S, mayor es la supresión de las oscilaciones del sistema que se logra con la estimulación.

La segunda métrica relevante es el valor absoluto medio de las acciones.

$$\overline{|a|} = \frac{1}{T_{eval}} \sum_{t=1}^{T_{eval}} |a_t|$$

donde T_{eval} es el número de acciones realizadas, o lo que es lo mismo, el número de instantes de tiempo que dura la evaluación del agente. La magnitud de las acciones determina la cantidad de energía suministrada al sistema por el dispositivo DBS, por lo que es deseable que las acciones sean lo más pequeñas posible en valor absoluto.

Estas dos métricas deben interpretarse conjuntamente, puesto que un valor de S muy grande no es indicativo de una buena solución si la magnitud de las acciones aplicadas es también grande. Por este motivo, un algoritmo de control eficaz debería proporcionar un equilibrio entre estos dos objetivos, logrando una supresión significativa de las oscilaciones sin suministrar una cantidad de energía excesivamente alta.

5.2 Diseño experimental del algoritmo DDPG

En los experimentos con este algoritmo se ha utilizado un tamaño de lote de 256 y un valor del factor de descuento para las recompensas $\gamma = 0.99$. Estos valores se han elegido por ser los valores por defecto en la implementación utilizada. El tamaño del *buffer* de reproducción es un millón, que coincide con el número de instantes de tiempo que dura el entrenamiento, por lo que, a efectos prácticos, es equivalente a tener un *buffer* de tamaño ilimitado

El actor y el crítico se implementan con dos MLP idénticos de dos capas ocultas con función de activación ReLU. En la última capa del actor se emplea como función de activación la tangente hiperbólica para que las salidas estén acotadas al intervalo $[-1, 1]$. En la última capa del crítico no se utiliza ninguna función de activación. Los pesos y sesgos de todas las capas de las redes principales se incializan con una distribución uniforme en el intervalo $(-\sqrt{\frac{1}{J}}, \sqrt{\frac{1}{J}})$, donde J es la dimensión de la entrada de la capa. Los parámetros iniciales del actor y el crítico objetivos son una copia de los del actor y el crítico principales.

En el diseño experimental se ha estudiado el efecto de la variación de los siguientes hiperparámetros y decisiones de implementación:

1. Dimensión del espacio de estados (M), es decir, el número de valores anteriores de U que utiliza el agente para tomar las decisiones.
2. Valor del coeficiente de actualización de las redes objetivo (τ).
3. Desviación típica del ruido Gaussiano añadido a las acciones seleccionadas durante el entrenamiento (σ).
4. Número de neuronas de las capas ocultas del actor y el crítico.
5. Tasa de aprendizaje (η).

En la Tabla 1 se exponen las condiciones experimentales utilizadas en cada una de las fases.

Fase	M	τ	Ruido	Tamaño MLP	η
1	Variable	0.005	No	16	0.001
2	8	Variable	No	16	0.001
3	8	0.005	Variable	16	0.001
4	8	0.005	No	Variable	0.001
5	8	0.005	No	16	Variable

Tabla 1: Condiciones experimentales utilizadas para el algoritmo DDPG.

En todos los casos se ha entrenado el modelo durante un millón de iteraciones empleando el optimizador Adam. El entrenamiento comienza en el instante 10000 y durante los 10000 instantes previos se han seleccionado acciones aleatorias para disponer de un conjunto de experiencias amplio antes del entrenamiento.

5.3 Diseño experimental del algoritmo PPO

En los experimentos con este algoritmo se han fijado los valores de los siguientes hiperparámetros:

- Factor de descuento para las recompensas: $\gamma = 0.99$.
- Factor de compromiso entre sesgo y varianza en el estimador de la ventaja: $\lambda = 0.95$.
- Umbral de acotación en el cálculo de la ventaja subrogada: $\epsilon_{clip} = 0.2$.
- Tasa de aprendizaje: $\eta = 3 * 10^{-4}$.

Igual que en el diseño experimental anterior, estos valores se han elegido por ser los valores por defecto en la implementación utilizada.

El actor y el crítico se implementan con dos MLP idénticos de dos capas ocultas empleando como función de activación la tangente hiperbólica. En la última capa del actor y del crítico no se utiliza función de activación. En todas las capas se emplea una inicialización ortogonal para los pesos [38]. Esta inicialización se hace con factor de ganancia $\sqrt{2}$ en las capas ocultas del actor y del crítico, 0.01 en la última capa del actor y 1 en la última capa del crítico. Los sesgos de todas las capas se inicializan a 0.

Para prevenir actualizaciones de los parámetros excesivamente grandes, se aplica una técnica conocida como acotación de gradientes [32]. Sea g la concatenación de los vectores de gradientes de las dos redes (actor y crítico). En cada actualización de los parámetros, se comprueba que la norma 2 de g no supere un cierto umbral, que en este caso toma el valor de 0.5. Si $||g||_2 >= 0.5$, los gradientes se reescalan de la forma:

$$g \leftarrow 0.5 * \frac{g}{||g||_2}$$

En el diseño experimental se ha estudiado el efecto de la variación de los siguientes hiperparámetros:

1. Dimensión del espacio de estados (M).
2. Número de transiciones generadas en cada iteración de entrenamiento (T).
3. Número de entornos ejecutándose en paralelo (N).
4. Tamaño de lote.
5. Número de épocas de entrenamiento en cada iteración (ζ).
6. Coeficiente del término de entropía en la función de pérdida (c_H).
7. Número de neuronas de las capas ocultas del actor y el crítico.

En la Tabla 2 se exponen las condiciones experimentales utilizadas en cada una de las fases.

Fase	M	T	N	Lote	ζ	c_H	Tamaño MLP
1	Variable	16384	8	64	10	0	16
2	256	Variable	8	64	10	0	16
3	256	1024	Variable	64	10	0	16
4	256	1024	8	Variable	10	0	16
5	256	1024	8	128	Variable	0	16
6	256	1024	8	128	10	Variable	16
7	256	1024	8	128	10	0	Variable

Tabla 2: Condiciones experimentales utilizadas para el algoritmo PPO.

En todos los casos se ha entrenado el modelo durante diez millones de instantes de tiempo del simulador empleando el optimizador Adam. El motivo de emplear un número de transiciones mayor para este algoritmo que para el DDPG es que este último realiza una actualización de los parámetros por cada transición generada en el entorno, mientras que el PPO solo actualiza los parámetros después de generar T transiciones. Aunque después de las T transiciones se realicen múltiples actualizaciones de los parámetros, el número de actualizaciones de los parámetros en el PPO es considerablemente menor que el número de transiciones generadas. Por tanto, el coste computacional del PPO es menor que el del DDPG si se entrena con el mismo número de transiciones.

6 Resultados

En las Tablas 5-16 del Anexo se muestra la media y la desviación típica, calculada a partir de cinco réplicas, de la recompensa media obtenida en la evaluación con cada configuración de hiperparámetros estudiada. En las Figuras 9-20 se presentan gráficos de cajas de la recompensa media en la evaluación construidos también con las cinco réplicas, marcando en cada caja la mediana de las réplicas con una línea naranja.

Hay que tener en cuenta que la función de recompensa depende del valor de M, puesto que para calcularla se emplea la media de los M valores más recientes de U. Con el fin de utilizar el mismo criterio de evaluación en todos los casos, en todas las tablas del Anexo y en los gráficos de este capítulo se representa la función de recompensa calculada con los 1000 valores más recientes de U.

6.1 Resultados con el algoritmo DDPG

6.1.1 Variación de M

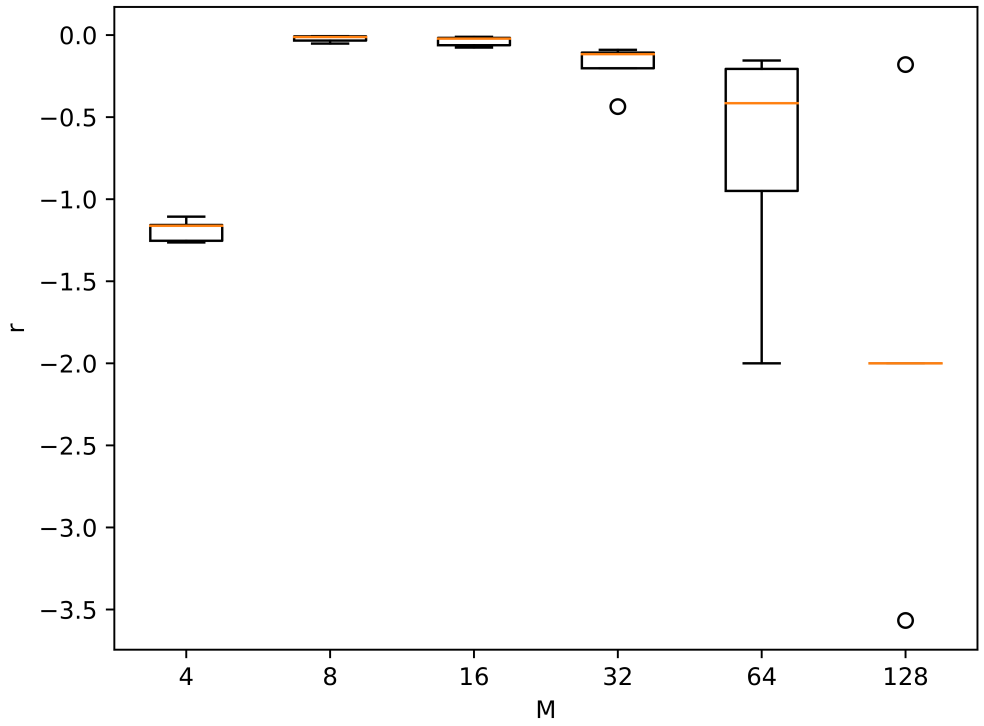

Figura 9: Recompensas medias obtenidas en los experimentos con el algoritmo DDPG en función del valor de M.

Los valores de M considerados son 4, 8, 16, 32, 64 y 128. En la Figura 9 se observa que la elección de este hiperparámetro tiene un impacto decisivo en la eficacia del algoritmo. Las mayores recompensas se obtienen con valores de M entre 8 y 16. Con un valor de M más pequeño, el rendimiento desciende drásticamente, lo que indica que utilizar los 4 valores más recientes de U no es suficiente para predecir adecuadamente el comportamiento del sistema. Sin embargo, para valores de M más altos, el rendimiento también empeora.

Esto puede resultar contraintuitivo, pero se puede explicar porque las redes neuronales del actor y el crítico tienen más parámetros cuanto mayor es el valor de M. Una red neuronal con más parámetros es más difícil de optimizar, porque tiene más facilidad de caer en mínimos locales. Por tanto, se puede concluir que el agente predice de manera óptima la evolución del sistema con un valor de M de aproximadamente 8. Aumentar el valor de M implica proporcionar al agente información innecesaria, o que al menos no es capaz de aprovechar, y en consecuencia, converge a una solución peor.

6.1.2 Variación de τ

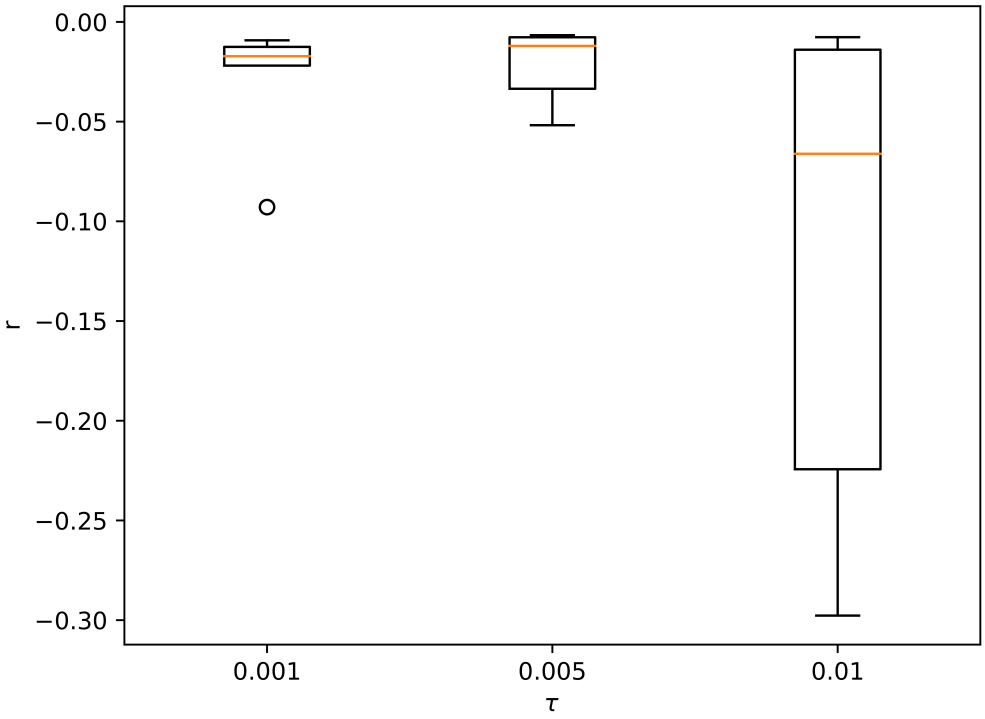

Figura 10: Recompensas medias obtenidas en los experimentos con el algoritmo DDPG en función del valor de τ.

Los valores considerados para τ son 0.001, 0.005 y 0.01. En la Figura 10 se observa que aumentar el valor de τ a 0.01 aumenta la variabilidad de resultados en las diferentes ejecuciones. Esto concuerda con lo que se podría esperar, puesto que un valor más pequeño de τ implica mayor estabilidad de las redes objetivo. En cambio, reducir el valor de τ a 0.001 no parece tener un efecto importante.

6.1.3 Variación del ruido de exploración

Se han tomado como valores de desviación típica para el ruido $\sigma = 0.001$ y $\sigma = 0.01$, comparando su efecto con el de no añadir ningún tipo de ruido. Durante la evaluación no se añade ruido en ningún caso.

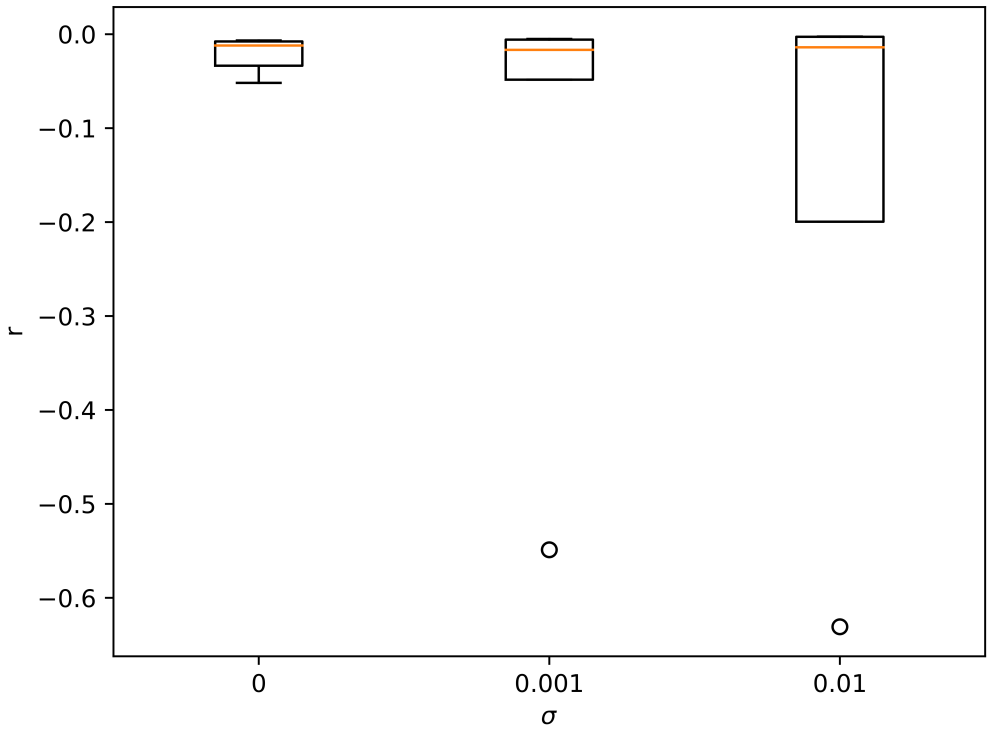

Figura 11: Recompensas medias obtenidas en los experimentos con el algoritmo DDPG en función de la desviación típica del ruido de exploración.

En la Figura 11 se observa que, si bien la mediana de las cinco réplicas no se ve apenas afectada por la incorporación de ruido durante el entrenamiento, sí que aumenta la variabilidad de los resultados, siendo esta mayor cuanto mayor es el valor de σ. Es posible que las características de la función de recompensa de este problema permitan llegar a una solución óptima sin necesidad de introducir aleatoriedad para forzar la exploración, y en consecuencia el ruido de exploración no aporta ningún beneficio.

6.1.4 Variación del tamaño de las capas ocultas de los MLP

Los tamaños considerados para las capas ocultas son 8, 16 y 32. A la vista de la Figura 12, parece que el rendimiento en la mayoría de réplicas es similar para las distintas arquitecturas de red. Sin embargo, con los tamaños 8 y 32 aparece un valor atípico con una recompensa mucho menor, cosa que no se da con el tamaño 16, lo que aparentemente indica que los MLP de tamaño 16 convergen de manera más consistente en este problema.

6.1.5 Variación de la tasa de aprendizaje

La tasa de aprendizaje se ha variado entre los valores 0.0001 y 0.001. En la Figura 13 se ve claramente que la variabilidad de los resultados es mayor cuando se utiliza una tasa de aprendizaje más pequeña, y que en promedio se llega a soluciones peores.

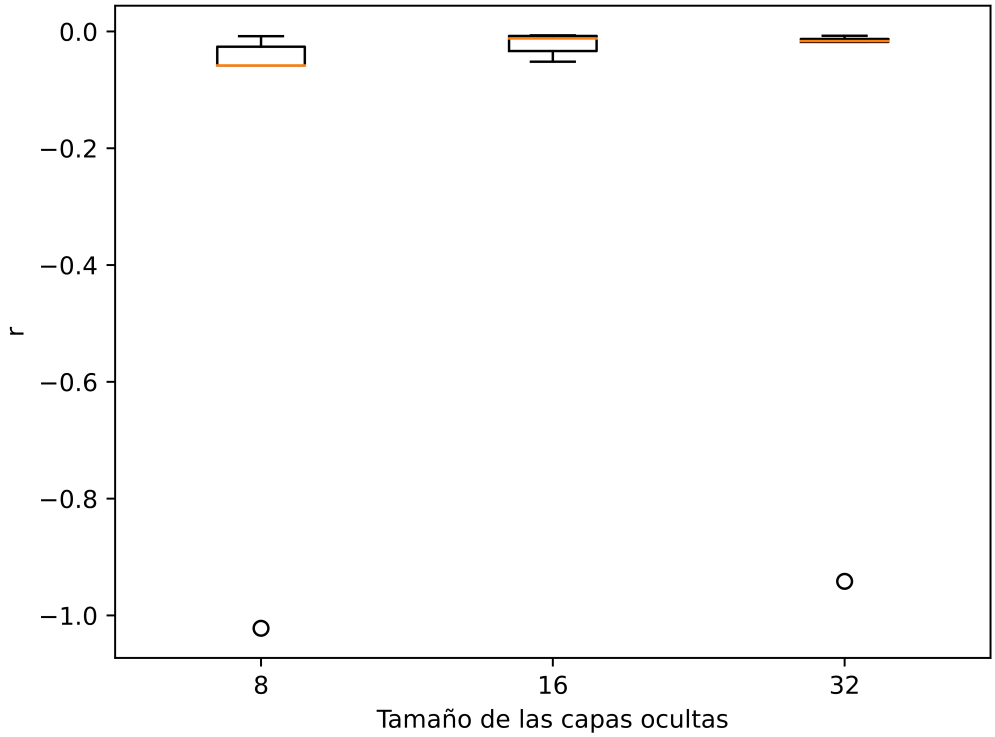

Figura 12: Recompensas medias obtenidas en los experimentos con el algoritmo DDPG en función del tamaño de las capas ocultas de los MLP.

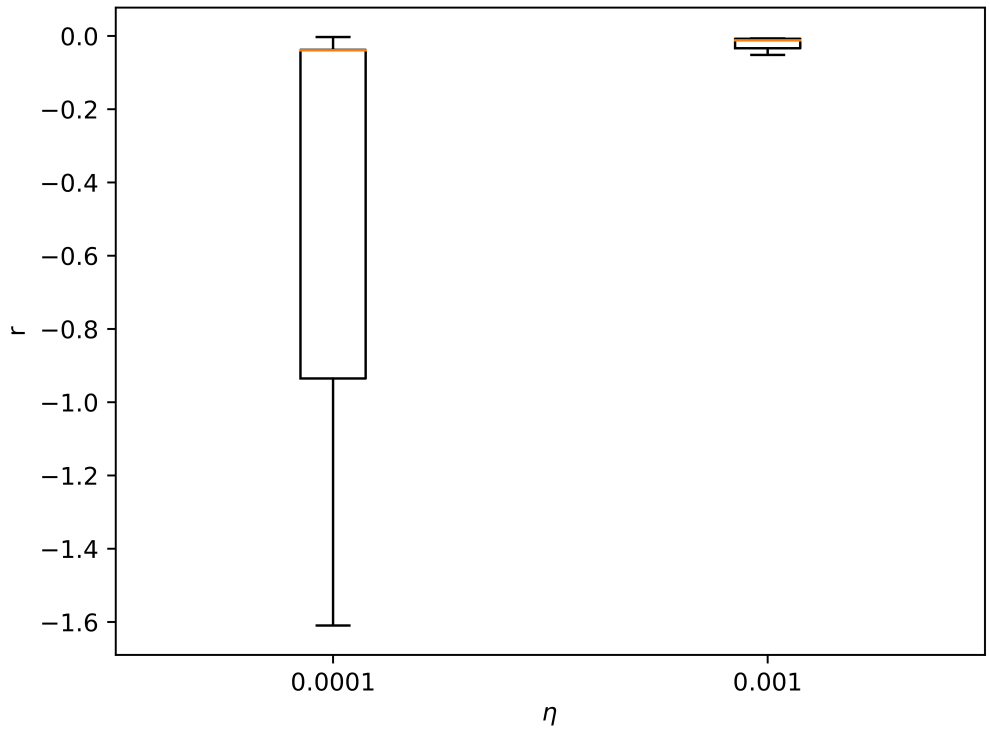

Figura 13: Recompensas medias obtenidas en los experimentos con el algoritmo DDPG en función del valor de η.

6.2 Resultados con el algoritmo PPO

6.2.1 Variación de M

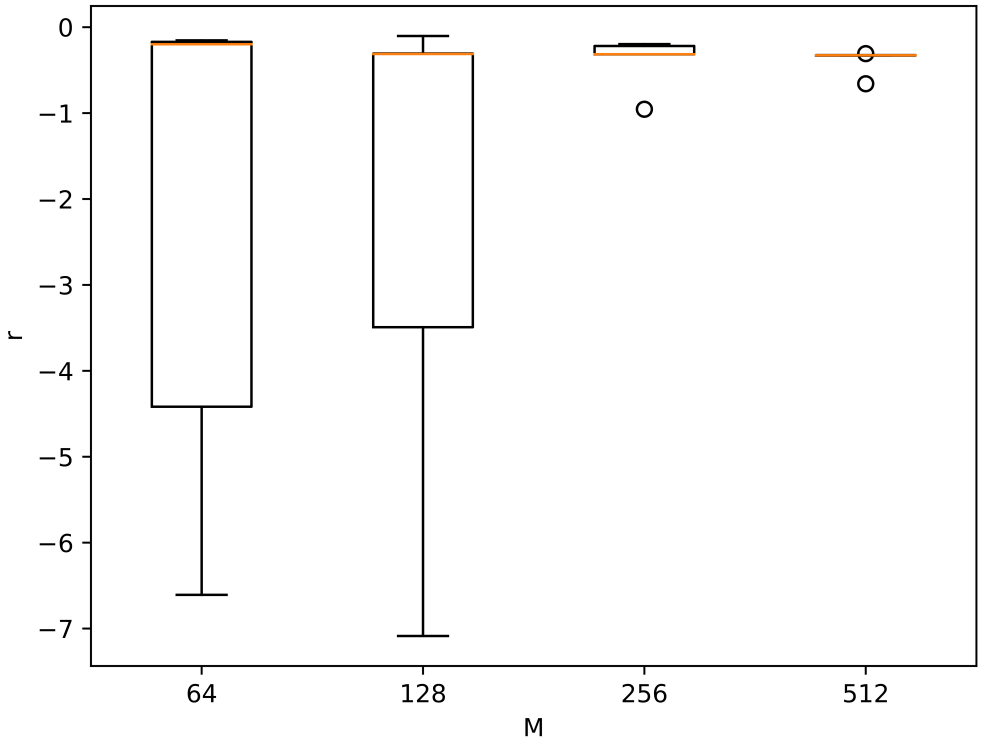

Figura 14: Recompensas medias obtenidas en los experimentos con el algoritmo PPO en función del valor de M.

Los valores de M considerados son 64, 128, 256 y 512. Sorprendentemente, el efecto de este hiperparámetro es opuesto al que tiene en el algoritmo DDPG. En la Figura 14 se pone de manifiesto que el valor mínimo de M necesario para predecir adecuadamente el comportamiento del sistema es mucho mayor que en el otro algoritmo, ya que se encuentra entre 128 y 256. A la vista del primer gráfico de la Figura 8, que refleja la configuración del simulador utilizada en estos experimentos, este valor mínimo podría ser cercano al periodo de la señal. Si se utiliza un valor de M por debajo de ese umbral, el rendimiento empeora notablemente, sin embargo, utilizar un valor de M mayor de lo necesario no deteriora el rendimiento.

6.2.2 Variación de T

Los valores considerados para T son 512, 1024, 2048, 4096, 8192 y 16384. Como se han empleado 8 entornos en paralelo, la longitud de las trayectorias generadas en cada uno de ellos es $T/8$. En la Figura 15 se observa que el valor de este hiperparámetro es de gran importancia para el buen desempeño del algoritmo. El valor óptimo se encuentra en torno a 1024. Si se utiliza un valor demasiado pequeño, el rendimiento empeora drásticamente, lo que puede explicarse porque una muestra pequeña tiene mayor probabilidad de estar sesgada.

Sin embargo para valores más grandes también se produce un deterioro del rendimiento.

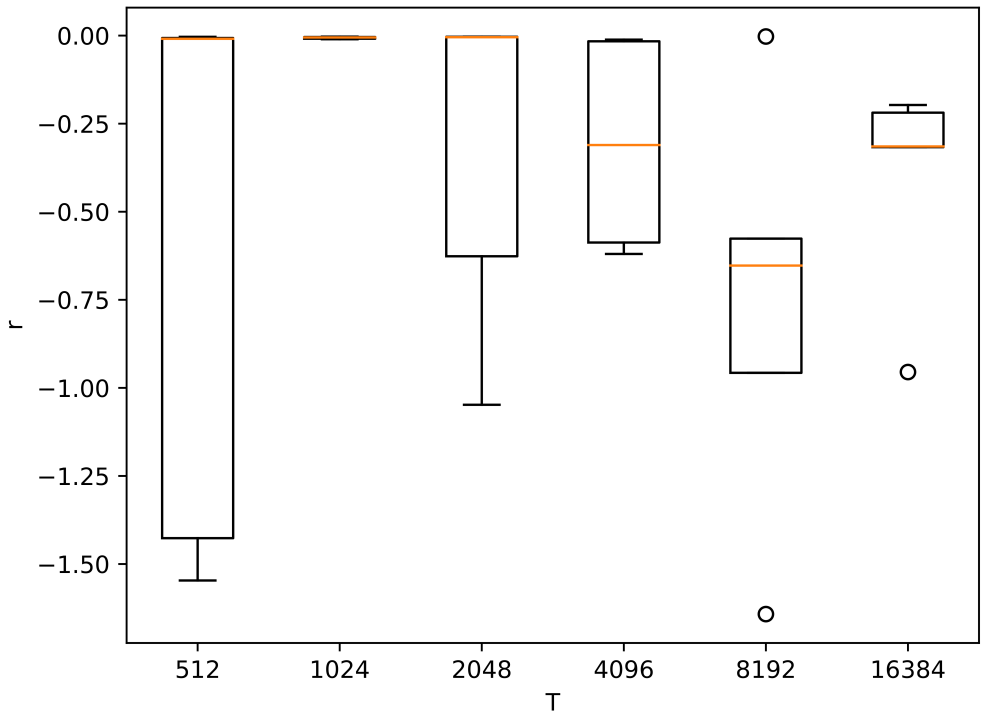

Figura 15: Recompensas medias obtenidas en los experimentos con el algoritmo PPO en función del número de experiencias utilizadas en cada iteración de entrenamiento.

Teniendo en cuenta que el tamaño de lote es constante, cuanto mayor es el número de experiencias recolectadas, se realiza un mayor número de actualizaciones de los parámetros en cada iteración de entrenamiento. Planteamos la hipótesis de que realizar un gran número de actualizaciones de los parámetros en cada iteración puede llevar a que la nueva política se aleje excesivamente de la anterior, introduciendo inestabilidad en el aprendizaje y contrarrestando el efecto que se pretende lograr con la acotación de la ventaja subrogada.

6.2.3 Variación del número de entornos

Se han considerado los valores de N: 4, 8, 16 y 32. Como en cada iteración de entrenamiento se utilizan 1024 experiencias, la longitud de las trayectorias generadas en cada entorno es $1024/N$. La eficacia del algoritmo no parece verse excesivamente influida por el número de entornos cuando se utilizan 8 o menos (ver Figura 16). Esto supone una gran ventaja, ya que permite reducir el tiempo de entrenamiento mediante la paralelización de 8 simulaciones a la hora de generar las trayectorias. Si se utiliza un número de entornos mayor, el rendimiento empieza a deteriorarse, probablemente a causa de que la longitud de las trayectorias se reduce y en consecuencia, las estimaciones de las ventajas son de peor calidad.

6.2.4 Variación del tamaño de lote

Los valores considerados para el tamaño de lote son 32, 64, 128 y 256. El rendimiento del algoritmo es similar para tamaños de lote de 64 y 128 (ver Figura 17). Con un tamaño de 256

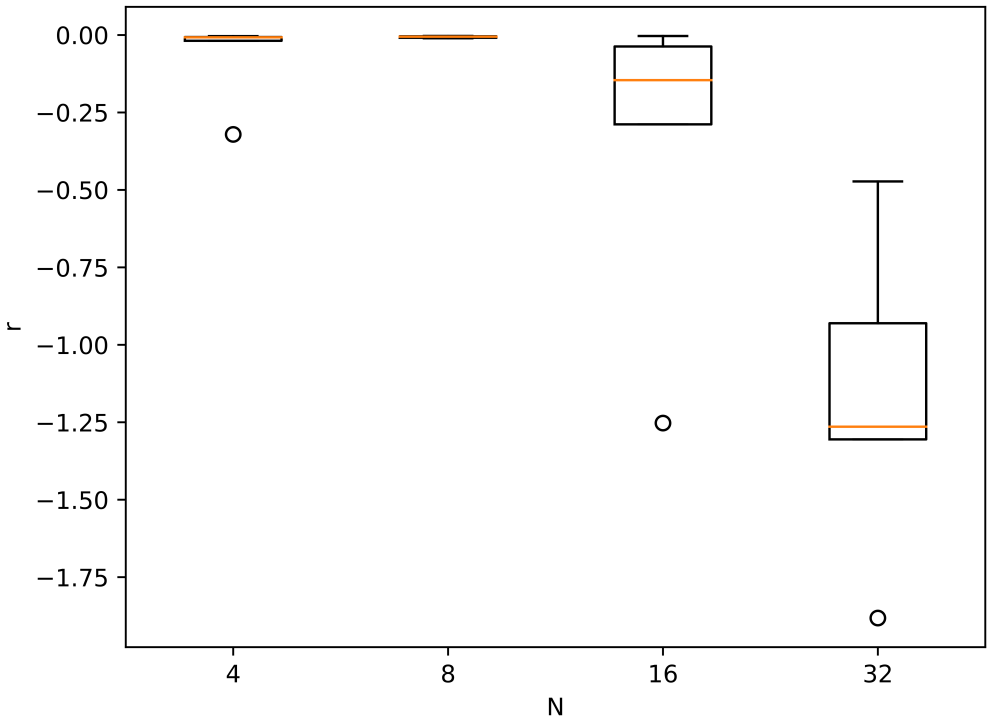

Figura 16: Recompensas medias obtenidas en los experimentos con el algoritmo PPO en función del número de entornos en paralelo.

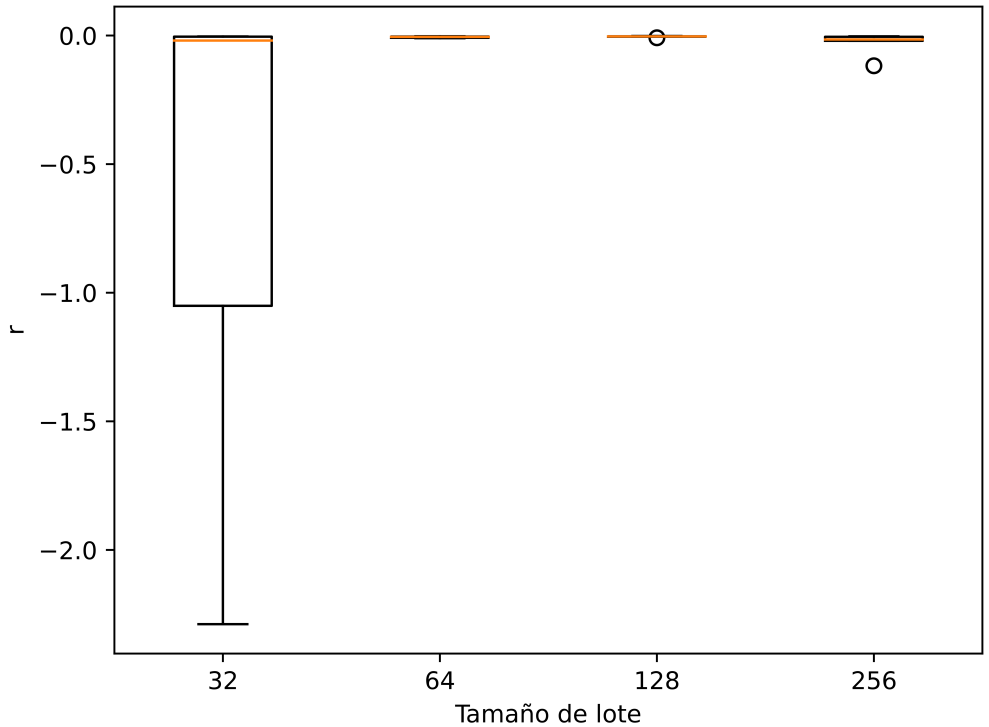

Figura 17: Recompensas medias obtenidas en los experimentos con el algoritmo PPO en función del tamaño de lote.

se observa un ligero empeoramiento, mientras que con un tamaño de 32 el rendimiento empeora drásticamente. Esta observación es coherente con la hipótesis expuesta anteriormente, en la que se planteaba que un número excesivo de actualizaciones de los parámetros en cada iteración de entrenamiento es perjudicial para la estabilidad del aprendizaje.

Comparando las Figuras 15 y 17, se puede ver esta idea de forma más clara. En la Figura 15, al pasar de $T = 1024$ a $T = 2048$, con tamaño de lote 64 en ambos casos, el número de actualizaciones de los parámetros por iteración se duplica, y se observa una caída del rendimiento. En la Figura 17, al pasar de tamaño de lote 64 a 32, con $T = 1024$ en ambos casos, el número de actualizaciones de los parámetros por iteración también se duplica, y también se observa una caída del rendimiento.

6.2.5 Variación del número de épocas de entrenamiento

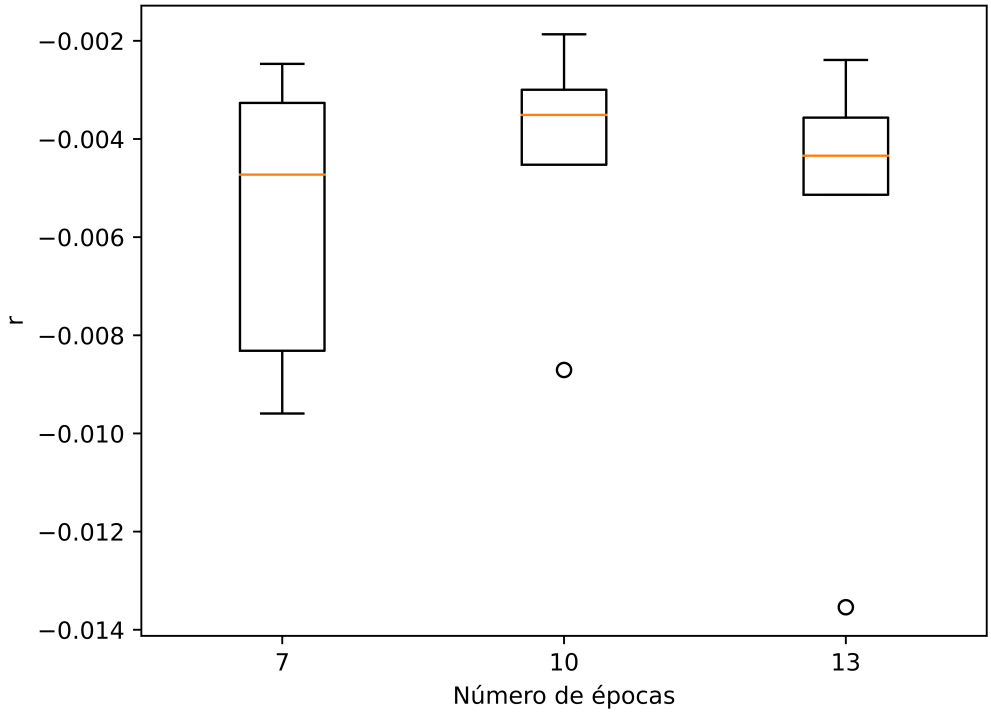

Figura 18: Recompensas medias obtenidas en los experimentos con el algoritmo PPO en función del número de épocas de entrenamiento en cada iteración.

Teniendo en cuenta que el valor por defecto para el número de épocas en la implementación utilizada es 10, se han considerado los valores 7, 10 y 13. No parece haber una diferencia significativa en el rendimiento dentro del rango de ζ que se ha estudiado (ver Figura 18).

6.2.6 Variación de la ponderación del término de entropía

Se ha dado al coeficiente c_H los valores 0, 0.001 y 0.01. En la Figura 19 se observa que un valor de $c_H = 0.001$ es demasiado pequeño para que tenga algún efecto notable, mientras que un valor de 0.01 aumenta la variabilidad de los resultados, haciendo que se obtenga un rendimiento mucho peor en una de las réplicas. A la vista de estos resultados se puede concluir que incluir el término

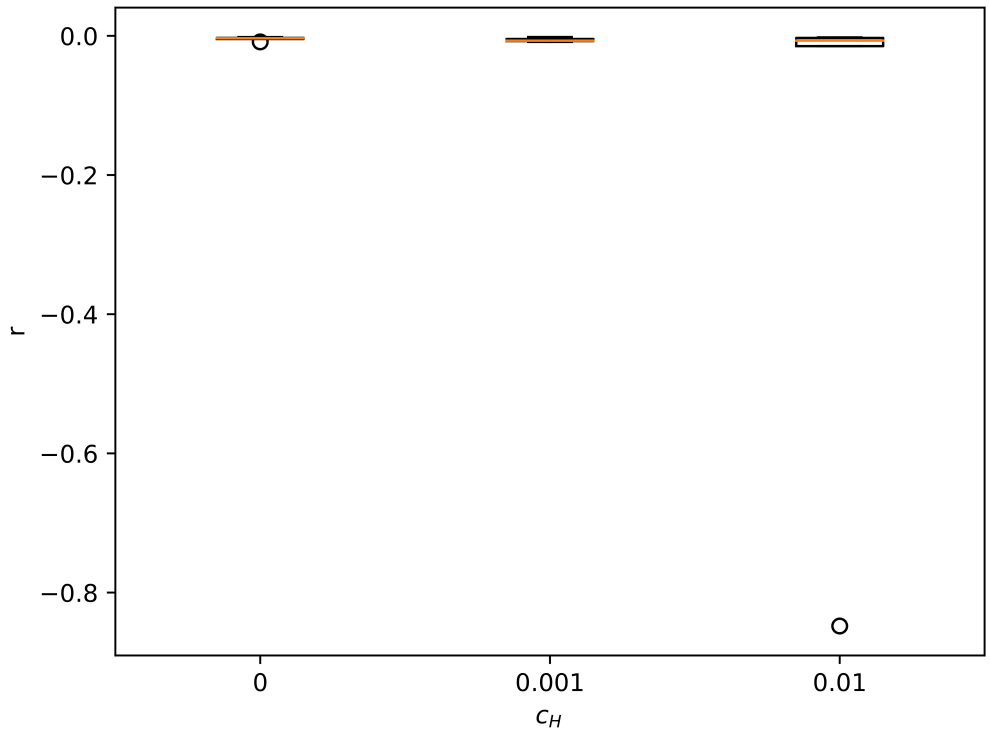

Figura 19: Recompensas medias obtenidas en los experimentos con el algoritmo PPO en función de la ponderación del término de entropía en la función de pérdida.

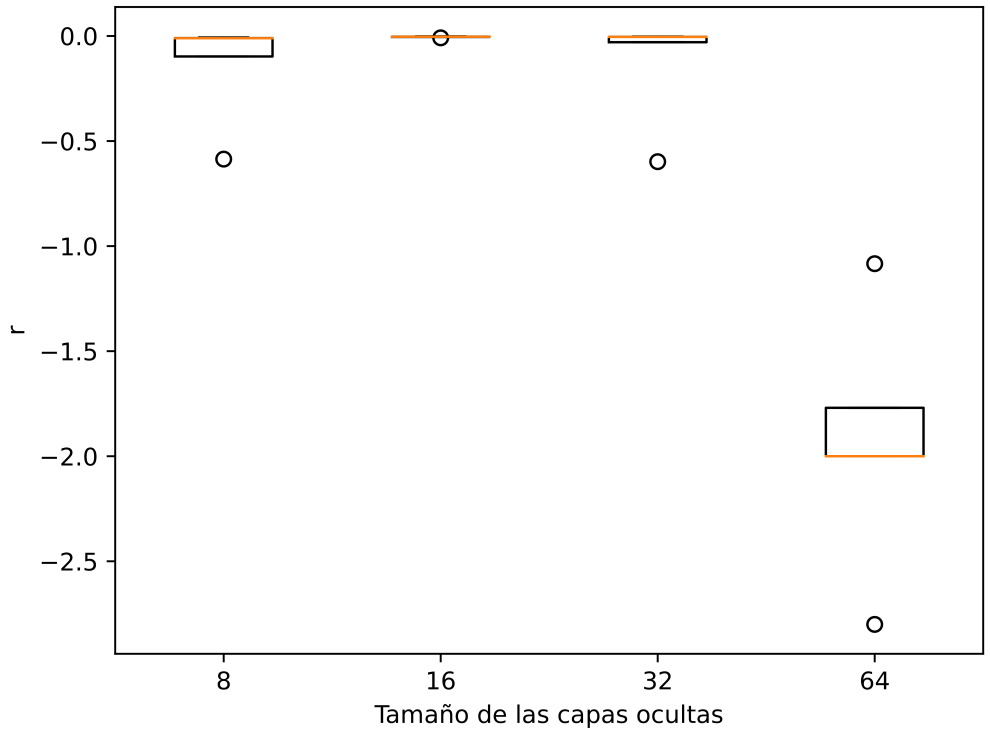

Figura 20: Recompensas medias obtenidas en los experimentos con el algoritmo PPO en función del tamaño de las capas ocultas de los MLP.

de entropía es perjudicial en este problema. Para el algoritmo DDPG se llegaba a una conclusión similar respecto al ruido añadido a las acciones durante el entrenamiento, que es el mecanismo equivalente al término de entropía para fomentar la exploración, lo que apoya la hipótesis de que en este problema no se requiere incentivar la exploración para llegar a la solución óptima.

6.2.7 Variación del tamaño de las capas ocultas de los MLP

Los tamaños considerados para las capas ocultas son 8, 16, 32 y 64. Los resultados son similares a los que se obtienen para el algoritmo DDPG. El tamaño que mejor parece funcionar es 16, ya que con tamaños de 8 y de 32 aumenta la variabilidad de los resultados y se llega a una solución mucho peor en una de las réplicas (ver Figura 20). En este diseño experimental también se han incluido redes de tamaño 64, y se observa que las recompensas obtenidas son mucho menores que con las redes más pequeñas. Como ya se ha comentado, las redes con mayor número de parámetros son más difíciles de optimizar, porque tienen más facilidad de quedarse en mínimos locales.

6.3 Evaluación

6.3.1 Evaluación del algoritmo DDPG

En base a los resultados obtenidos, la configuración del algoritmo que se ha seleccionado es la que utiliza $M = 8$, $\tau = 0.005$, $\eta = 0.001$, tamaño 16 de las capas ocultas de los MLP y no utiliza ruido de exploración. En la Tabla 3 se expone el coeficiente de supresión, S, obtenido con este algoritmo y el valor absoluto medio de las acciones en cinco réplicas diferentes con dos configuraciones distintas del simulador.

ϵ	Réplica	1	2	3	4	5	Media
0.03	S	10.25	7.497	51.43	6.055	52.41	25.53
	$\overline{\lvert a \rvert}$	0.0007723	0.006357	0.003635	0.01103	0.003128	0.004984
0.02	S	8.941	37.88	32.94	40.95	1.458	24.43
	$\overline{\lvert a \rvert}$	0.001793	0.007435	0.0005777	0.007779	0.01312	0.006141

Tabla 3: Métricas de evaluación obtenidas para el algoritmo DDPG en cada una de las cinco réplicas con cada una de las configuraciones del simulador.

Con la primera configuración del simulador se consigue una supresión razonable de las oscilaciones en todas las réplicas, si bien en dos de ellas se alcanza una supresión mucho mayor que en las otras. Con la segunda configuración se consigue una supresión razonable en cuatro de las cinco réplicas y en la restante no se consigue suprimir las oscilaciones. La supresión media que se consigue en ambas configuraciones es similar, pero la supresión máxima con $\epsilon = 0.03$ es mayor que con $\epsilon = 0.02$, lo que se explica porque la primera configuración presenta oscilaciones regulares y es más fácil predecir el comportamiento del sistema. El valor absoluto medio de las acciones está en torno a 0.005 con $\epsilon = 0.03$ y 0.006 con $\epsilon = 0.02$.

6.3.2 Evaluación del algoritmo PPO

En base a los resultados obtenidos, la configuración del algoritmo que se ha seleccionado es la que utiliza $M = 256$, $T = 1024$, $N = 8$, tamaño de lote 128, $\zeta = 10$, tamaño 16 de las capas ocultas de los MLP y no incluye el término de entropía en la función de pérdida. En la Tabla 4 se expone el coeficiente de supresión, S, obtenido con este algoritmo y el valor absoluto medio de las acciones en cinco réplicas diferentes con dos configuraciones distintas del simulador.

ϵ	Réplica	1	2	3	4	5	Media
0.03	S	28.04	45.40	14.43	47.08	57.72	38.53
	$\overline{\lvert a \rvert}$	0.0007503	0.001980	0.001611	0.0006703	0.001580	0.001318
0.02	S	2.516	19.77	37.64	13.79	38.54	22.45
	$\overline{\lvert a \rvert}$	0.001284	0.0008716	0.001145	0.0007535	0.0007766	0.0009661

Tabla 4: Métricas de evaluación obtenidas para el algoritmo PPO en cada una de las cinco réplicas con cada una de las configuraciones del simulador.

Con la primera configuración del simulador se consigue una supresión considerable de las oscilaciones en todas las réplicas. Con la segunda configuración se consigue una supresión razonable en cuatro de las cinco réplicas y en la restante la supresión es muy baja. Tanto la supresión media como la supresión máxima son mayores con la primera configuración del simulador que con la segunda. El valor absoluto medio de las acciones está en torno a 0.001 con ambas configuraciones.

6.3.3 Comparación

Se puede concluir que ambos algoritmos son capaces de encontrar políticas eficaces en ambas configuraciones del simulador con los hiperparámetros seleccionados, pero existe una importante variabilidad entre las diferentes réplicas. Con la segunda configuración del simulador, en ocasiones no se converge a una solución aceptable, lo que indica que la supresión de las oscilaciones es más difícil cuando estas no son regulares. En las Figuras 21-24 del Anexo se representa gráficamente la evolución del campo global U en la mejor de las cinco réplicas, para cada uno de los algoritmos y configuraciones del simulador.

Como referencia para evaluar la magnitud de las acciones, hay que tener en cuenta que, para suprimir las oscilaciones empleando una acción de valor constante, se requiere un valor próximo a 0.05. Este valor está un orden de magnitud por encima de las acciones aplicadas por ambos algoritmos.

Con la primera configuración del simulador, el valor promedio de S en las cinco réplicas que se obtiene con el algoritmo PPO es aproximadamente 1.5 veces superior al que se obtiene con el DDPG, y el valor absoluto medio de las acciones es en torno a cuatro veces menor. Con la segunda configuración del simulador, el valor promedio de S obtenido con el algoritmo PPO es ligeramente inferior al que se obtiene con el DDPG, pero el valor absoluto medio de las acciones es más de seis veces menor.

7 Conclusiones

Los resultados obtenidos reflejan que los dos algoritmos propuestos son capaces de aprender una política para suprimir las oscilaciones colectivas de un grupo de neuronas sincronizadas, sin utilizar ningún conocimiento explícito acerca de las dinámicas de dicho sistema neuronal. Se pone de manifiesto la importancia que tiene en los algoritmos de aprendizaje por refuerzo seleccionar los hiperparámetros adecuados para el problema al que se van a aplicar, ya que una elección de hiperparámetros inadecuada puede ser la causa de que el algoritmo no logre el objetivo esperado. En concreto, en los experimentos realizados, se observa que el algoritmo PPO obtiene políticas más eficientes que el DDPG, en lo que respecta a la cantidad de energía suministrada al sistema. En cualquier caso, las políticas aprendidas por ambos algoritmos son notablemente más eficientes que la estimulación con pulsos de frecuencia constante.

Es importante recalcar que en este trabajo se ha utilizado una formulación muy simplificada del problema de control de la DBS. Los modelos que se utilizan en neurología para caracterizar enfermedades como el Párkinson son más complejos que el que se ha considerado en este trabajo, puesto que incluyen múltiples poblaciones de neuronas que interactúan entre sí con conexiones sinápticas excitadoras o inhibidoras [6]. En estos modelos, se pueden definir otras métricas de evaluación, como la tasa de errores de transmisión de información de determinados tipos de neuronas. En consecuencia, se propone como trabajo futuro la aplicación de los algoritmos estudiados en este trabajo a otros modelos neuronales simulados que describan de manera más detallada y realista las dinámicas del cerebro.

También tendría interés la aplicación de otros algoritmos de aprendizaje por refuerzo desarrollados más recientemente, por ejemplo, las mejoras al algoritmo DDPG descritas en [39].

Otro punto importante a estudiar sería si las políticas aprendidas se pueden aplicar de manera eficaz a múltiples tipos de dinámicas neuronales sin necesidad de un reentrenamiento. Conseguir este objetivo facilitaría en gran medida la aplicación de esta metodología en un dispositivo DBS real, ya que se podría aprender una política en un entorno simulado y después trasladar los parámetros aprendidos al dispositivo. Esta estrategia, unida al uso de una red de pequeño tamaño para implementar la política, como las que se han empleado en este trabajo, sería especialmente adecuada si el dispositivo tiene una memoria limitada y escasa capacidad computacional.

Referencias

[1] Alim Louis Benabid. "Deep brain stimulation for Parkinson's disease". En: *Current Opinion in Neurobiology* 13.6 (2003), págs. 696-706. ISSN: 0959-4388. DOI: https://doi.org/10.1016/j.conb.2003.11.001. URL: https://www.sciencedirect.com/science/article/pii/S0959438803001739.

[2] Dmitrii Krylov et al. "Reinforcement learning framework for deep brain stimulation study". En: *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*. IJCAI'20. Yokohama, Yokohama, Japan, 2021. ISBN: 9780999241165.

[3] Wikipedia. *Closed-loop controller*. 2024. URL: https://en.wikipedia.org/wiki/Closed-loop_controller (visitado 25-05-2024).

[4] Chao-Hung Kuo, Gabrielle White-Dzuro y Andrew Ko. "Approaches to closed-loop deep brain stimulation for movement disorders". En: *Neurosurgical Focus* 45 (ago. de 2018), E2. DOI: 10.3171/2018.5.FOCUS18173.

[5] Anne Beuter, Jean-Pascal Lefaucheur y Julien Modolo. "Closed-loop cortical neuromodulation in Parkinson's disease: An alternative to deep brain stimulation?" En: *Clinical Neurophysiology* 125.5 (2014), págs. 874-885. ISSN: 1388-2457. DOI: https://doi.org/10.1016/j.clinph.2014.01.006. URL: https://www.sciencedirect.com/science/article/pii/S1388245714000376.

[6] Meili Lu et al. "Application of Reinforcement Learning to Deep Brain Stimulation in a Computational Model of Parkinson's Disease". En: *IEEE Transactions on Neural Systems and Rehabilitation Engineering* 28.1 (2020), págs. 339-349. DOI: 10.1109/TNSRE.2019.2952637.

[7] National Institute of Neurological Disorders y Stroke. *Deep Brain Stimulation for Movement Disorders*. 2024. URL: https://www.ninds.nih.gov/health-information/disorders/deep-brain-stimulation-movement-disorders (visitado 26-05-2024).

[8] David J. Burn y Alexander I. Tröster. "Neuropsychiatric Complications of Medical and Surgical Therapies for Parkinson's Disease". En: *Journal of Geriatric Psychiatry and Neurology* 17.3 (2004). PMID: 15312281, págs. 172-180. DOI: 10.1177/0891988704267466. eprint: https://doi.org/10.1177/0891988704267466. URL: https://doi.org/10.1177/0891988704267466.

[9] Xue Bin Peng et al. "Sim-to-Real Transfer of Robotic Control with Dynamics Randomization". En: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. Brisbane, Australia: IEEE Press, 2018, págs. 1-8. DOI: 10.1109/ICRA.2018.8460528. URL: https://doi.org/10.1109/ICRA.2018.8460528.

[10] David Silver et al. "Deterministic policy gradient algorithms". En: *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*. ICML'14. Beijing, China: JMLR.org, 2014, I–387–I-395.

[11] John Schulman et al. "Proximal Policy Optimization Algorithms". En: *ArXiv* abs/1707.06347 (2017). URL: https://api.semanticscholar.org/CorpusID:28695052.

[12] Donald Michie, David J. Spiegelhalter y Charles C. Taylor. "Machine Learning, Neural and Statistical Classification". En: 2009. URL: https://api.semanticscholar.org/CorpusID:15773445.

[13] Michael A. Nielsen. *Neural Networks and Deep Learning*. misc. 2018. URL: http://neuralnetworksanddeeplearning.com/.

[14] Ken-Ichi Funahashi. "On the approximate realization of continuous mappings by neural networks". En: *Neural Networks* 2.3 (1989), págs. 183-192. ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(89)90003-8. URL: https://www.sciencedirect.com/science/article/pii/0893608089900038.

[15] Xavier Glorot, Antoine Bordes y Yoshua Bengio. "Deep Sparse Rectifier Neural Networks". En: *International Conference on Artificial Intelligence and Statistics*. 2011. URL: https://api.semanticscholar.org/CorpusID:2239473.

[16] Sunitha Basodi et al. "Gradient amplification: An efficient way to train deep neural networks". En: *Big Data Mining and Analytics* 3.3 (2020), págs. 196-207. DOI: 10.26599/BDMA.2020.9020004.

[17] Geoffrey Hinton. *Lecture 6d - A separate, adaptive learning rate for each connection*. Neural Networks for Machine Learning. 2012.

[18] Aston Zhang et al. *Dive into Deep Learning*. https://D2L.ai. Cambridge University Press, 2023.

[19] John Duchi, Elad Hazan y Yoram Singer. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". En: *J. Mach. Learn. Res.* 12.null (jul. de 2011), págs. 2121-2159. ISSN: 1532-4435.

[20] Diederik P. Kingma y Jimmy Ba. "Adam: A Method for Stochastic Optimization". En: *CoRR* abs/1412.6980 (2014). URL: https://api.semanticscholar.org/CorpusID:6628106.

[21] OpenAI. *Spinning Up*. 2018. URL: https://spinningup.openai.com/en/latest/ (visitado 28-04-2024).

[22] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". En: *Nature* 518 (2015), págs. 529-533. URL: https://api.semanticscholar.org/CorpusID:205242740.

[23] OpenAI: Marcin Andrychowicz et al. "Learning dexterous in-hand manipulation". En: *Int. J. Rob. Res.* 39.1 (ene. de 2020), págs. 3-20. ISSN: 0278-3649. DOI: 10.1177/0278364919887447. URL: https://doi.org/10.1177/0278364919887447.

[24] Tuomas Haarnoja et al. "Learning to Walk via Deep Reinforcement Learning". En: *ArXiv* abs/1812.11103 (2018). URL: https://api.semanticscholar.org/CorpusID:57189150.

[25] April Yu, Raphael Palefsky-Smith y Rishi Bedi. "Deep Reinforcement Learning for Simulated Autonomous Vehicle Control". En: 2016. URL: https://api.semanticscholar.org/CorpusID:33671880.

[26] Juan Chen, Zhengxuan Xue y Daiqian Fan. "Deep Reinforcement Learning Based Left-Turn Connected and Automated Vehicle Control at Signalized Intersection in Vehicle-to-Infrastructure Environment". En: *Inf.* 11 (2020), pág. 77. URL: https://api.semanticscholar.org/CorpusID:213372053.

[27] Christopher Berner et al. "Dota 2 with Large Scale Deep Reinforcement Learning". En: *ArXiv* abs/1912.06680 (2019). URL: https://api.semanticscholar.org/CorpusID:209376771.

[28] Timothy P. Lillicrap et al. "Continuous control with deep reinforcement learning". En: *CoRR* abs/1509.02971 (2015). URL: https://api.semanticscholar.org/CorpusID:16326763.

[29] Vijay R. Konda y John N. Tsitsiklis. "Actor-Critic Algorithms". En: *Neural Information Processing Systems*. 1999. URL: https://api.semanticscholar.org/CorpusID:207779694.

[30] John Schulman et al. "Trust region policy optimization". En: *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*. ICML'15. Lille, France: JMLR.org, 2015, págs. 1889-1897.

[31] John Schulman et al. "High-Dimensional Continuous Control Using Generalized Advantage Estimation". En: *CoRR* abs/1506.02438 (2015). URL: https://api.semanticscholar.org/CorpusID:3075448.

[32] Shengyi Huang et al. "The 37 Implementation Details of Proximal Policy Optimization". En: *ICLR Blog Track*. 2022.

[33] Kenneth H. Norwich. "Information, sensation, and perception". En: 1993. URL: https://api.semanticscholar.org/CorpusID:142326841.

[34] Wulfram Gerstner y Werner M. Kistler. *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge University Press, 2002.

[35] Daniel Cebrián-Lacasa et al. "Six decades of the FitzHugh-Nagumo model: A guide through its spatio-temporal dynamics and influence across disciplines". En: 2024. URL: https://api.semanticscholar.org/CorpusID:269187655.

[36] Farama Foundation. *Gymnasium Documentation*. 2023. URL: https://gymnasium.farama.org/ (visitado 02-06-2024).

[37] Antonin Raffin et al. "Stable-Baselines3: Reliable Reinforcement Learning Implementations". En: *Journal of Machine Learning Research* 22.268 (2021), págs. 1-8. URL: http://jmlr.org/papers/v22/20-1364.html.

[38] Andrew M. Saxe, James L. McClelland y Surya Ganguli. "Exact solutions to the nonlinear dynamics of learning in deep linear neural networks". En: *CoRR* abs/1312.6120 (2013). URL: https://api.semanticscholar.org/CorpusID:17272965.

[39] Scott Fujimoto, Herke van Hoof y David Meger. "Addressing Function Approximation Error in Actor-Critic Methods". En: *International Conference on Machine Learning*. 2018. URL: https://api.semanticscholar.org/CorpusID:3544558.

Anexo

M	$media(\overline{r})$	$std(\overline{r})$
4	-1.188	0.06044
8	-0.02234	0.01766
16	-0.03747	0.02633
32	-0.1902	0.1288
64	-0.7452	0.6876
128	-1.949	1.073

Tabla 5: Media y desviación típica de la recompensa media obtenida en los experimentos con el algoritmo DDPG en función del valor de M.

τ	$media(\overline{r})$	$std(\overline{r})$
0.001	-0.03075	0.03136
0.005	-0.02234	0.01766
0.01	-0.1220	0.1176

Tabla 6: Media y desviación típica de la recompensa media obtenida en los experimentos con el algoritmo DDPG en función del valor de τ.

σ	$media(\overline{r})$	$std(\overline{r})$
0	-0.02234	0.01766
0.001	-0.1249	0.2125
0.01	-0.1699	0.2423

Tabla 7: Media y desviación típica de la recompensa media obtenida en los experimentos con el algoritmo DDPG en función de la desviación típica del ruido de exploración.

Tamaño de las capas ocultas	$media(\overline{r})$	$std(\overline{r})$
8	-0.2346	0.3942
16	-0.02234	0.01766
32	-0.1993	0.3712

Tabla 8: Media y desviación típica de la recompensa media obtenida en los experimentos con el algoritmo DDPG en función del tamaño de las capas ocultas de los MLP.

η	$media(\overline{r})$	$std(\overline{r})$
0.0001	-0.5250	0.6466
0.001	-0.02234	0.01766

Tabla 9: Media y desviación típica de la recompensa media obtenida en los experimentos con el algoritmo DDPG en función del valor de η.

M	$media(\overline{r})$	$std(\overline{r})$
64	-2.310	2.705
128	-2.260	2.723
256	-0.4004	0.2815
512	-0.3897	0.1345

Tabla 10: Media y desviación típica de la recompensa media obtenida en los experimentos con el algoritmo PPO en función del valor de M.

T	$media(\overline{r})$	$std(\overline{r})$
512	-0.5986	0.7261
1024	-0.006298	0.002804
2048	-0.3371	0.4295
4096	-0.3091	0.2640
8192	-0.7662	0.5359
16384	-0.4004	0.2815

Tabla 11: Media y desviación típica de la recompensa media obtenida en los experimentos con el algoritmo DDPG en función del número de experiencias utilizadas en cada iteración de entrenamiento.

N	$media(\overline{r})$	$std(\overline{r})$
4	-0.07139	0.1248
8	-0.006298	0.002804
16	-0.3453	0.4645
32	-1.171	0.4643

Tabla 12: Media y desviación típica de la recompensa media obtenida en los experimentos con el algoritmo PPO en función del número de entornos en paralelo.

Tamaño de lote	$media(\overline{r})$	$std(\overline{r})$
32	-0.6734	0.9028
64	-0.006298	0.002804
128	-0.004321	0.002354
256	-0.03232	0.04313

Tabla 13: Media y desviación típica de la recompensa media obtenida en los experimentos con el algoritmo PPO en función del tamaño de lote.

ζ	$media(\overline{r})$	$std(\overline{r})$
7	-0.005674	0.002804
10	-0.004321	0.002354
13	-0.005795	0.003978

Tabla 14: Media y desviación típica de la recompensa media obtenida en los experimentos con el algoritmo PPO en función del número de épocas de entrenamiento en cada iteración.

c_H	$media(\overline{r})$	$std(\overline{r})$
0	-0.004321	0.002354
0.001	-0.005880	0.002460
0.01	-0.1750	0.3366

Tabla 15: Media y desviación típica de la recompensa media obtenida en los experimentos con el algoritmo PPO en función de la ponderación del término de entropía en la función de pérdida.

Tamaño de las capas ocultas	$media(\overline{r})$	$std(\overline{r})$
8	-0.1422	0.2244
16	-0.004321	0.002354
32	-0.1278	0.2354
64	-1.931	0.5494

Tabla 16: Media y desviación típica de la recompensa media obtenida en los experimentos con el algoritmo PPO en función del tamaño de las capas ocultas de los MLP.

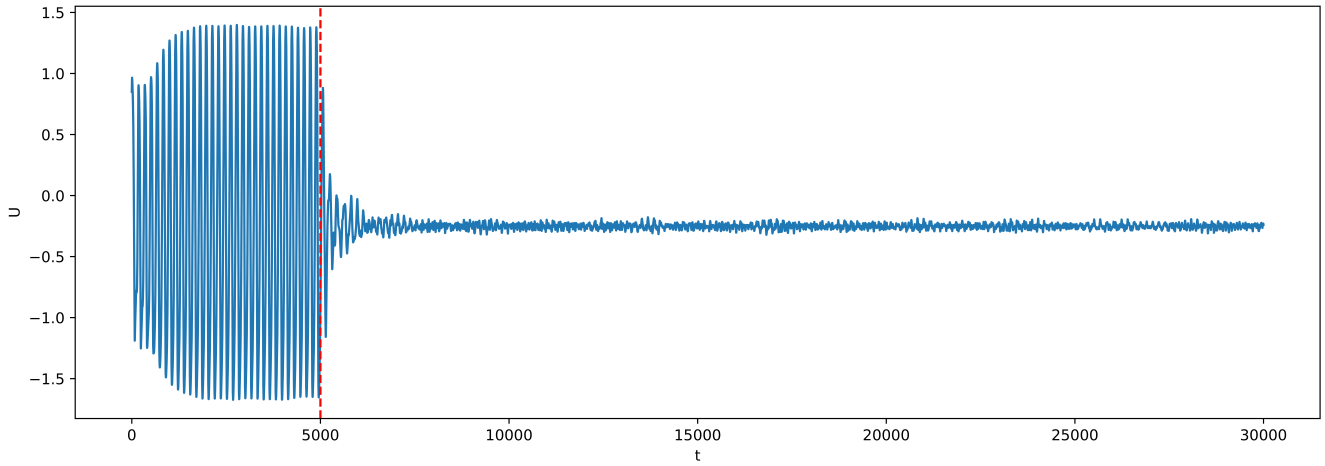

Figura 21: Supresión de la sincronización de un grupo de neuronas con factor de acoplamiento $\epsilon = 0.03$ conseguida con el algoritmo DDPG con la mejor configuración de hiperparámetros encontrada.

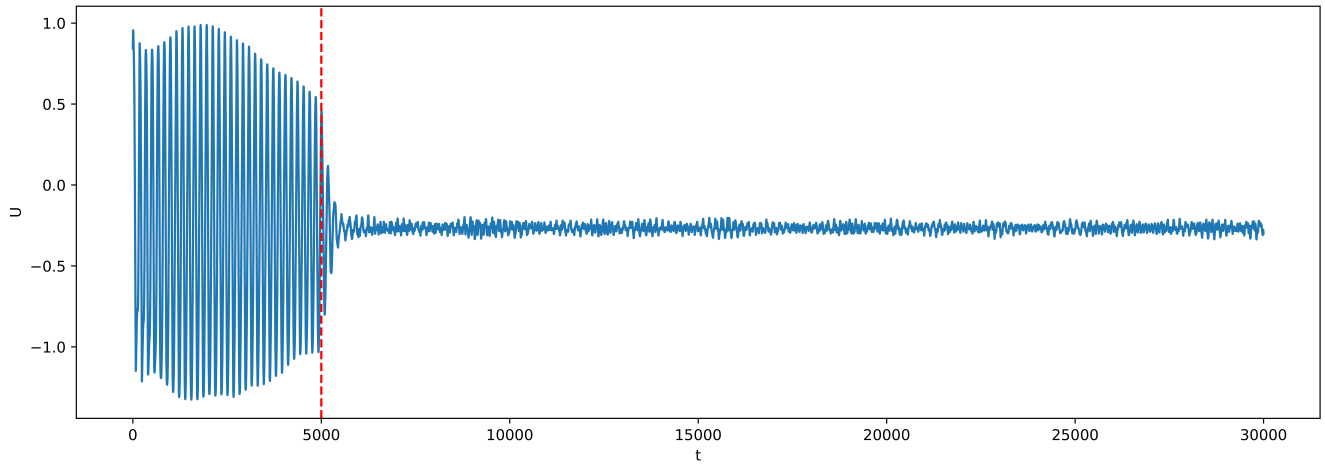

Figura 22: Supresión de la sincronización de un grupo de neuronas con factor de acoplamiento $\epsilon = 0.02$ conseguida con el algoritmo DDPG con la mejor configuración de hiperparámetros encontrada.

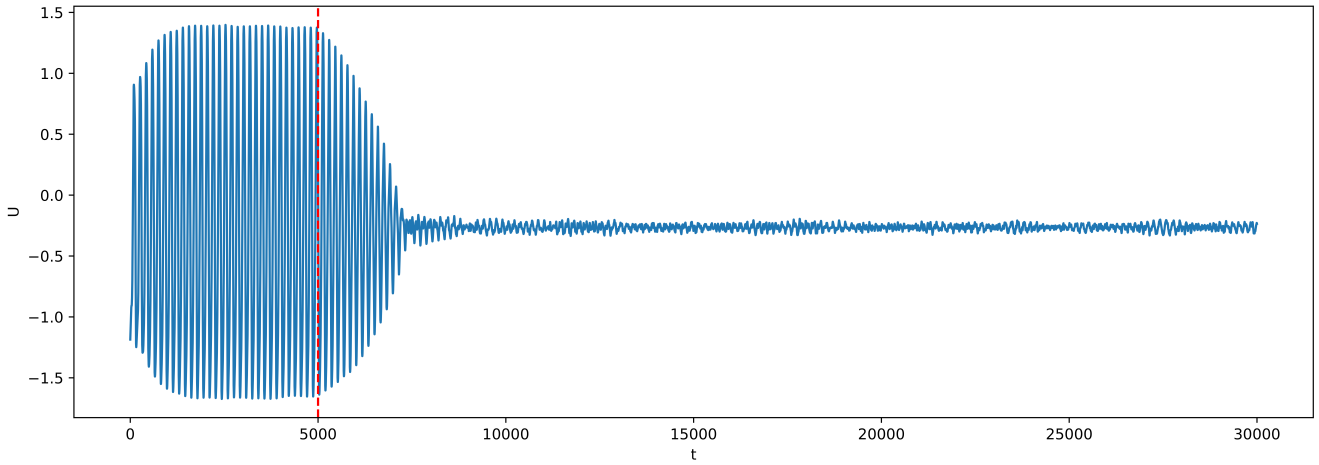

Figura 23: Supresión de la sincronización de un grupo de neuronas con factor de acoplamiento $\epsilon = 0.03$ conseguida con el algoritmo PPO con la mejor configuración de hiperparámetros encontrada.

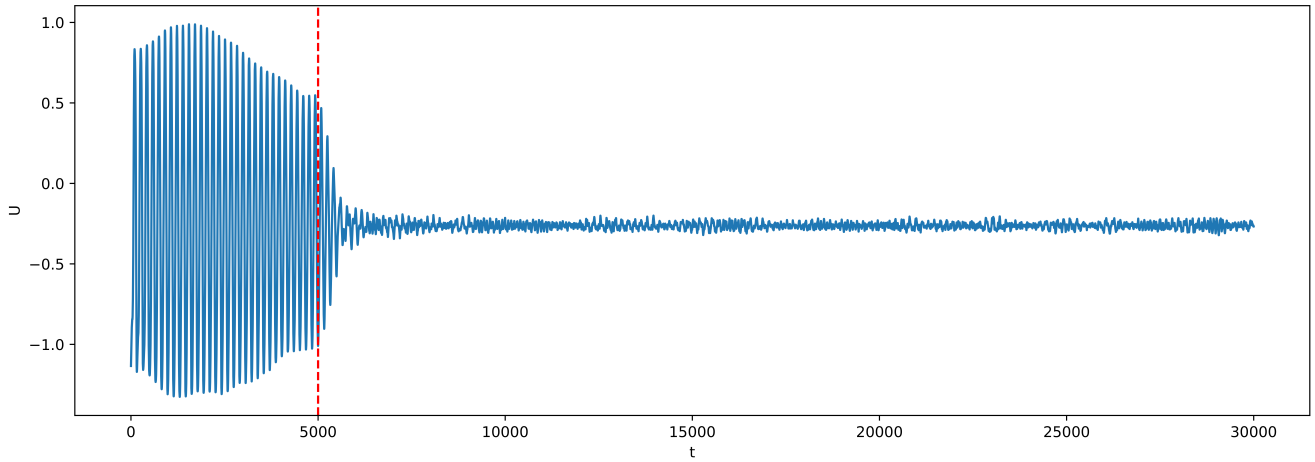

Figura 24: Supresión de la sincronización de un grupo de neuronas con factor de acoplamiento $\epsilon = 0.02$ conseguida con el algoritmo PPO con la mejor configuración de hiperparámetros encontrada.