UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

IMPLEMENTACIÓN DE ESQUEMAS DE APRENDIZAJE REFORZADO OFFLINE, PARA EL CONTROL DE PROCESOS DINÁMICOS.

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO

HERNÁN IGNACIO PADILLA CASTILLO

PROFESOR GUÍA:
MARCOS ORCHARD CONCHA

MIEMBROS DE LA COMISIÓN:
JAVIER RUIZ DEL SOLAR SAN MARTÍN
DORIS SÁEZ HUEICHAPAN

SANTIAGO DE CHILE
2023

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO
POR: HERNÁN IGNACIO PADILLA CASTILLO
FECHA: 2023
PROF. GUÍA: MARCOS ORCHARD CONCHA

IMPLEMENTACIÓN DE ESQUEMAS DE APRENDIZAJE REFORZADO OFFLINE, PARA EL CONTROL DE PROCESOS DINÁMICOS.

El aprendizaje reforzado es una rama del *Machine Learning* que ha dado que hablar en el último tiempo, por el gran éxito que ha obtenido en una amplia variedad de áreas. Sin embargo, este campo todavía cuenta con limitaciones, producto de la gran cantidad de datos y tiempo de entrenamiento requeridos para producir resultados óptimos. También, resulta difícil controlar el comportamiento del agente mientras aprende, lo que puede resultar perjudicial en contextos sensibles, como por ejemplo el desarrollo de vehículos autónomos, y la atención médica. Por esto que en el último tiempo se ha intensificado el desarrollo de algoritmos de aprendizaje reforzado *offline* que permitan evitar los riesgos de permitir que el agente aprenda de manera *online*. Estos algoritmos buscan compensar las deficiencias mencionadas transformando el área a la disciplina de los datos, el susodicho cuarto paradigma de la ciencia. Por esto resulta prometedora la investigación en esta área, a fin de encontrar nuevas maneras de expandir los logros de este fructífero campo.

El presente trabajo comprende el estudio de distintos esquemas de aprendizaje reforzado, desde los algoritmos más conocidos, desarrollados inicialmente para una implementación *online*, hasta los nuevos desarrollos bajo un paradigma *offline*. El propósito de esto es el poner a prueba las capacidades que poseen estos nuevos métodos, evaluándolos bajo un enfoque *offline*, para descubrir la utilidad y el provecho que se le puede sacar a estos.

La metodología utilizada en este trabajo, se enfoca en la implementación de los esquemas estudiados en diversas configuraciones y condiciones,para evaluar en detalle las fortalezas y debilidades de estos algoritmos, al emplearlo en el control de procesos dinámicos. De esta manera, se implementan sobre un proceso dinámico definido, distintos algoritmos de aprendizaje reforzados. Se hace en distintos tipos de configuraciones, con distintos conjuntos de datos y funciones de recompensa, a fin de evaluar los desempeños de estos, y poder generar un análisis de los distintos aspectos manipulados en la implementación.

Los resultados obtenidos en este trabajo, evidencian las capacidades y limitaciones de los algoritmos de aprendizaje reforzado estudiados, junto con el trabajo y consideraciones necesario para poder emplearlos de manera efectiva. Asimismo, estos resultados contribuyen a comprender la importancia de seguir avanzando en la investigación de nuevos algoritmos que permitan extender el alcance de estas técnicas a una mayor cantidad de campos y aplicaciones.

Agradecimientos

Quiero agradecer primero a mi familia, quienes me han apoyado incondicionalmente, pese a mis caídas y reveces, y que siempre se han esforzado por que nada me falte. En segundo lugar agradezco a mis compañeros de universidad, quienes me permitieron hacer más ameno este largo proceso de aprendizaje y desarrollo académico, y con quienes compartí tantos momentos de estudio y esparcimiento en estos años en la universidad. En tercer lugar, agradezco a los amigos que he hecho a lo largo de la vida, los que han aportado de alguna forma un granito de arena en ser quien soy. Finalmente, agradezco a la vida por los caminos que me ha llevado, que pese a los embates que me ha planteado a lo largo de ella, me ha entregado hermosos momentos, junto con la fortaleza para crecer y salir adelante, lo que me ha permitido llegar a este punto.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Capítulo 1

Introducción

1.1. Contexto y justificación

El aprendizaje reforzado [1] puede considerarse como el área que integra los campos de la teoría de control y el *Machine learning*. Este, esencialmente, es una rama del *Machine Learning* que lidia con como aprender estrategias de control que interactúen con ambientes complejos. En este sentido es un sistema para aprender como interactuar con el ambiente a partir de la experiencia. Esta idea esta inspirada biológicamente por como aprenden los animales, basándose en la psicología conductista que, a través de prueba y error, obtienen nociones fundamentales que rigen el ambiente.

Esta área del *Machine Learning* es considerada actualmente como una de las más prometedoras, debido al alto desempeño, en ciertos casos sobre-humano, que ha logrado en diversos dominios de implementación. Algunos de estos ejemplos más conocidos son, el caso AlphaGo [2], TD-Gammon [3], DQN en Atari [4], aplicaciones en robótica [5][6], entre otros. Debido a esto, existe gran interés por extender su uso a más áreas, pero el principal obstáculo para esto es el paradigma, principalmente *online*, que posee. Por esta razón, el aprendizaje reforzado *offline* representa una alternativa para poder desarrollar poderosos motores de decisión y generalización que, como mencionan Levine et al. [7], podrían aplicarse en áreas donde una metodología *online* resulta costosa y/o peligrosa, como las áreas de la salud, educación, robótica, vehículos autónomos, entre otras.

Estas posibles aplicaciones motivan el estudio de los distintos esquemas de aprendizaje reforzado *offline*, puesto que las últimas investigaciones y desarrollos ofrecen la posibilidad de orientar esta área del *Machine Learning* en una disciplina orientada a los datos [7]. Debido a esto, es que se plantea el desarrollo de este trabajo exploratorio, con el fin de evaluar las capacidades de estos esquemas de aprendizaje reforzado, junto con adquirir un entendimiento de las condiciones y exigencias necesarias para implementar adecuadamente estos esquemas.

1.2. Objetivos

1.2.1. Objetivo general

Evaluar la capacidad de los esquemas de Aprendizaje Reforzado *offline* mediante el desarrollo de un controlador para un proceso dinámico no lineal, en ausencia de un modelo exacto del proceso.

1.2.2. Objetivos específicos

1. Explorar distintos esquemas actuales de aprendizaje reforzado *offline* y su implementación mediante redes profundas.

2. Selección de un proceso sobre el cual se realizará el estudio del desarrollo de un controlador, junto con la búsqueda o desarrollo de su base de datos a implementar.

3. Codificar alguno de los esquemas de aprendizaje reforzado estableciendo y definiendo distintos aspectos de su implementación.

4. Desarrollar y evaluar un controlador para el proceso dinámico seleccionado empleando aprendizaje reforzado *offline*.

Capítulo 2

Marco teórico y estado del arte

El presente documento se sitúa en el estudio del aprendizaje reforzado fuera de línea, que corresponde a una nueva rama de algoritmos desarrollados en torno al aprendizaje reforzado, área perteneciente al creciente campo del control orientado por datos. Este campo puede resumirse en la integración de métodos matemáticos modernos y el aprendizaje de máquinas para el control de procesos dinámicos [8].

El enfoque en el estudio de algoritmos fuera de línea se plantea en contraste al paradigma inicialmente *online* con el que comenzó el aprendizaje reforzado, puesto que este en su formulación original planteaba el desarrollo de agentes que a través de la exploración e interacción con el entorno, aprendían a comportarse óptimamente y cumplir un objetivo propuesto. El cambio de este paradigma se encuentra motivado por el deseo de extender el éxito obtenido por esta área en las últimas décadas, a nuevos campos en los que su planteamiento inicialmente *online* limita su aplicación.

2.1. Control de sistemas

Esta es una rama de la ingeniería que se encarga de diseñar, analizar y optimizar sistemas que interactúan con el mundo físico. El objetivo del control de sistemas es lograr que un sistema funcione de manera adecuada y estable, adaptándose a cambios en el entorno y cumpliendo con ciertos requisitos o objetivos. Este se basa en las matemáticas y la teoría de control para analizar, modelar y optimizar el funcionamiento de sistemas físicos. La teoría de control es una rama de las matemáticas que se encarga de estudiar las leyes que gobiernan el comportamiento de sistemas dinámicos y cómo influir en ellos para conseguir un comportamiento deseado.

2.1.1. Control PID

Esta técnica de control se construye a partir de un mecanismo de lazo cerrado, que a través de la retroalimentación de la salida, busca estimar el error $e(t)$ entre una variable medida del sistema y un punto deseado para ella. Como se puede apreciar en la Figura 2.1, este método se conforma de 3 elementos. Siendo P el primero, corresponde a un elemento proporcional a un error en el instante t, el segundo I, un elemento proporcional a la acumulación de los errores pasados hasta el instante t, y el tercero D, un proporcional a la tasa de cambio o velocidad con que está creciendo o disminuyendo el error en el instante t. Como lo indican Tehrani y Mpanda: “El control PID puede ser entendido como un control que toma el presente, el

pasado y el futuro del error en consideración" [9].

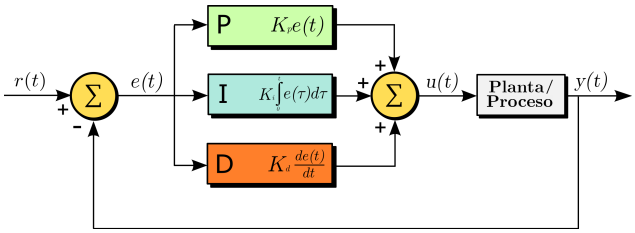

Figura 2.1: Diagrama de bloques de un controlador PID. Fuente: Adaptado de [10]

2.1.2. Regulador lineal cuadrático

El regulador lineal cuadrático, también conocido como LQR, es un método perteneciente al control óptimo, área de las matemáticas en la que se busca controlar sistemas dinámicos planteados mediante la resolución de un problema de optimización. En este problema de optimización, las restricciones sobre las variables vienen dadas por ecuaciones diferenciales lineales que representan la dinámica del sistema, mientras que la función de costo viene dada por una función cuadrática que suma los errores de distintas variables del sistema a controlar con respecto a los valores deseados para estas. En la Figura 2.2 se presenta el diagrama de lazo cerrado del controlador, donde se aprecian las ecuaciones de las dinámicas del sistema correspondientes a $\dot{x}$, y la función de costo dada por y.

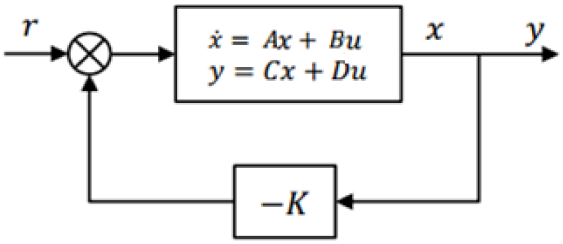

Figura 2.2: Diagrama de bloques de un controlador LQR. Fuente: [11]

2.1.3. Control predictivo por modelo

Esta es una técnica de control que se basa en el uso de un modelo matemático del proceso a controlar. El objetivo del control predictivo por modelo, también llamado MPC, es predecir cómo se comportará el proceso en el futuro y utilizar esta información para realizar ajustes en tiempo real para mantener el proceso en un estado deseado. Para implementar el control predictivo por modelo, es necesario construir un modelo matemático de las dinámicas del proceso que se desea controlar. Este modelo puede ser basado en ecuaciones físicas que describen el comportamiento del proceso, o puede ser un modelo basado en datos que ha sido entrenado a partir de datos históricos del proceso. Una vez que se tiene el modelo, se utiliza para predecir cómo se comportará el proceso en el futuro y se comparan estas predicciones con los valores deseados. Si se detecta una desviación entre el comportamiento predicho y los valores deseados, se realizan ajustes en los parámetros de control para tratar de mantener el proceso en un estado deseado. En la Figura 2.3 se tiene una representación del sistema de lazo cerrado del control MPC.

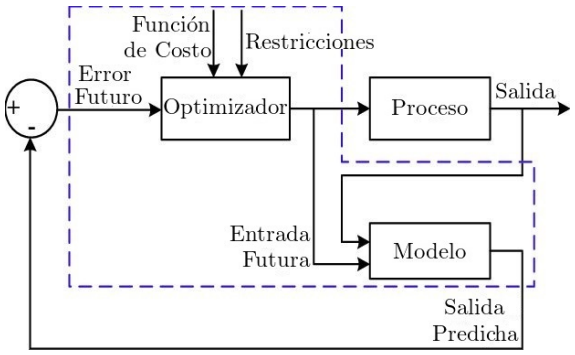

Figura 2.3: Diagrama de bloques de un controlador MPC. Fuente: Adaptado de [12]

2.2. Aprendizaje reforzado

El aprendizaje reforzado es un área del aprendizaje máquina, inspirado en la psicología conductista, la que estudia como agentes pueden aprender a través de la prueba y error, es decir, trata de replicar como los humanos aprendemos de nuestro entorno interactuando y interviniendo en el, sin la necesidad de poseer conocimientos previos al respecto. De esta manera, se busca desarrollar agentes, que sin decirles ni enseñarles explícitamente como deben actuar frente a un problema, sean capaces de descubrir como comportarse de forma óptima, al identificar que acciones son las mejores o que generan una mayor recompensa mediante la experimentación en el entorno. Por esta razón, esta área se define como un tercer paradigma del aprendizaje máquina, distinto al aprendizaje supervisado y no supervisado, puesto que en el primero a las máquinas se les enseña explícitamente como deben actuar, y en el segundo se busca que la máquina aprenda a identificar patrones o extraer información de los datos, sin definirle un objetivo.

2.2.1. Proceso de decisión de Markov

El aprendizaje reforzado, en un sentido general, aborda el problema de aprender a controlar un sistema dinámico. En este contexto, el problema del sistema dinámico se formula a través de un proceso de decisión de Markov (MDP), definido por la tupla $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, r)$, en donde $\mathcal{S}$ y $\mathcal{A}$ definen los espacios de estado y acción, $\mathcal{T}$ una distribución de probabilidad con respecto a la dinámica del sistema representada por $\mathcal{T}(s_t + 1|s_t, a_t)$ y $r(s_t|a_t)$ una función de recompensa. El objetivo principal en el aprendizaje reforzado es aprender una *policy* o política que define una distribución de probabilidad sobre las distintas acciones que se pueden tomar para cierto estado del sistema, conociéndose esta como $\pi(a_t|s_t)$. A partir de esta distribución π se tiene $p(s_{t+1}|s_t, a_t)$ la probabilidad de transicionar al estado s_{t+1} al tomar la acción a_t estando en el estado s_t. Entonces el resultado del proceso secuencial de toma de acciones realizada por el agente puede ser visualizada a partir de el modelo gráfico presentado en la Figura 2.4[13], en donde se debe tener en cuenta que la probabilidad de transición $p(s_{t+1}|s_t, a_t)$ es desconocida para el agente.

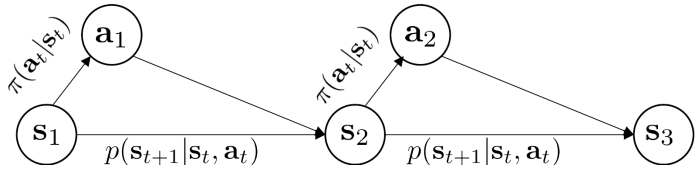

Figura 2.4: Proceso secuencial de interacción del agente con el ambiente. Fuente: [13]

Si la trayectoria τ es una secuencia de estados y acciones de largo T, dada por todas las acciones y estados ocurridas desde un tiempo $t = 1$ hasta T, la distribución de probabilidad p_π sobre esta trayectoria viene dada por la Ecuación 2.1

$$p_\pi(\tau) = p(s_1) \prod_{t=1}^{T} \pi(a_t|s_t)\mathcal{T}(s_{t+1}|s_t, a_t). \tag{2.1}$$

La función objetivo del aprendizaje reforzado es $J(\pi)$ de la Ecuación 2.2, que correspondería a la esperanza sobre la trayectoria de la distribución mencionada anteriormente, en donde se busca maximizar la recompensa final obtenida. En esta ecuación γ representa un factor de descuento, que trata de reflejar los efectos del tiempo en esta ganancia que se busca maximizar.

$$J(\pi) = \mathbb{E}_{\tau \sim p_\pi(\tau)} \left[\sum_{t=1}^{T} \gamma^t r(s_t, a_t) \right] \tag{2.2}$$

De esta manera, los modelos de aprendizaje reforzado buscan mejorar estas *policies* o políticas de decisión, para maximizar la recompensa. En este sentido, a mayor recompensa que obtenga mi modelo, mejor esta cumpliendo controlar el sistema dinámico. Otros conceptos importantes que se desprenden a partir de esta función objetivo definida, son las nociones de funciones de valor. En primer lugar tenemos la función de valor de estado V^π dada por la Ecuación 2.3, que corresponde a la recompensa total esperada si partiendo en un estado s_t se siguiera una política π.

$$V^\pi(s_t) = \mathbb{E}_\pi \left[\sum_{t'=T}^{\infty} \gamma^{t'} r(s_{t'}, a_{t'}|, s_t, a_t) \right] \tag{2.3}$$

En segundo lugar tenemos la función de acción-valor Q^π dada por la expresión 2.4, que corresponde a la recompensa total esperada si tras tomar una acción a_t en un estado s_t se sigue una política π.

$$Q^\pi(s_t, a_t) = \mathbb{E}_\pi \left[\sum_{t'=T}^{\infty} \gamma^{t'} r(s_{t'}, a_{t'}|s_t) \right] \tag{2.4}$$

En tercer lugar tenemos la función de ventaja dada por la Ecuación 2.5. Esta resta el valor del estado V, que mide que tan bueno es el estado actual en el que nos encontramos, a nuestra función de acción-valor Q, que mide el valor de tomar una acción en particular, en un estado respectivo. A partir de esto podemos entender que la función de ventaja A mide que tan buena es una acción, con respecto al resto de las posibles en el estado actual.

$$Q(s_t, a_t) = V(s_t) + A(s_t, a_t). \tag{2.5}$$

2.2.2. Tipos de algoritmos de aprendizaje reforzado

Existen principalmente 3 criterios para clasificar los distintos algoritmos de aprendizaje reforzado: el empleo de modelos, el como se formula la optimización y el como se lleva a cabo el proceso de optimización.

La primera categoría distingue si los métodos emplean o no un modelo del entorno en el que el agente opera. Para los métodos basados en modelo, también llamados *model-based*, se plantea la existencia de un modelo que es capaz de representar las dinámicas del entorno. En estos métodos el agente tiene acceso a información del entorno, o bien, tratan de entenderlo creando un modelo aproximado que lo represente. Estos métodos emplean programación dinámica, para encontrar las mejores acciones a tomar a partir de los modelos que van generando. Por otro lado, se distinguen también los métodos libres de modelo, conocidos como *model-free*, los que no buscan ni tienen interés por conocer la dinámica del entorno que los rodea, ya que solo les interesa conocer la consecuencia de sus actos al interactuar con el. Las ventajas de los métodos basados en modelos, es que pueden resultar ser más eficientes y rápidos en aprender una buena política, pero esto requiere tener más conocimientos del problema al momento de implementar estos métodos, por lo que los métodos libres de modelo resultan ser más fáciles de implementar.

En la segunda categoría, los métodos se separan si es que estos aproximan directamente la política $\pi(s, a)$, conocidos como *policy-based*, o si es que estos optimizan la función de acción-valor $Q(s, a)$, conocidos como *value-based*. Los primeros, basados en optimizar la política trabajan directamente sobre la función objetivo $J(\pi)$, para lo que requieren parametrizar esta función mediante alguna aproximación lineal, o el empleo de una red neuronal. Este tipo de métodos aprende directamente la política estocástica que dictamina que acción tomar según el estado. Los segundos, optimizan las funciones de valor presentadas en la Subsección 2.2.1, mediante métodos iterativos que emplean distintas ecuaciones de programación dinámica. En estos últimos tipos de métodos la política es representada de manera implícita, puesto que emplean estas funciones de valor obtenidas para determinar y escoger cual es la mejor acción a tomar en un estado. Pese a la diferencia entre estos tipos de métodos, se han desarrollado algoritmos que logran integrarlos con el fin de aprovechar las fortalezas de de cada uno, a los que se les conoce como *actor-critic*.

La tercera categoría distingue si la política es actualizada directamente en base a las acciones que esta toma, lo que se conoce como *on-policy*, o si actualiza su política independientemente de los actos que está tomando en el momento, llamado *off-policy*. Los métodos *on-policy* trabajan optimizando su política de acción en base a las acciones que están tomando, buscando corregir o potenciar inmediatamente sus decisiones de acción tomadas. Por otro lado, los métodos *off-policy* pueden emplear una política de decisión para explorar el entorno y obtener datos, mientras que optimizan otra política de forma paralela, empleando estos datos recabados, o incluso empleando datos generados por otros agentes.

El trabajo desarrollado en este informe explorará principalmente algoritmos de aprendizaje reforzado del tipo *model-free*, *off-policy* y *valued-based*, no obstante explorará también algunos métodos que emplean la formulación de optimización actor-critic.

2.2.2.1. Q-Learning

Este método derivado del método estadistico *Q-value*, fue presentado el año 1989 [14] y es conocido como el clásico ejemplo de un algoritmo *model-free* y *value-based*, el cual resuelve del problema del aprendizaje reforzado enseñándole al agente que acción es mejor tomar para cada uno de los estados posibles del sistema. Para esto, el algoritmo actualiza una función de acción-valor $Q(s, a)$ mediante la interacción con el ambiente, empleando aprendizaje de diferencia temporal, también conocido como *TD-learning*. El *TD-Learning* consiste en predecir el valor futuro de una variable en una secuencia de estado, calculándolo a partir de la combinación de la recompensa inmediata y su propia predicción de recompensa en el momento siguiente. Luego, al avanzar al siguiente estado y recibir nueva información, comparará la predicción actual con la realizada anteriormente, donde este algoritmo calculará la diferencia entre estas predicciones y utilizará esta diferencia temporal para ajustar la antigua predicción a la nueva. Ahora, transfiriendo este tipo de aprendizaje al *Q-Learning*, la función de acción-valor $Q(s, a)$ que se busca optimizar vendrá dada por Ecuación 2.6

$$Q(s_t, a_t) = r_t + \gamma \max_a Q(s_{t+1}, a), \tag{2.6}$$

expresión que se le conoce como ecuación de Bellman. Luego, siguiendo con el proceso de aprendizaje basado en *TD-Learning*, la actualización de los valores Q se realizará empleando la Ecuación 2.7

$$Q^{new}(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left(r_t + \gamma \max_a Q(s_{t+1}, a) - Q_t(s_t, a_t) \right). \tag{2.7}$$

Luego de terminado el proceso de entrenamiento, y encontrar nuestra política del sistema $Q^{\pi}(s_t, a_t)$, el agente entrenado escogerá la mejor acción a tomar mediante la Ecuación 2.8

$$Q^*(s_t, a_t) = \max_{\pi} Q^{\pi}(s_t, a_t). \tag{2.8}$$

Como la función $Q^{\pi}(s_t, a_t)$ entrega un valor para todos los pares acción-valor del sistema, este algoritmo es representado por una tabla llamada $Q-table$, que tendrá dimensiones $S \times A$, con S y A el espacio de estados y acciones posibles. Este algoritmo tabular, ha demostrado tener éxito cuando tanto el espacio de estado como el de acción se encuentran relativamente acotados, o no son de gran tamaño, siendo una solución bastante eficiente para estos casos. Sin embargo, se debe distinguir que para casos donde no se cumplen estas condiciones, o que también no se cuenta con una exploración adecuada de todos los pares estado-acción, el método resulta infructuoso o poco práctico, pues se le dificulta lograr converger a una buena política de acción. Esto porque el método no cuenta con grandes capacidades de generalización, o resulta demasiado costoso ser capaz de encontrar los valores Q para todos los pares estado-acción, siendo necesario optar por otro tipo de algoritmos.

2.2.2.2. Actor-Critic

Basado en *Q-Learning*, se diferencia de el en que optimiza tanto una función de política, como una función de valor Q, como se mencionó en 2.2.2. Para esto, el algoritmo emplea dos componentes distintos: un actor y un crítico [15]. El actor es responsable de seleccionar la acción a tomar en un estado determinado, mientras que el crítico es responsable de evaluar la acción seleccionada y actualizar la función Q. El actor mencionado se entrena para aprender una política de toma de decisiones $\pi(a|s;\theta)$ óptima a partir de la experiencia y las recompensas obtenidas en el entorno. Esta políticas se actualiza minimizando la diferencia entre la

acción seleccionada y la acción óptima mediante la Ecuación 2.9

$$\theta = \theta - \alpha \nabla log \pi(a_t|s_t;\theta) \cdot Q(s_t, a_t), \tag{2.9}$$

donde θ son los parámetros de la función que modela la política π. Por otro lado, el crítico se encarga de evaluar la acción seleccionada y actualizar una función Q, la que se diferencia de la trabajada en *Q-Learning*, puesto que viene dada por la expresión de la Ecuación 2.10

$$Q(s_t, a_t) = r_t - \gamma V(s_{t+1}; w), \tag{2.10}$$

donde w son los parámetros de la función de valor optimizada mediante diferencia temporal, como en el *Q-Learning* convencional.

2.3. Aprendizaje reforzado profundo

Como se mencionó anteriormente, los métodos como el *Q-Learning* poseen dificultades cuando se trabaja con espacios de estado y acción muy grandes o continuos, puesto que el método no resulta escalable y termina fallando en lograr aproximar adecuadamente una buena política. Por esta razón es que resulta útil trabajar con redes neuronales, puesto que estas se efectivas como aproximadores de funciones tanto lineales como no lineales. Las redes neuronales son una herramienta valiosa en el aprendizaje reforzado profundo debido a su capacidad para procesar grandes cantidades de datos, aprender de manera no supervisada y generalizar de manera efectiva. Esto permite que el aprendizaje reforzado profundo maneje problemas más complejos y aborde problemas que el aprendizaje reforzado convencional no puede resolver, dando paso a la introducción de nuevos algoritmos más capaces.

2.3.1. Deep Q Learning

Este algoritmo presentado el año 2013 [4], representó una revolución en su momento, puesto que introdujo por primera vez un método libre de modelos que pudiera ser implementado en entornos con espacios de estado y/o acción complejos, o muy grandes. El *Deep Q-Learning*, también llamado DQL, se basa en el *Q-Learning* convencional, solo que este emplea redes neuronales para aprender la función Q y tomar decisiones óptimas en el entorno. Esto permite que el algoritmo aprenda a partir de un gran número de estados y acciones y se adapte a entornos complejos y no estructurados. Las ecuaciones que definen este algoritmo son idénticas a las empleadas en el *Q-Learning*, solo que en este caso, se actualiza la función acción-valor mediante una red Q o *Q-Network*, y un algoritmo de optimización como el descenso de gradiente. La red neuronal se entrena para minimizar la función de pérdida del valor esperado, que mide la diferencia entre el valor esperado de una acción en un estado determinado y el valor real obtenido tras tomar la acción.

2.3.1.1. Double Deep Q Learning

Propuesto por Google DeepMind en el año 2015 [16], este algoritmo es una adaptación del *Deep Q Learning* original, el cual era conocido de sufrir problemas de sobre estimación de los valores de la función $Q(a, s)$, bajo ciertas circunstancias. Inspirado en el *Double Q-Learning* [17], esta modificación al DQL, propone separar la responsabilidad de evaluar la política avara de acción, de la estimación de los valores, asignándoles esta tarea a dos *Q-Network* distintas. De esta manera se trabaja con una red principal con parámetros θ, encargada de seleccionar

la mejor acción posible, y con una red objetivo de parámetros θ', encargada de evaluar la acción tomada. Buscando alterar lo menor posible el DQL original, la actualización de los parámetros θ de la red principal se realiza de idéntica manera, solo que para esta instancia la ecuación de Bellman es modificada mediante la expresión de la Ecuación 2.11

$$Q(s_t, a; \theta) = r(s_t, a_t) + \gamma Q\left(s_{t+1}, \underset{a}{\text{argmax}} Q(s_{t+1}, a; \theta); \theta'\right), \tag{2.11}$$

donde es posible apreciar que los parámetros de la red objetivo son usados para estimar los valores de estado para s_{t+1}. En cuanto a la actualización de los parámetros θ' de la red objetivo, este se realiza de manera periódica copiándolos de la red principal, buscando mantener estáticos estos parámetros por este periodo de espera, lo que permitiría reducir las sobre estimaciones experimentadas por la red principal.

2.3.1.2. Duelind Deep Q Learning

Este algoritmo presentado, de igual manera que el anterior, por Google DeepMind en el año 2015 [18], propone emplear un cambio en la arquitectura de las redes neuronales empleadas en *Deep Q Learning*. Este cambio de arquitectura consiste en separar los valores de la función $Q(a, s)$ en dos estimadores distintos, consistentes de la función de valor $V(s)$ y la función de ventaja $A(s, a)$. Recordamos que esta función de ventaja fue presentada anteriormente en la Ecuación 2.5. Este cambio en la arquitectura se propone mediante el empleo de dos capas neuronales paralelas, asignándole a una el trabajo de calcular V y a la otra el de calcular A. No obstante, la implementación directa de la función Q presentado en la Ecuación 2.5, no es posible, ya que durante el proceso de entrenamiento de la red, resultan inidentificables los aportes correspondientes de V y A al cálculo de Q. Por esto se propone alterar el cálculo de Q de la forma presentada en la Ecuación 2.12

$$Q(s_t, a_t) = V(s_t) + (A(s_t, a_t) - \frac{1}{|\mathcal{A}|} \sum_{a'} A(s_t, a)), \tag{2.12}$$

donde $\mathcal{A}$ viene dado por la dimensionalidad del espacio de acción. Este cambio propuesto en la arquitectura busca ayudar en el proceso de generalización del agente, pues plantea facilitarle el identificar de mejor manera, si un posible estado futuro es peor con respecto al resto. En la Figura 2.5 se presenta el diagrama desarrollado por DeepMind, que permite visualizar las diferencias entre la arquitectura de *Q-Network* y la nueva arquitectura de red de duelo, o *Dueling Q-Network*, propuesta.

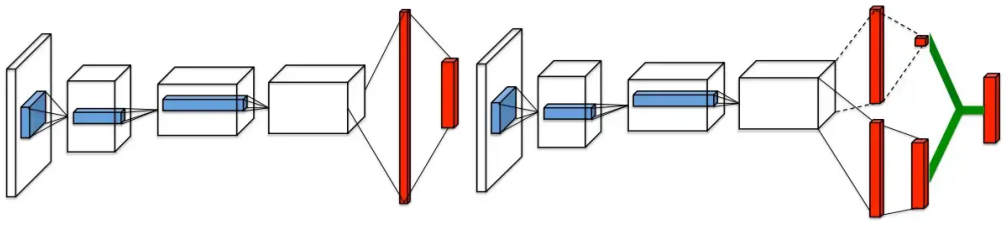

Figura 2.5: Una red neuronal Q clásica (izquierda) y una red neuronal de duelo (derecha). La red neuronal de duelo implementa las dos capas paralelas, que son combinadas mediante la Ecuación 2.12. Fuente: Adaptado de [18]

2.3.1.3. Deep Q Learning from Demostration

Conocido también como DQfD [19], es una variante de *Deep Q-Learning*. A diferencia del algoritmo original, que solo utiliza recompensas para guiar el aprendizaje del agente, DQfD utiliza tanto recompensas como demostraciones de expertos para guiar el aprendizaje. Las demostraciones de expertos son soluciones óptimas del problema proporcionadas al agente durante el aprendizaje. Estas demostraciones se utilizan para proporcionar una "pista" al agente sobre cómo resolver el problema, lo que le permite aprender más rápidamente y a menudo llegar a una solución más óptima. Para emplear estas demostraciones expertas, se requieren unos ajustes en la función de pérdida empleada para actualizar la función Q. Por ejemplo, si una demostración de expertos muestra que tomar una determinada acción en un estado específico es óptimo, entonces el valor objetivo para esa acción en ese estado se ajustará para reflejar esto. Para calcular este valor objetivo ajustado utilizando las demostraciones de expertos, primero se selecciona una demostración de expertos y se determina la acción óptima tomada en cada estado de la demostración. Luego, se asigna a cada acción óptima tomada en cada estado un valor objetivo de 1 y se asigna a todas las demás acciones un valor objetivo de 0. Adicionalmente se utiliza una técnica de regularización llamada "regularización L2" para evitar el sobre ajuste de la red neuronal. La regularización $L2$ se utiliza para agregar un término de penalización a la función de pérdida que mide la complejidad de la red neuronal y se ajusta para evitar que la red tenga un rendimiento peor en el conjunto de datos de prueba. Con estos ajustes, la función de pérdida utilizada para optimizar la red neuronal corresponde a la Ecuación 2.13

$$\begin{aligned} J(Q) = &\left(r_t + \gamma \max_a \left[Q(s_{t+1}, a)\right] - Q(s, a)\right)^2 \\ &+ \lambda_E \cdot \left(\max_a \left[Q(s_t, a) + l(a_E, a)\right] - Q(s_t, a_E)\right) + \lambda_{L2} \cdot \sum \theta^2. \end{aligned} \tag{2.13}$$

En la Ecuación 2.13 $l(a_E, a)$ es el valor objetivo ajustado utilizando las demostraciones de expertos para la acción a_E en el estado s_t, λ_E controla la importancia relativa de las demostraciones de expertos, λ_{L2} controla la intensidad de la regularización, θ son los pesos de la red neuronal y $\sum \theta^2$ es la suma de los cuadrados de los pesos de la red neuronal.

2.3.1.4. Neural Fitted Q Iteration

Este método también conocido como NFQ, es una versión actualizada del algoritmo *Fitted Q iteration* [20], el cual fue desarrollado con la intención de superar las desventajas que poseía el *Q-Learning* al trabajar con espacios de estado y/o acción continuos, o discretos muy grandes. Este método previo buscaba implementar algoritmos basados en árboles, de tal manera de entrenar múltiples funciones Q, que convergerían a una mejor solución que en el caso del *Q-Learning* estandar. En cuanto a esta nueva versión, que como su nombre lo indica emplea redes neuronales y que se le conoce también como NFQ [21], opera de manera un tanto similar al *Deep Q-Learning*. En este algoritmo la red también es entrenada empleando la ecuación de Bellman para actualizar los parámetros de Q, salvo que en este caso la red recibe de input tanto el estado como la acción para la que se desea calcular su valor Q. El método opera de esta manera porque trata de encontrar el valor Q óptimo para cada estado s y acción a, lo que significa encontrar la acción que maximizará la recompensa esperada a largo plazo.

2.3.2. Soft Actor Critic

Soft Actor-critic, también llamado SAC, se encuentra basado en el algoritmo *actor-critic*, que es una técnica de aprendizaje por refuerzo que combina dos componentes: un actor y un crítico. El actor toma decisiones en el entorno y el crítico evalúa la calidad de esas decisiones. A diferencia del *actor-critic* original que utiliza una política determinista, SAC utiliza una política suave (*soft*) que es una distribución de probabilidad sobre las acciones. Esto permite una mayor flexibilidad en la toma de decisiones y puede mejorar la estabilidad y el rendimiento del algoritmo. La política suave se calcula mediante la Ecuación 2.14

$$\pi(a_t|s_t) = \frac{\exp(Q(s_t, a_t)/\alpha)}{\sum_a \exp(Q(s_t, a)/\alpha)}, \tag{2.14}$$

con α un parámetro de temperatura. Este método también se distingue porque su función objetivo cuenta con una componente entrópica, que se utiliza para maximizar la diversidad en las decisiones tomadas por el actor. Esta componente se puede calcular utilizando la Ecuación 2.15

$$\mathcal{H}(\pi) = -\sum_a \pi(a|s_t) \log \pi(a|s_t), \tag{2.15}$$

donde $\mathcal{H}$ es la entropía y π es la política suave. Luego, la función objetivo completa de la política, vendrá dada por la Ecuación 2.16

$$J(\pi) = \mathbb{E}_{\tau \sim p_\pi(\tau)} \left[\sum_{t=1}^{T} \gamma^t r(s_t, a_t) + \alpha \mathcal{H}(\pi(\cdot|s_t) \right]. \tag{2.16}$$

2.4. Aprendizaje reforzado fuera de línea

Como se mencionó previamente en la Subsección 2.2.2, se distinguen dos tipos de aprendizaje reforzado según el como se haga la actualización de política. En el primero, conocido como *on-policy* o *online reinforcement learning* (a), las políticas de decisión son actualizadas a través de la información que adquiere al interactuar de forma in situ sobre el sistema dinámico que se quiere controlar. O sea que la política implementa la última información obtenida de su última interacción con el medio. En el segundo tipo, se tiene el aprendizaje reforzado *off-policy* (b), en el cual esta información que adquiere in situ se va almacenando en un *buffer* o espacio de memoria haciendo una actualización de las políticas de decisión de manera más esporádica, lo que lo hace un proceso más eficiente, por reducir el tiempo de cómputo que se gasta al estar actualizando las políticas constantemente. En la Figura 2.6 [7] se puede apreciar una representación gráfica de estos tipos de aprendizaje reforzado.

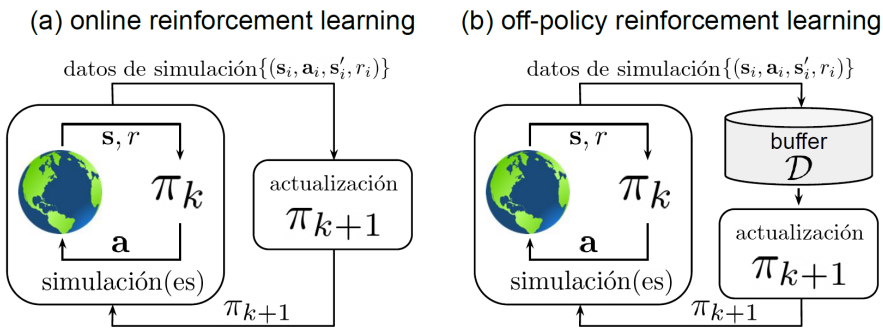

Figura 2.6: Tipos de aprendizaje reforzado *online*
Fuente: Adaptado de [7]

A diferencia de los dos tipos de aprendizaje mencionados anteriormente, el aprendizaje reforzado *offline* (c) se emplea una sola vez las bases de datos existentes o previamente adquiridas, con la que se entrena el modelo para extraer una política de acción cualquiera, que generalmente se desconoce. En este sentido, la política de acción se obtiene solo al final, cuando se es entrenado completamente el modelo, y se obtiene la mejor política extraíble de los datos. Se tiene una representación gráfica del aprendizaje reforzado *offline* en la Figura 2.7 [7].

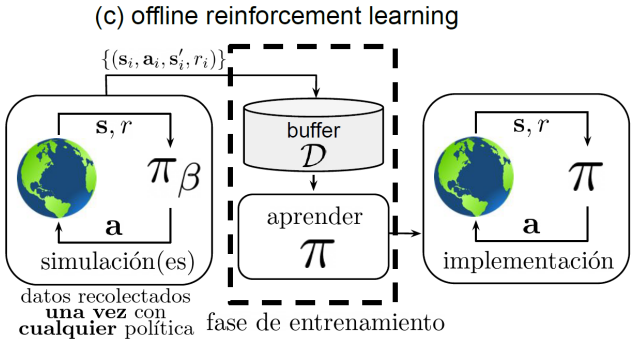

Figura 2.7: Aprendizaje reforzado *offline*. Fuente: Adaptado de [7]

Llamaremos formalmente a $\mathcal{D} = \{(s_i, a_i, s'_i, r_i)\}$ las tuplas que conforman el *dataset* disponible para el agente para que aprenda sus políticas. Se asume que los estados s en este conjunto de datos poseen una distribución desconocida $s \sim d^{\pi_\beta}(s)$, que a la vez es inducida por la distribución de una política desconocida $a \sim \pi_\beta(a|s)$. Con respecto a los estados futuros, están dados por las correctas dinámicas de probabilidad del sistema, que también son desconocidas, pero que se asumen iguales, mientras que la recompensa $r \leftarrow r(s|a)$ viene dada por la función de recompensa también desconocida. En el caso del aprendizaje reforzado *offline*, aún se tiene el objetivo de encontrar la política de acción que maximice $J(\pi)$ una vez que se emplee el agente en el ambiente. A diferencia del aprendizaje reforzado *online*, el aprendizaje reforzado *offline* no permite utilizar al agente en el ambiente durante el entrenamiento, lo que hace que las distribuciones relevantes sean desconocidas y dificulta su implementación. Además, existen otros problemas que pueden afectar la adaptabilidad y generalización de los agentes en comparación con el aprendizaje reforzado *online*. Estos problemas están relacionados con la estacionariedad de la distribución de los datos de entrenamiento y la ausencia de retroalimentación en tiempo real del entorno durante el entrenamiento. Uno de los principales problemas es el cambio de distribución, que ocurre cuando la distribución del

conjunto de datos de entrenamiento difiere significativamente de la distribución del entorno real en el que se utilizará el agente. Otro problema común es la sobreestimación del valor de las acciones del agente en el conjunto de datos de entrenamiento. Estos problemas pueden afectar la capacidad del agente para tomar decisiones óptimas en situaciones nuevas y no vistas. Además de los problemas de cambio de distribución y sobreestimación mencionados, otro problema común en el aprendizaje reforzado *offline* es la falta de exploración. Este problema surge debido a la ausencia de retroalimentación en tiempo real del entorno durante el entrenamiento, lo que hace difícil descubrir acciones no probadas y limita el conocimiento del agente sobre las posibles acciones. A pesar de estas dificultades, las siguientes razones [13] permiten justificar que es posible implementar el aprendizaje reforzado *offline*, para obtener un agente que pueda actuar de forma adecuada, una vez es empleado en el ambiente:

- Es posible encontrar "buenas políticas" dentro de un *dataset* lleno de tanto malos como buenos comportamientos.
- Buenos comportamientos encontrados para cierto estado pueden ser generalizados sugiriendo que estos sean posibles buenos comportamientos en otros estados.
- Buenos comportamientos pueden ser combinados entre si.

2.4.1. Estrategias de aprendizaje reforzado offline

Se distinguen principalmente 3 tipos de estrategias de aprendizaje reforzado *offline*. La primera de ellas es la de evaluación *offline* mediante *importance sampling*, que consiste en una técnica usada para estimar propiedades de una distribución en particular, buscando en este caso obtener las distintas trayectorias de posibles políticas de decisión que puedan existir en los datos. La segunda de ellas se realiza implementando programación dinámica, la cual busca aprender una función que relacione los distintos estados con las acciones, para luego extraer de ella la política de decisión óptima. El tercer tipo de estrategia corresponde al basado en modelos, donde en este el aprendizaje no se realiza solo en base a ir tomando acciones y ver el estado al que se llega, sino que durante el entrenamiento, también se le asiste al agente entrenado, pudiéndole entregar en casos información de que acción es mejor. Esto último puede considerarse como que posee un mayor factor de aprendizaje supervisado que el resto de estrategias. Como se ha mencionado antes, el trabajo realizado en este informe se centra en el análisis y comparación de distintos tipos de estrategias de aprendizaje reforzado. Por lo tanto, a continuación, se presentarán tres algoritmos distintos que emplean una o más de las estrategias mencionadas anteriormente para el posterior análisis y comparación de los distintos tipos de estrategias de aprendizaje reforzado.

2.4.1.1. Conservative Q Learning

Un problema presente en los algoritmos *off-policy*, como el *Q-Learning* y sus derivados, es la sobre estimación de los valores de la función $Q(a, s)$. Este problema resulta ser más complejo para el aprendizaje por refuerzo *offline*, puesto que en su proceso de aprendizaje los datos empleados son estáticos y el cambio de distribución entre este conjunto de datos y la política aprendida se acentúa [7]. Con el fin de afrontar estas limitaciones, Kumar en al. [22] presentan este algoritmo, que plantea aprender una función Q conservadora, que se refiere a que el valor esperado de la política aprendida limite inferiormente al valor de la función Q real. Para esto, el algoritmo plantea que al momento de entrenar nuestra función

Q, se busque minimizar los valores Q esperados, al mismo tiempo que se considera el error de Bellman con respecto a los valores Q reales. De esta manera se utiliza la Ecuación 2.17

$$\begin{aligned} Q^{k+1} \leftarrow \underset{Q}{\operatorname{argmin}} \alpha \cdot (\mathbb{E}_{s\sim\mathcal{D},a\sim\mu(a|s)}[Q(s,a)] - \mathbb{E}_{s\sim\mathcal{D},a\sim\pi_\beta(a_t|s_t)}[Q(s_t,a_t)]) \\ + \frac{1}{2}\mathbb{E}_{s_t,a_t,s_{t+1}\sim\mathcal{D}}\left[\left(Q(s_t,a_t) - \mathcal{B}^\pi Q^k(s_t,a_t)\right)^2\right] \end{aligned} \tag{2.17}$$

en el proceso de actualización de nuestra función Q, donde α es un factor de *trade off*, $\mathcal{D}$ es el conjunto de datos con el que se entrena generado por la política $\pi_\beta(a|s)$, $\mu_(a|s)$ es la distribución actual de los pares estado-acción, y $\mathcal{B}^\pi$ es el operador de Bellman. De esta manera, la expresión dada está conformada por el cálculo de dos términos, siendo el primero la diferencia entre el valor esperado Q de la distribución actual y el valor esperado Q con respecto a la política que se está aprendiendo en el entrenamiento, ponderados por el factor de *trade off* escogido. El segundo término corresponde a un término de regularización que minimiza la discrepancia entre la función Q actualizada y la función Q aproximada por el operador de Bellman. Este enfoque pesimista no está libre de fallas, pues puede sufrir de subestimación de los valores de Q, y de un ajuste excesivo a los datos de entrenamiento, por lo que el parámetro α debe ser correctamente ajustado para evitar estos dos problemas.

2.4.1.2. Implicit Q Learning

Como ya se ha mencionado anteriormente, el aprendizaje reforzado *offline* trata de ser capaz de aprender una política mejor que la que se le presenta en los datos recolectados con los que aprende, mientras que al mismo tiempo debe minimizar la desviación de la política generada para evitar errores al enfrentarse con situaciones no vistas producto de un cambio de distribución. Con este problema en mente, Kostrikov et al. [23] proponen este algoritmo, el cual plantea no volver a evaluar acciones fuera del conjunto de datos con el que aprende, aún logrando ser capaz de aprender una mejor política que la presente en los datos, mediante la generalización. Para lograr esto, el algoritmo se implementa utilizando *soft actor-critic*, teniendo un actor que toma decisiones en el entorno y un crítico que evalúa la calidad de esas decisiones. La red neuronal del actor aprenderá la política $\pi(a_t|s_t)$, con la que seleccionará la acción mediante la Ecuación 2.18

$$a^*(s_t;\pi) = \max_a \pi(s_t,a). \tag{2.18}$$

Esta función de política será entrenada, para maximizar el retorno esperado y la entropía de la política. Esto se hará mediante la función de pérdida de la Ecuación 2.19

$$L(\psi) = \mathbb{E}_{(s_t,a_t)\sim\mathcal{D}}[A(s_t,a_t)\cdot\log\pi_\psi(a_t|s_t)], \tag{2.19}$$

En la que ψ son los parámetros de la red, y $A(s_t,a_t)$ es la función de ventaja que estimará el crítico. Esta función de ventaja, trabaja de similar manera que la vista en las redes de duelo, presentadas en la Subsubsección 2.3.1.2, a excepción de que se emplearán dos redes separadas para $Q(s,a)$ y $V(s)$. De esta manera el crítico constará de dos redes neuronales que estimarán la ventaja mediante la Ecuación 2.20

$$A(s,a) = Q(s_t,a_t;\theta) - V(s;\phi), \tag{2.20}$$

donde θ serán los parámetros de la función de acción-valor y ϕ los de la función de valor estado. La idea principal de hacer esto, es que ayuda a reducir la varianza en la estimación de los valores de acción-estado, lo que puede mejorar la estabilidad del agente y reducir la oscilación en su comportamiento.

2.4.1.3. Batch Constrained Q Learning

Los algoritmos *off-policy*, como DQL, requieren una gran cantidad de experiencia para aprender eficazmente, puesto que trabajar con conjunto de datos fijos introduce errores de extrapolación. Este método, conocido también como BCQ [24], aborda esta limitación utilizando lotes o *batches* de experiencia, en lugar de solo una experiencia a la vez, para actualizar la función de valor-acción. Esto permite al agente aprender de manera más eficiente, ya que puede aprovechar las correlaciones entre las experiencias en el lote. También este sistema de aprendizaje reforzado emplea *soft actor-critic* para aprovechar sus capacidades de generalización y estabilidad. Por consiguiente el algoritmo contará con un actor que toma decisiones en el entorno y un crítico, que en esta instancia lo emplea como un modelo generativo, para producir acciones similares a las vista en el *batch* empleado. En este método el actor aprenderá la función de acción-valor $Q_\theta(a_t|s_t)$, con la que seleccionará la acción mediante la expresión

$$a^*(s_t;\pi) = \underset{\frac{a|G_\omega(a|s_{t+1})}{\max_{a'} G_\omega(a'|s_{t+1})}>\tau}{\operatorname{argmax}} Q_\theta(s_t,a) \tag{2.21}$$

donde $G_\omega(a|s)$ es el modelo generativo, el cual calcula la probabilidad de la acción tomada con respecto a las acciones en el *batch*, la que se toma en cuenta solo de superar un *threshold* de probabilidad τ. También en esta ecuación θ son los parámetros de la red del actor, mientras que ω son los parámetros del crítico, que en este caso es el modelo generativo. El actor estimara la función Q mediante la clásica Ecuación 2.22 de Bellman

$$Q(s_t,a_t) = r(s_t,a_t) + \gamma \max_a Q(s_{t+1},a), \tag{2.22}$$

por lo que empleará el error de Bellman como función objetivo. Por otro lado, el modelo generativo operará similar al termino entrópico del *soft actor-critic* convencional, que tendrá la función objetivo a optimizar dada por la Ecuación 2.23

$$J_\omega = -\sum_{(s,a)\in M} \log G_\omega(a|s), \tag{2.23}$$

con M el *batch* empleado al momento de entrenar. Este modelo generativo se implementa mediante una red neuronal adicional, que se entrena utilizando conjuntos de datos de pares estado-acción, buscando que esta red aprenda a generar acciones para un estado dado, que sean similares a las acciones reales observadas en los datos de entrenamiento. Este método, posee la particularidad de que la política de acción que produce debe utilizar tanto la red del actor como la del crítico para operar.

Capítulo 3

Formulación del problema y metodología de trabajo

El trabajo efectuado en esta memoria consideró la realización de un trabajo exploratorio, a través del estudio de algunos de los esquemas más actuales de aprendizaje reforzado *offline*. También se buscó el comprender como es que estos esquemas pueden aplicarse para controlar un sistema dinámico del que se desconoce un modelo que lo represente, por medio del desarrollo de un controlador. Esto se realizó mediante un estudio del arte del aprendizaje reforzado *offline*, investigando posteriormente como se realiza su implementación en *Python*, lenguaje escogido debido a su extensivo uso en el campo del *Machine Learning*. Tras esta etapa de estudio inicial, se hizo la búsqueda de un sistema o proceso dinámico, que resulte de interés su estudio, con el fin de desarrollar un sistema de control sobre el, que implemente los esquemas de aprendizaje reforzado *offline* investigados.

3.1. Proceso dinámico a controlar: El péndulo invertido

Como se mencionó anteriormente, se desea evaluar la capacidad de los esquemas estudiados al emplearce como sistemas de control en procesos dinámicos. Por esta razón se ha optado por trabajar con el clásico problema de control del péndulo invertido o *CartPole* descrito por Barto, Sutton, y Anderson en "Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problem" [25]. Este sistema dinámico consiste en una barra conectada en uno de sus extremos a un carro, el cual se puede desplazar de manera horizontal por una pista. El objetivo principal es ser capaz de mantener la barra alzada de manera vertical, sin salirse de cierto intervalo de posición definido en la pista, y debiendo realizar esto solo aplicando fuerzas de manera horizontal sobre el carro. El modelo de este sistema estará definido por cuatro variables de estado consistentes de

- x: Posición horizontal del carro $[m]$,
- $\dot{x}$: Velocidad horizontal del carro $[m/s]$,
- θ: Ángulo del poste con el eje vertical $[rad]$,
- $\dot{\theta}$: Velocidad angular del poste $[rad/s]$,

y el controlador empleado sobre este sistema solo podrá actuar aplicando una fuerza de $10N$ en la dirección izquierda o derecha sobre el carro, a intervalos de tiempo discreto. En las Ecuaciones 3.1 y 3.2, se presentan las dinámicas no lineales que modelan el problema del péndulo invertido.

$$\ddot{\theta} = \frac{g \sin\theta_t \left[\frac{-F_t - ml\dot{\theta}_t^2 \sin\theta_t + \mu_c \text{sgn}(\dot{x}_t)}{m_c + m}\right] - \frac{\mu_p \dot{\theta}_t}{ml}}{l\left[\frac{4}{3} - \frac{m\cos^2\theta_t}{m_c + m}\right]}, \tag{3.1}$$

$$\ddot{x} = \frac{F_t + ml\left[\dot{\theta}_t^2 \sin\theta_t - \ddot{\theta}\cos\theta_t\right] - \mu_c \text{sgn}(\dot{x}_t)}{m_c + m}, \tag{3.2}$$

donde g es la aceleración de gravedad, m_c la masa del carro, m la masa de la barra, l la mitad del largo de la barra, μ_c el coeficiente de fricción entre el carro y la pista, μ_p el coeficiente de fricción entre el carro y la barra, y F_t la fuerza aplicada sobre el carro. En la Figura 3.1 se presenta un dibujo esquemático del péndulo invertido sobre un carro.

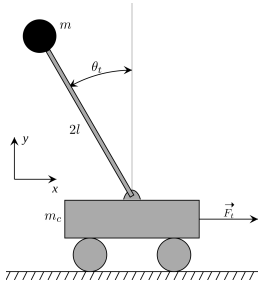

Figura 3.1: Diagrama del péndulo invertido sobre un carro. Fuente: [26]

3.2. Python y Gym

El trabajo realizado en esta investigación, decidió llevarse a cabo empleando el lenguaje de programación *Python*, por su popularidad y la cantidad de librerías con las que cuenta, para trabajar en problemas de *Machine Learning* y Control de Sistemas. También una motivación para emplear este lenguaje, fue la existencia de la API *Gym* de OpenAI, desarrollada originalmente para implementar algoritmos de aprendizaje reforzado. En *Gym*, existen múltiples ambientes virtuales, en los que es posible implementar esquemas de aprendizaje reforzado, y *Machine Learning* en general, además de ya contar con una versión del modelo de péndulo invertido, presentado en la Sección 3.1. En la Figura 3.2 se presenta el ambiente virtual del péndulo invertido, llamado *CartPole* en la API.

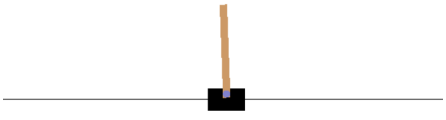

Figura 3.2: Ambiente del péndulo invertido en Gym

3.3. Datos para el entrenamiento fuera de linea

El enfoque fuera de linea requiere emplear conjuntos de datos previamente obtenidos para entrenar los distintos algoritmos de aprendizaje reforzado. Estos conjuntos de datos deben contar con la información, que permita representar el problema como un proceso de decisión de Markov [1], o sea deben contener trayectorias de estado, acción, recompensa y estado siguiente, que emplearán los agentes para aprender. Adicionalmente, los agentes requieren trabajar con datos de los que puedan extraer políticas de decisión convenientes, osea, que les muestren buenos comportamientos que aprender, para emplearlos en encontrar una política de acción con un alto rendimiento, o cercana a la óptima. Por esta razón, se plantea obtener datos de soluciones ya probadas del problema del péndulo invertido, proponiendo emplear tres sistemas de control distintos:

1. Controlador PID
2. Regulador Lineal cuadrático
3. Control predictivo por modelo

Se decidió escogerlos, porque ya existen implementaciones de ellos empleando *Python*, sobre el ambiente *CartPole* mencionado en la Sección 3.2, acelerando de esta manera el proceso de generar los datos. Debido a las distintas capacidades de estos métodos, se decidió emplear los controladores PID y LQR para generar datos del balanceo del péndulo, mientras que el MPC se utilizó para generar datos de como alzar el péndulo, cuando este se encuentra de manera vertical hacia abajo. Para cada uno de los controladores PID y LQR, se registraron 5000 interacciones en el ambiente *CartPole*, divididas en episodios de interacción cada 100 pasos en el ambiente. Al comienzo de cada uno de estos episodios, una variable aleatoria uniforme configuraba las condiciones iniciales del carro en los intervalos de posición $[-1.5, 1.5]$ y de inclinación $[-0.5, 0.5]$ *rad*, para ser capaz de abarcar en los datos un buen porcentaje del espacio de estado del sistema. Para el MPC se realizó algo similar que en los otros controladores, solo que para este se registró el doble de transiciones, y la posición angular inicial del péndulo, fue estando de manera vertical hacia abajo. Se decidió emplear el MPC de distinta forma al resto de los controladores, puesto que se usó para obtener datos de como elevar el péndulo, y no solo equilibrarlo, como en el caso de los otros. Debido a la capacidad de estos controladores, de resolver los problemas de elevar y balancear el péndulo, se refirió a los datos generados por ellos de origen experto. Adicionalmente a los registros de estos tres controladores, se registró 5000 transiciones empleando una política aleatoria, bajo las mismas condiciones que los dos primeros controladores mencionados. Se decidió generar estos datos aleatorios, para estudiar los efectos de la inclusión de estos, en el desempeño del aprendizaje de los agentes, para evaluar la capacidad de generalización y la robustez de cada algoritmo. De esta manera se consideró 4 conjuntos de datos según el tipo de política empleada en su generación:

1. Conjunto de datos PID de 5000 transiciones
2. Conjunto de datos LQR de 5000 transiciones
3. Conjunto de datos MPC de 10000 transiciones
4. Conjunto de datos aleatorios de 5000 transiciones

Estos conjutos de datos se emplearon para construir múltiples bases de datos, que se detallaran posteriormente en la Subsección 3.5.1. Estas bases de datos se construyeron en función de dos problemas definidos en la Subsección 3.5.2, con los que se propone evaluar el desempeño de los distintos algoritmos. Al momento de generar estos conjuntos de datos también se tomaron consideraciones de la función de recompensa que se emplearía en el ambiente, las que se detallaran a continuación.

3.3.1. Función de recompensa

El ambiente *CartPole* implementa por defecto, una función de recompensa discreta descrita por Sutton et al. [25] en su planteamiento del péndulo invertido, la que se puede generalizar mediante la Ecuación 3.3,

$$R_s = \begin{cases} 1 & \text{if } |x| < x_{threshold} \text{ and } |\theta| < \theta_{threshold}, \\ 0 & \text{otherwise.} \end{cases} \tag{3.3}$$

A esta función se le referirá, de ahora en adelante, como la función de recompensa default. Como esta es una función de tipo discreta, se propuso trabajar también con una función continua, para contrastar como impactan estas distintas funciones en el aprendizaje. La función propuesta se presenta en la Ecuación 3.4,

$$R_s = \cos\theta - 0.1 \cdot x^2. \tag{3.4}$$

A esta función, se le referirá como la función de recompensa customizada. Teniendo en cuenta estas dos funciones planteadas, los cuatro conjuntos de datos que se mencionaron en la Sección 3.3 fueron generados dos veces, empleando ambas funciones de recompensa por separado.

3.4. Formulación y configuración de algoritmos

Los algoritmos de aprendizaje reforzado que se implementados en este trabajo fueron:

1. *Deep Q-Learning*,
2. *Deep Q-Learning from Demostration*,
3. *Neural Fitted Q Iteration*,
4. *Conservative Q-Learning*,
5. *Implicit Q-Learning* y
6. *Batch Constrained Q-Learning*.

Cada uno de estos algoritmos, a excepción de NFQ, fue implementado considerando cuatro variantes del mismo. Estas variantes fueron:

1. versión vainilla (estandar),
2. con *Double Q-Networks*,
3. con *Dueling Networks* y

4. con *Double Dueling Q-Networks*,

donde las redes *Double* y *Dueling*, son las variantes sobre redes Q, detalladas en las subsecciones 2.3.1.1 y 2.3.1.2. Sobre el algoritmo NFQ solo se implementó la versión vainilla y *Double*, puesto que por su manera de codificarse, no era posible que empleara *Dueling Networks*. Esto da un total de 22 codificaciones de algoritmos de aprendizaje reforzado distintos. Estos algoritmos y sus variantes, fueron codificados empleando las librearías de *Numpy* y *Tensorflow* en *Python*. Se definió que cada red que emplearan estos algoritmos, serían de 2 capas ocultas de 32 neuronas, empleando la función de activación *ReLu*.

3.5. Entrenamiento y evaluación de las implementaciones

Para poder evaluar las capacidades de los distintos esquemas codificados, se definió dos casos del problema del péndulo invertido. El primero de ellos, fue entrenar los modelos para que fueran capaces de mantener equilibrado el péndulo, el segundo fue el de entrenarlos para que fueran capaces de alzar el péndulo, cuando este se encontrara en posición vertical hacia abajo.

3.5.1. Entrenamiento de agentes

Aunque algunos de los algoritmos de aprendizaje por refuerzo, como DQL, NFQ y DQfD, se desarrollaron inicialmente para su uso en línea, se han adaptado y mejorado con el tiempo para poder aplicarse offline, lo que ha dado lugar a la creación de algoritmos como CQL, BCQ e IQL, que se diseñaron específicamente para un uso fuera de linea. En el presente trabajo, se decidió entrenar todos los algoritmos mencionados de manera offline, con el objetivo de contrastarlos y analizar cómo las adaptaciones y mejoras en el aprendizaje por refuerzo han permitido expandir el uso de métodos como el Deep Q Learning a la implementación con datos fuera de línea. A continuación se presentará cómo se realizó el entrenamiento offline de cada uno de los algoritmos mencionados.

3.5.1.1. Balanceo de péndulo

Teniendo en consideración el primer caso planteado anteriormente, se realizó una serie de consideraciones, con respecto a los datos a emplear, en el entrenamiento de los agentes. Tomando en consideración el trabajo realizado por Schweighofer et al. [27], donde se analiza como las características de los datasets empleados, durante el entrenamiento influencian el desempeño de los algoritmos *offline*, se ha decidido distinguir la diferencia entre los datos obtenidos a partir de, una política de decisión experta y los obtenidos por una política aleatoria. Estos primeros datos mencionados corresponden a los obtenidos a través de las técnicas de control descritas en la Sección 3.3, los que se considera que representan maneras óptimas de resolver el problema del péndulo invertido. Con esto en consideración se plantean dos escenarios en el entrenamiento de cada algoritmo y sus respectivas variantes, siendo el primero donde se entrega a los agentes una base de datos solo con data experta, y en la segunda la adición de los datos obtenidos por la política aleatoria. Se busca poder contrastar estos dos escenarios y evaluar la influencia que pueden ejercer la calidad de datos empleados en el entrenamiento de los agentes.

Otra consideración respectiva a las bases de datos empleadas, considera el distinguir entre

las funciones de recompensa empleadas durante la obtención de los datos. Como se mencionó anteriormente en "Datos para el entrenamiento fuera de linea", para la obtención de datos se consideró trabajar tanto con la función de recompensa definida por Sutton et al. [25], como también con la función de recompensa customizada presentada en Subsección 3.3.1, la que buscaba reflejar de manera más precisa el objetivo al cual se desea llevar el péndulo. De esta manera se busca adicionar dos escenarios más a los planteados según el uso de solo data experta o mixta, agregando la consideración de cual función de recompensa se empleó al obtener los datos. Para el escenario planteado con data experta, se construyeron bases de datos conformadas por los conjuntos de datos obtenidos a partir e los controladores PID y LQR. En cuanto al caso de data mixta, se generaron bases de datos que consideraban los mismos datos para el escenario experto, pero que adicionalmente integraron los conjuntos de datos obtenidos con la política aleatoria. Por consiguiente se entrenarán los agentes considerando los 4 siguientes escenarios:

- Datos expertos empleando función de recompensa clásica.
- Datos expertos empleando función de recompensa customizada.
- Datos mixtos empleando función de recompensa clásica.
- Datos mixtos empleando función de recompensa customizada.

Estos escenarios plantean dos bases de datos expertas, conformadas por 10000 registros de transiciones cada una, obtenidas empleando los controladores, pero cada una empleando una función de recompensa diferente, y dos bases de datos mixtas conformadas por 15000 registros, pues adicionan a los datos expertos las 5000 transiciones generadas por la política aleatoria. Considerando los distintos escenarios formulados, se decidió realizar el proceso de entrenamiento de los agentes durante 3000 épocas, y empleando *batches* de 32 registros, extraídos de la base de datos según el escenario respectivo de ese agente, utilizando un sampleo aleatorio en cada entrenamiento, pero empleando la misma semilla para el sampleador respectivo de cada agente. Se utilizó una misma semilla para que cada agente fuera entrenado utilizando los mismos registros de datos, con el fin de reducir lo más posible los factores estocásticos en el proceso de entrenamiento.

3.5.1.2. Elevación del péndulo

Ahora, tomando en consideración el segundo caso planteado, se optó por trabajar solo con el conjunto de datos MPC y empleando la función de recompensa customizada. Se optó por esta manera, ya que para este caso no se tiene intención de estudiar los efectos de las distintas funciones de recompensas planteadas, ni tampoco los efectos de la calidad de los datos. Solo se desea poder evaluar la capacidad de los algoritmos, de poder aprender a elevar el péndulo, puesto que se considera una tarea mucho más compleja que solo mantenerlo balanceado. Por esto se decidió realizar un proceso de entrenamiento similar que el caso anterior, entrenamiento los agentes durante 3000 épocas, y empleando *batches* de 32 registros.

3.5.2. Evaluación de los agentes

A modo general, la evaluación de los múltiples agentes implementados se realizó cada 10 épocas de entrenamiento, evaluando 10 episodios distintos en el ambiente del péndulo invertido.

3.5.2.1. Balanceo del péndulo

Para el caso de aprender a balancear el péndulo, se considerando un horizonte máximo de 2500 *steps* o pasos para cada episodio. Usando como referencia las decisiones de evaluación empleadas en Neural Fitted Q Iteration [21], para las evaluaciones las posición inicial del carro y de la barra fueron obtenidas empleando una variable aleatoria uniforme, empleando de manera respectiva los intervalos $[-1.5, 1.5]$ y $[-0.5, 0.5]$ rad para cada posición, y empleando la misma semilla para esta variable aleatoria. En cuanto a la velocidad del carro y de la barra, estas se dejaron en cero. La tarea planteada para los agentes consideró el restringir la posición del carro en el rango $-2.4 < x < 2.4$ y la posición de la barra en el rango $-\pi/2 < \theta < \pi/2$ $[rad]$, buscando reportar el desempeño de los agentes mediante la cantidad de pasos máximo que estos lograban balancear al carro sin violar las restricciones de posición establecidas.

3.5.2.2. Elevación del péndulo

Para el caso de aprender a alzar el péndulo, se consideró un horizonte máximo de 500 *steps* o pasos para cada episodio. Los estados iniciales del péndulo se asignaron en $X = 0$ y $\theta = \pi$ $[rad]$. A diferencia del caso anterior de balancear el péndulo, en este no se evaluó la cantidad de recompensa que obtenían los agentes, sino que, se registró la mayor cantidad de *steps* seguidos, que el agente era capaz de elevar el péndulo y mantener su inclinación dentro del rango $-0.3 < \theta < 0.3$ $[rad]$.

Capítulo 4

Resultados

En este capítulo se procederá a presentar los resultados obtenidos tras la realización de los experimentos planteados en la Sección 3.5. Los experimentos se dividen, de manera general, en evaluar en primera instancia la capacidad de los agentes de balancear correctamente el péndulo de manera vertical hacia arriba, y en segunda, el ser capaz de alzar el péndulo al encontrarse inicialmente en posición vertical hacia abajo. Como se mencionó en la Sección 3.4, para los experimentos se consideraron los algoritmos *Deep Q-Learning*, *Deep Q-Learning from Demostration*, *Neural Fitted Q Iteration*, *Conservative Q-Learning*, *Implicit Q-Learning* y *Batch-Constrained Q-Learning*, junto con las variantes producto de implementar redes dobles y de duelo en su arquitectura.

A continuación se expondrán los resultados del primer experimento, consistente en balancear el péndulo, presentándolos según la categorización de política que compone los datos (experta o mixta), y el tipo de función de recompensa (default o customizada) al momento de generar los datos. Como se especificó en la Subsubsección 3.5.1.1, la base de datos experta correspondió a transiciones obtenidas a partir de los controladores PID y LQR, mientras que la base de datos mixta consideró los mismos datos de la experta, pero adicionando las transiciones obtenidas a partir de una política aleatoria. Para cada una de las variantes de los algoritmos mencionados en el párrafo anterior, se presentarán la función de pérdida obtenida durante el entrenamiento, y la recompensa promedio obtenida al evaluar por 10 episodios el desempeño del agente, cada 10 épocas de entrenamiento. Posteriormente se exhibirán los resultados del segundo experimento, correspondiente al de elevar el péndulo, en el que también se presentarán la función de pérdida obtenida, pero reemplazando la recompensa obtenida por el periodo de tiempo máximo que logra el agente mantener alzado el péndulo, para cada una de las variantes de los algoritmos trabajados. Tras la presentación de cada experimento respectivo se expondrá un análisis de los resultados reunidos.

4.1. Experimento 1: Balanceo del péndulo invertido

En esta sección se presentarán los resultados de las pruebas realizadas para el experimento 1, detallado en la Subsubsección 3.5.2.1. Esto se hará, mediante la presentación de las funciones de pérdida obtenidas durante el entrenamiento, y las recompensas obtenidas al evaluar cada uno de los algoritmos implementados, junto con sus diferentes variaciones arquitectónicas, detalladas en la Sección 3.4. Las bases de entrenamiento empleadas tuvieron un tamaño de 10000 registros para el caso de política experta, y 15000 registros para la mixta, con las que se entrenaron los agentes durante 3000 épocas, y empleando *batches* de 32 registros. La evaluación de los agentes se realizó considerando un horizonte máximo de 2500 *steps* o pasos para cada episodio. Las evaluaciones se realizaron empleando una variable aleatoria uniforme para la posición inicial del carro y la barra, y se mantuvieron la velocidad del carro y la barra en cero. El objetivo de los agentes fue restringir la posición del carro en el rango $-2.4 < x < 2.4$ y la posición de la barra en el rango $-\pi/2 < \theta < \pi/2$ $[rad]$, reportando el desempeño en términos de la cantidad de pasos máximo que los agentes lograron balancear al carro sin violar las restricciones de posición establecidas. La presentación de los resultados se encuentran divididos según el tipo de política que conforma los datos empleados en el entrenamiento, y según la función de recompensa empleada al momento de generar los datos. Posterior a la presentación de los resultados, se expone un análisis del desempeño obtenido por los distintos algoritmos.

4.1.1. Entrenamiento con base de datos generada por política experta y función de recompensa default

4.1.1.1. Deep Q-Learning

En las figuras 4.1, 4.2, 4.3 y 4.4 se visualizan las funciones de pérdida y recompensas obtenidas de entrenar y evaluar los agentes que implementaron el algoritmo Deep Q-Learning, junto con sus variantes Double DQL, Dueling DQL y Double Dueling DQL, para el experimento de balancear el péndulo invertido.

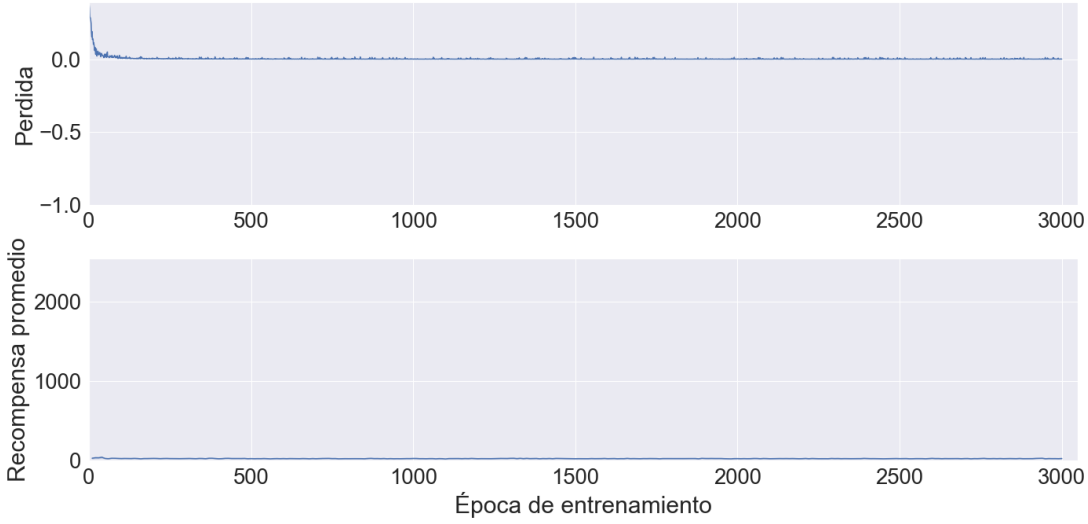

Figura 4.1: Pérdida y recompensas obtenidas para el modelo DQL

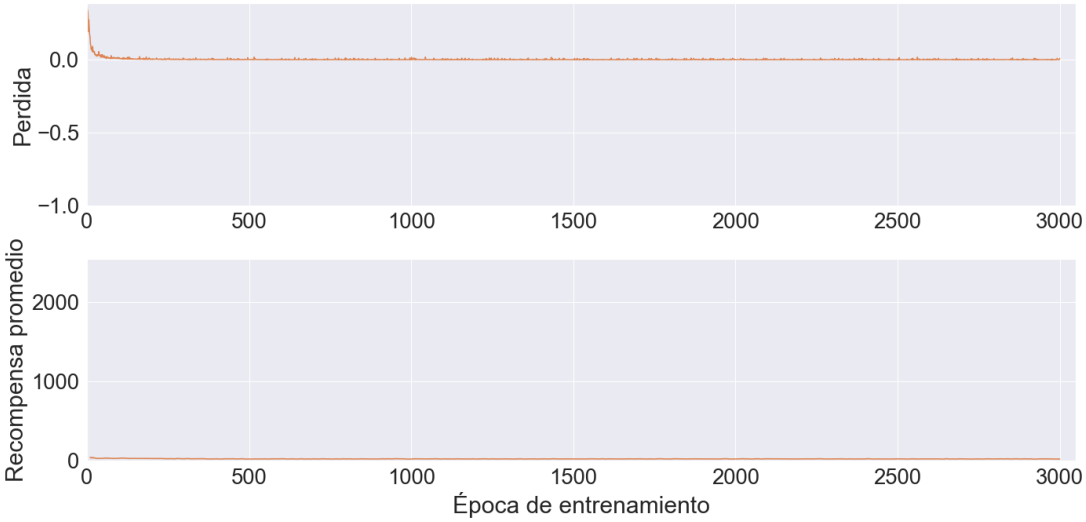

Figura 4.2: Pérdida y recompensas obtenidas para el modelo Double DQL

Figura 4.3: Pérdida y recompensas obtenidas para el modelo Dueling DQL

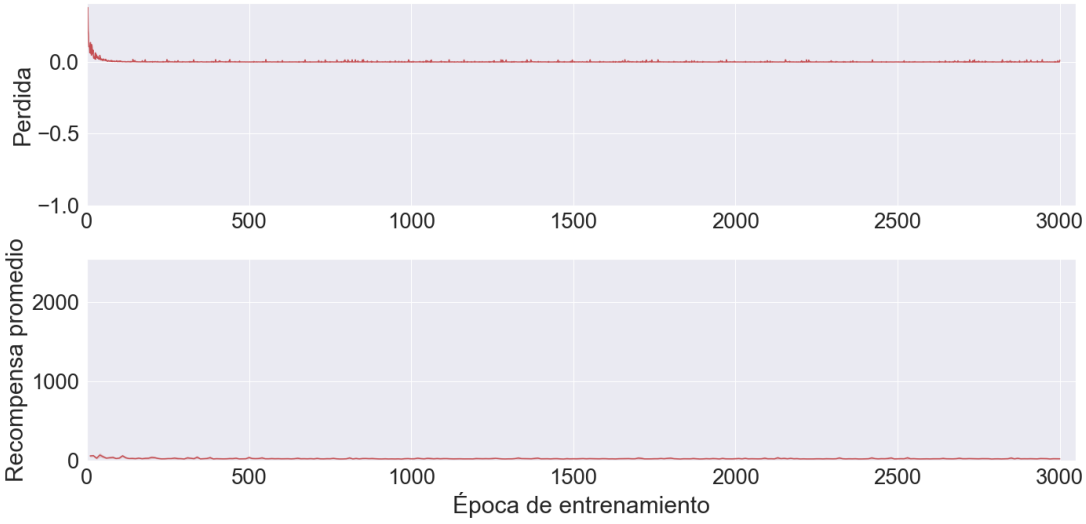

Figura 4.4: Pérdida y recompensas obtenidas para el modelo Double Dueling DQL

De los resultados presentados en los gráficos anteriores, se aprecia que este algoritmo no es capaz de mantener balanceado el péndulo invertido, al ser entrenado de manera *offline* con datos generados por una política experta empleando la función de recompensa default de este problema.

4.1.1.2. Deep Q-Learning from Demostration

En las figuras 4.5, 4.6, 4.7 y 4.8 se visualizan las funciones de pérdida y recompensas obtenidas de entrenar y evaluar los agentes que implementaron el algoritmo Deep Q-Learning from Demostration, junto con sus variantes Double DQfD, Dueling DQfD y Double Dueling DQfD respectivamente, para el experimento de balancear el péndulo invertido.

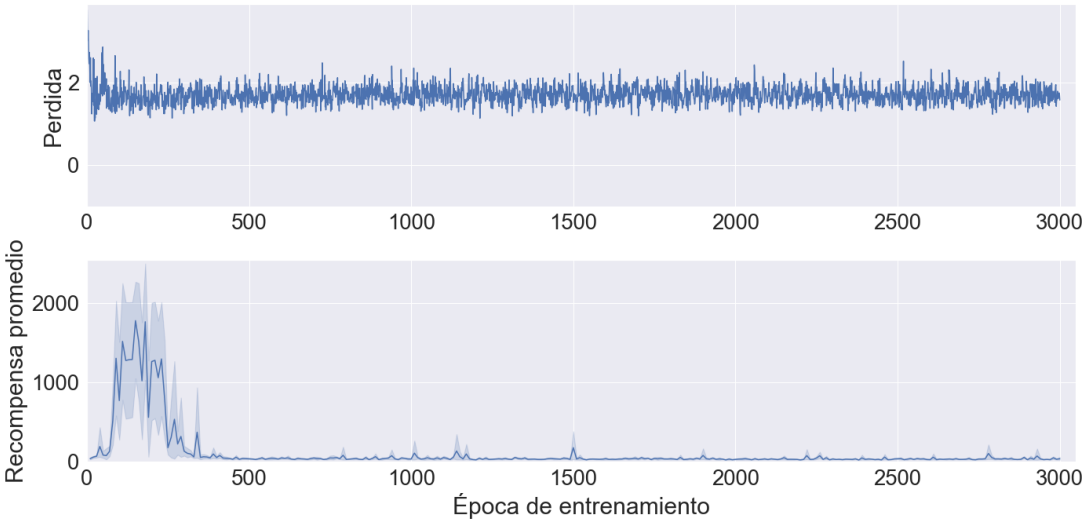

Figura 4.5: Pérdida y recompensas obtenidas para el modelo DQfD

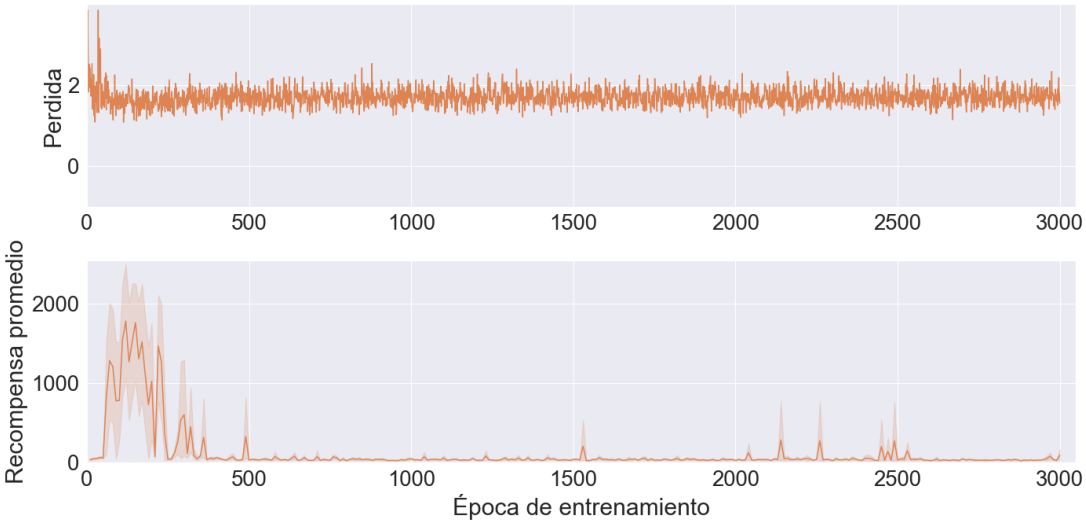

Figura 4.6: Pérdida y recompensas obtenidas para el modelo Double DQfD

Figura 4.7: Pérdida y recompensas obtenidas para el modelo Dueling DQfD

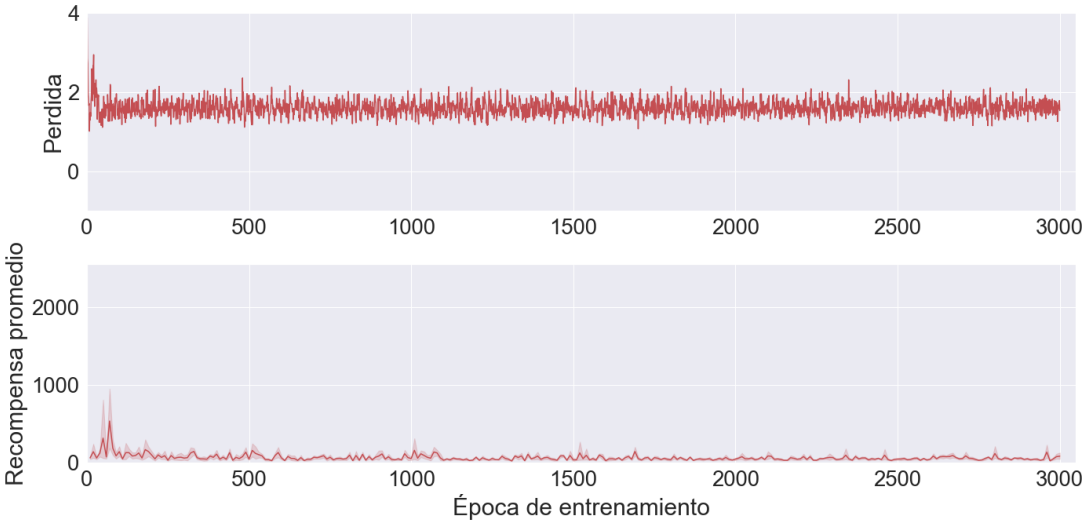

Figura 4.8: Pérdida y recompensas obtenidas para el modelo Double Dueling DQfD

De los resultados presentados en los gráficos anteriores, se aprecia que las versiones de este algoritmo que emplearon redes neuronales convencionales estaban logrando aprender una política satisfactoria durante las primeros 300 épocas, pero con las épocas siguientes su desempeño cayó no logrando aprender una política satisfactoria, fracasando de igual manera que las versiones del algoritmo que emplearon redes de duelo.

4.1.1.3. Neural Fitted Q Iteration

En las figuras 4.9 y 4.10 se visualizan las funciones de pérdida y recompensas obtenidas de entrenar y evaluar los agentes que implementaron el algoritmo Neural Fitted Q Iteration, junto con su variante Double NFQ, para el experimento de balancear el péndulo invertido.

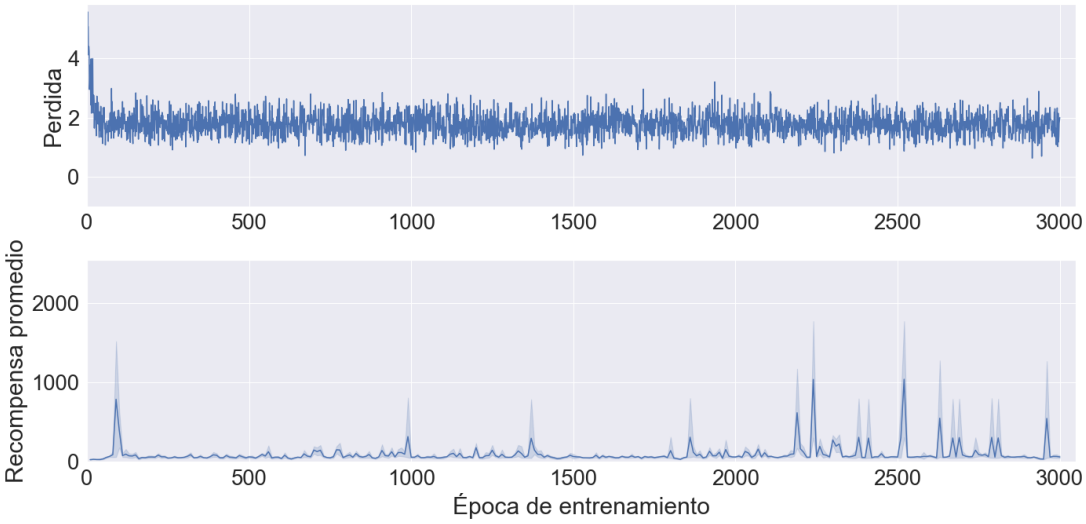

Figura 4.9: Pérdida y recompensas obtenidas para el modelo NFQ

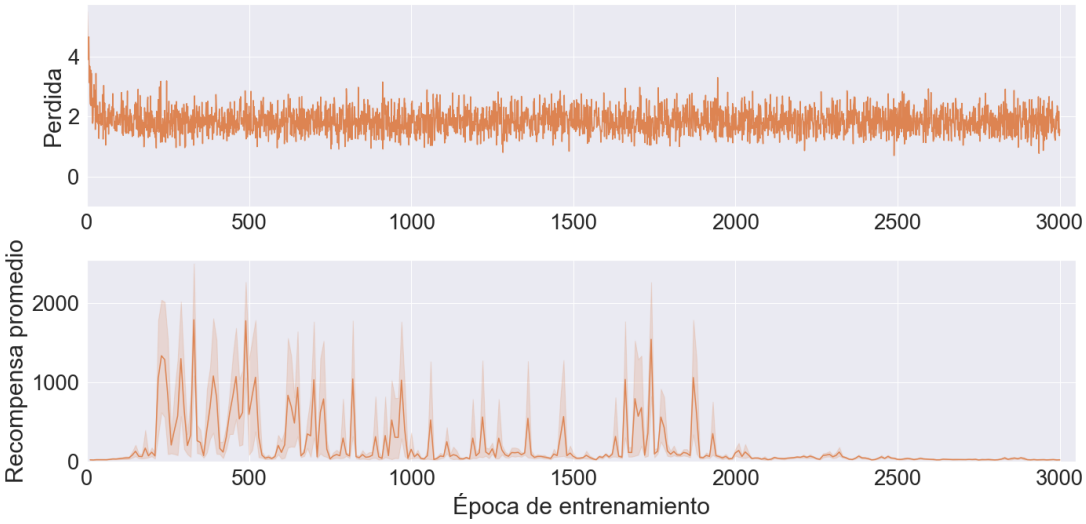

Figura 4.10: Pérdida y recompensas obtenidas para el modelo Double NFQ

Los resultados presentados en los gráficos, muestran que este algoritmo se beneficia notablemente de emplear *Double Q Networks*, puesto que aumenta su eficiencia en la velocidad de aprendizaje, y su desempeño máximo alcanzado. No obstante, este algoritmo sigue fallando en lograr alcanzar una política idónea.

4.1.1.4. Conservative Q-Learning

En las figuras 4.11, 4.12, 4.13 y 4.14 se visualizan las funciones de pérdida y recompensas obtenidas de entrenar y evaluar los agentes que implementaron el algoritmo Conservative Q-Learning, junto con sus variantes Double CQL, Dueling CQL y Double Dueling CQL respectivamente, para el experimento de balancear el péndulo invertido.

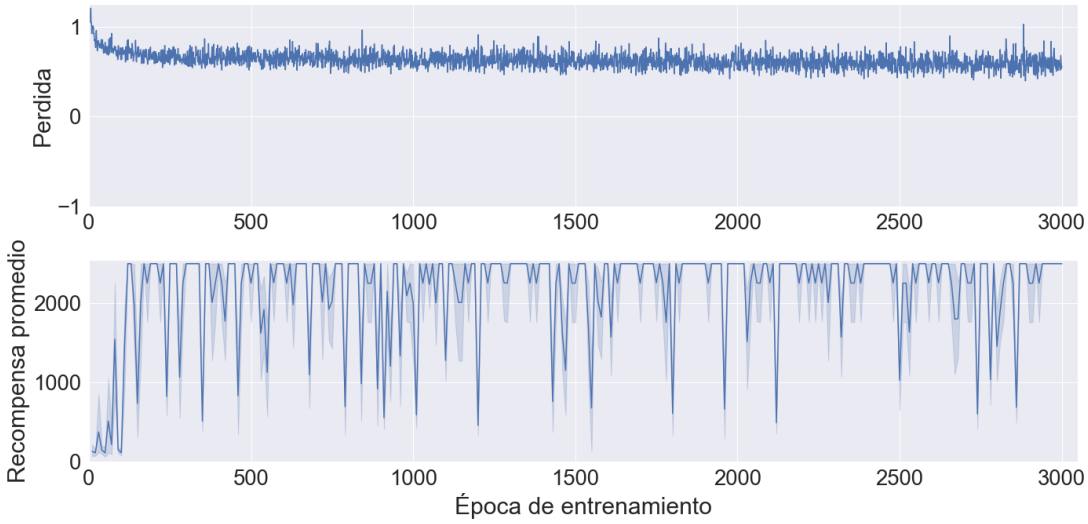

Figura 4.11: Pérdida y recompensas obtenidas para el modelo CQL

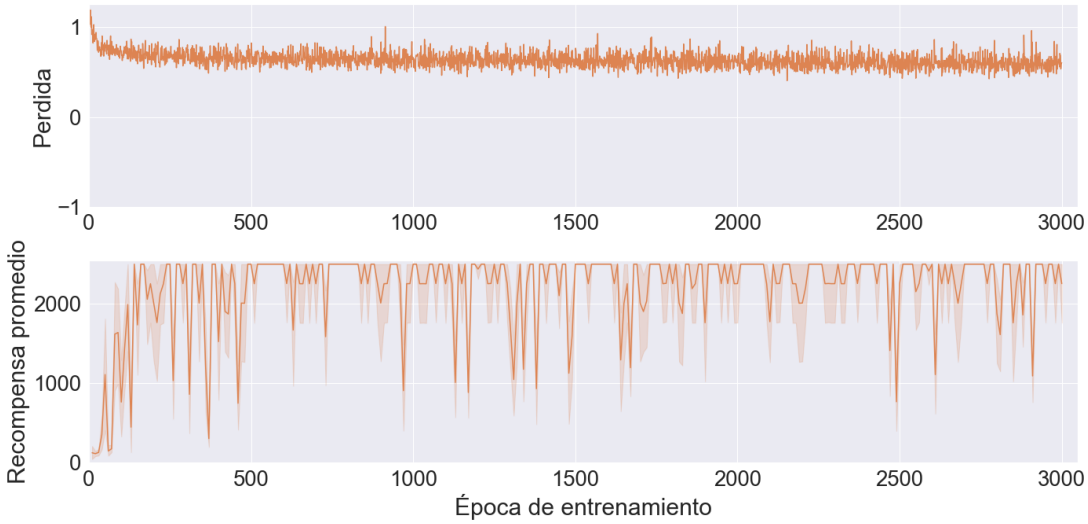

Figura 4.12: Pérdida y recompensas obtenidas para el modelo Double CQL

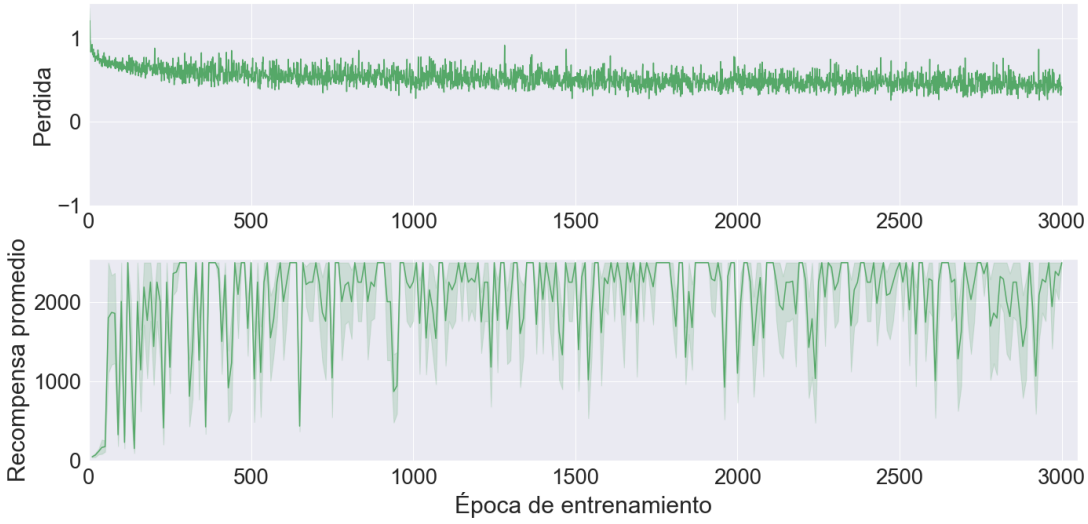

Figura 4.13: Pérdida y recompensas obtenidas para el modelo Dueling CQL

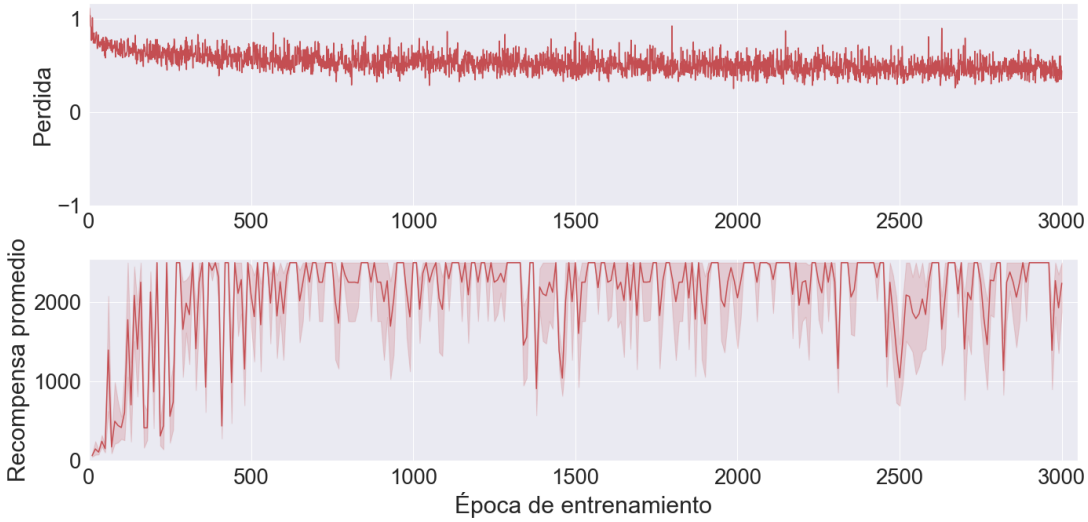

Figura 4.14: Pérdida y recompensas obtenidas para el modelo Double Dueling CQL

Los resultados presentados en los gráficos, muestran que en esta configuración de datos expertos con la función de recompensa default, el algoritmo alcanza un alto desempeño, con relativamente pocas épocas de entrenamiento. No obstante, se aprecia como con el pasar de las épocas de entrenamiento, la recompensa oscila notoriamente, indicando un posible sobreajuste a los datos por parte del agente.

4.1.1.5. Implicit Q-Learning

En las figuras 4.15, 4.16, 4.17 y 4.18 se visualizan las funciones de pérdida y recompensas obtenidas de entrenar y evaluar los agentes que implementaron el algoritmo Implicit Q-Learning, junto con sus variantes Double IQL, Dueling IQL y Double Dueling IQL respectivamente, para el experimento de balancear el péndulo invertido.

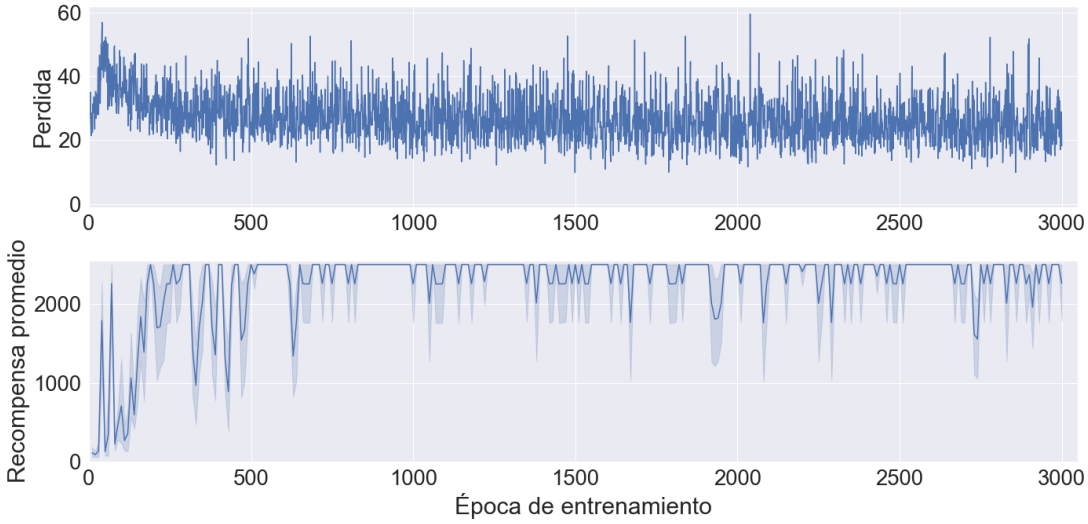

Figura 4.15: Pérdida y recompensas obtenidas para el modelo IQL

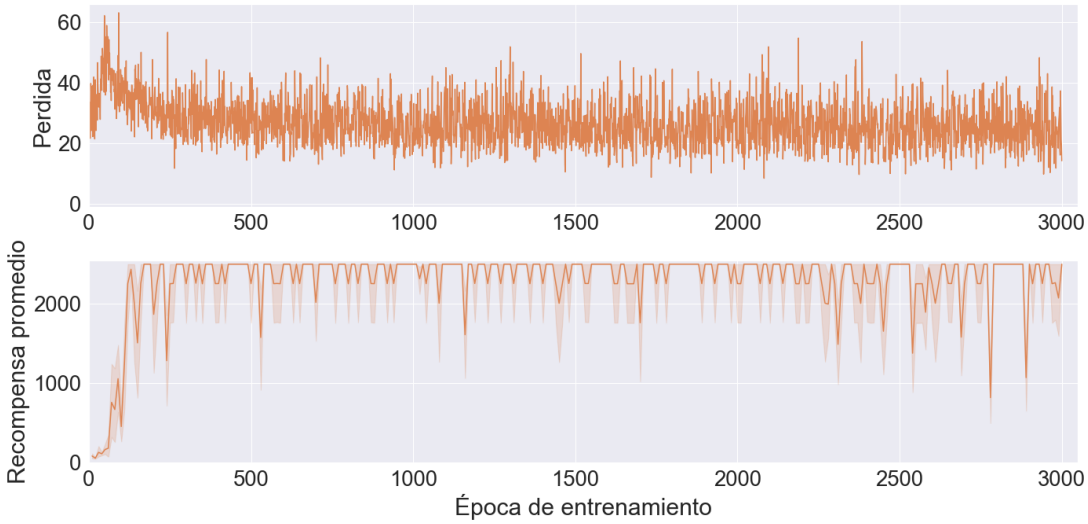

Figura 4.16: Pérdida y recompensas obtenidas para el modelo Double IQL

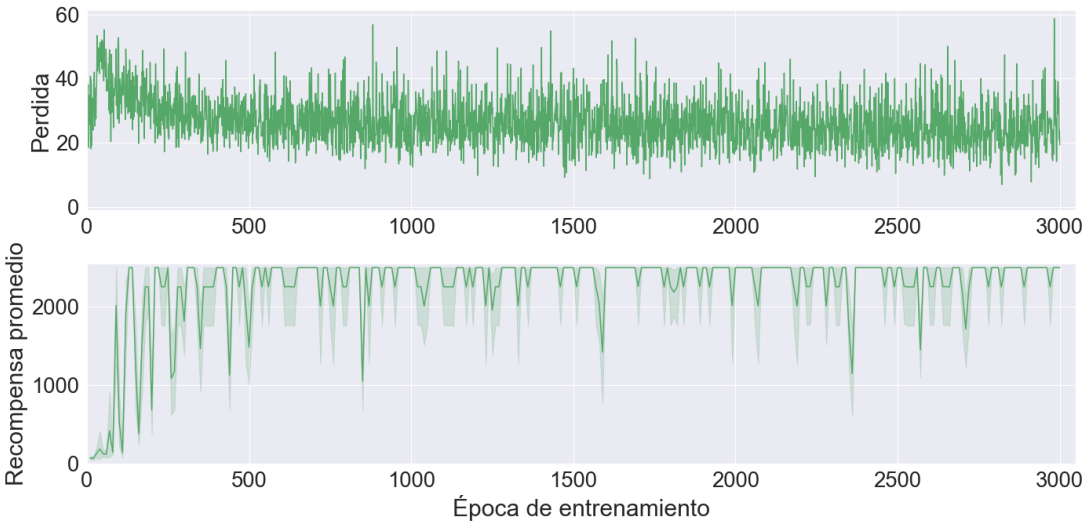

Figura 4.17: Pérdida y recompensas obtenidas para el modelo Dueling IQL

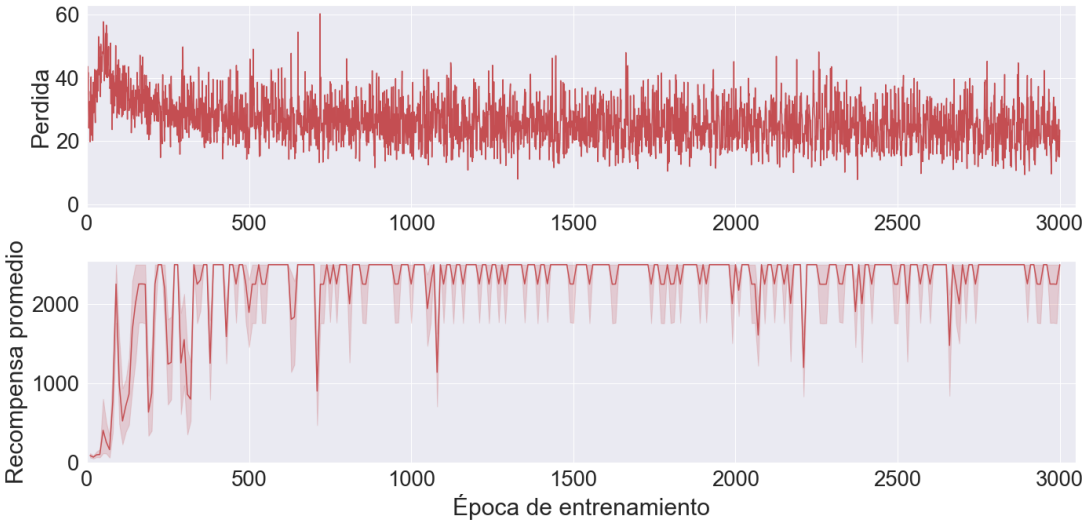

Figura 4.18: Pérdida y recompensas obtenidas para el modelo Double Dueling IQL

Los resultados muestran que en esta configuración, de datos expertos con una recompensa default, el algoritmo alcanza un alto desempeño con relativa rapidez.

4.1.1.6. Batch-Constrained Deep Q-Learning

En las figuras 4.19, 4.20, 4.21 y 4.22 se visualizan las funciones de pérdida y recompensas obtenidas de entrenar y evaluar los agentes que implementaron el algoritmo Batch-Constrained Deep Q-Learning, junto con sus variantes Double BCQ, Dueling Double BCQ y Double Dueling BCQ respectivamente, para el experimento de balancear el péndulo invertido.

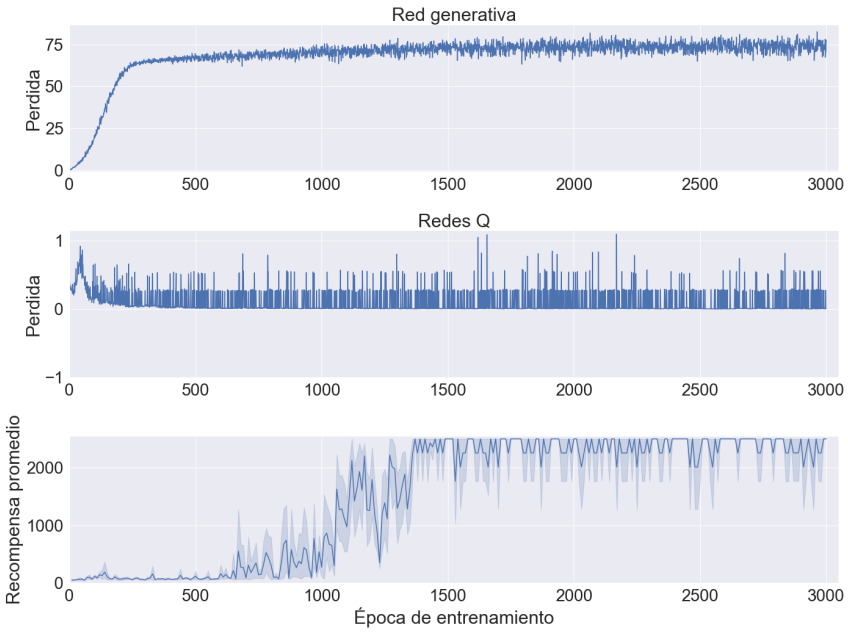

Figura 4.19: Pérdidas y recompensas obtenidas para el modelo BCQ

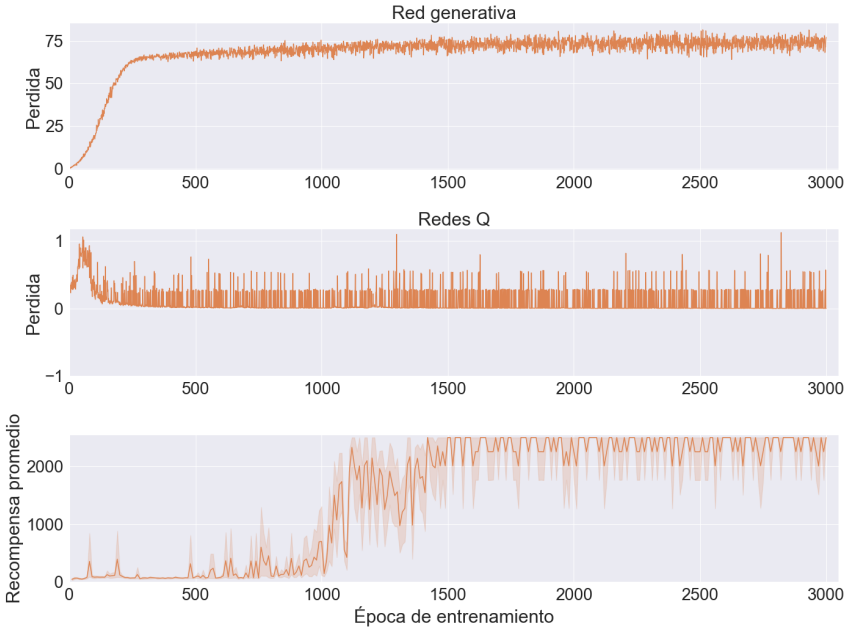

Figura 4.20: Pérdidas y recompensas obtenidas para el modelo Double BCQ

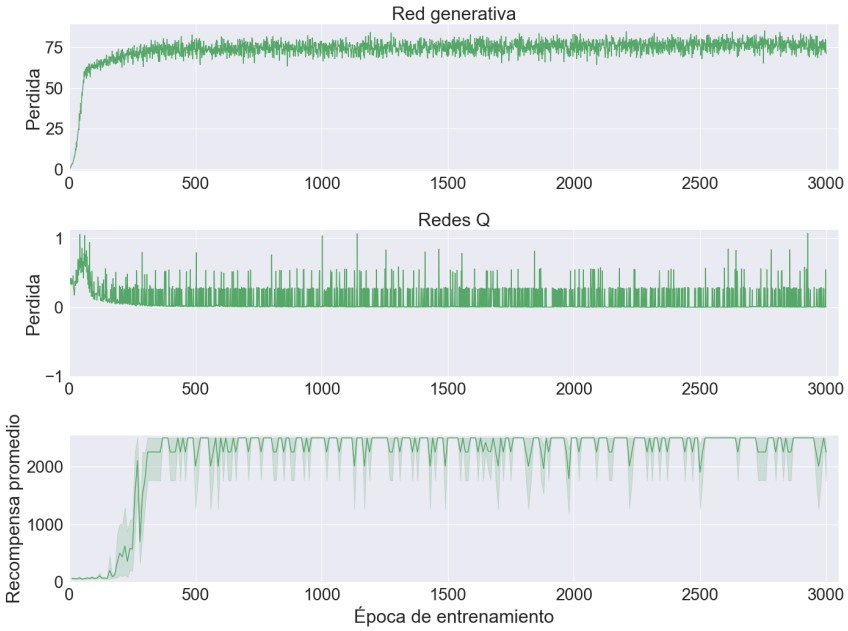

Figura 4.21: Pérdidas y recompensas obtenidas para el modelo Dueling BCQ

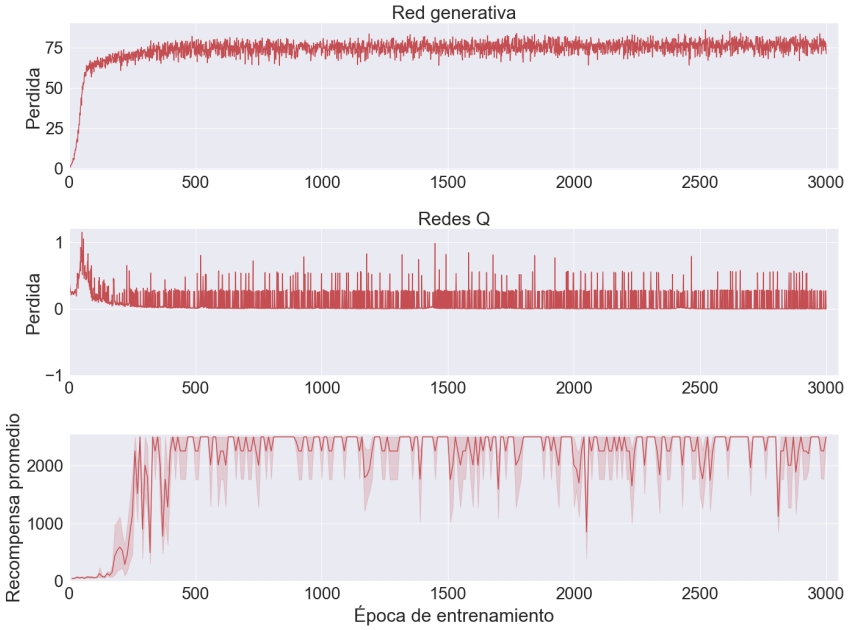

Figura 4.22: Pérdidas y recompensas obtenidas para el modelo Double Dueling BCQ

Los resultados de los gráficos muestran, que este algoritmo se beneficia de emplear *Dueling Networks*, puesto que aceleran su proceso de aprendizaje, en contraste a los casos donde solo implementa *Q-Networks* normales.

4.1.2. Entrenamiento con base de datos generada por política experta y función de recompensa customizada

4.1.2.1. Deep Q-Learning

En las figuras 4.23, 4.24, 4.25 y 4.26 se visualizan las funciones de pérdida y recompensas obtenidas de entrenar y evaluar los agentes que implementaron el algoritmo Deep Q-Learning, junto con sus variantes Double DQL, Dueling DQL y Double Dueling DQL, para el experimento de balancear el péndulo invertido.

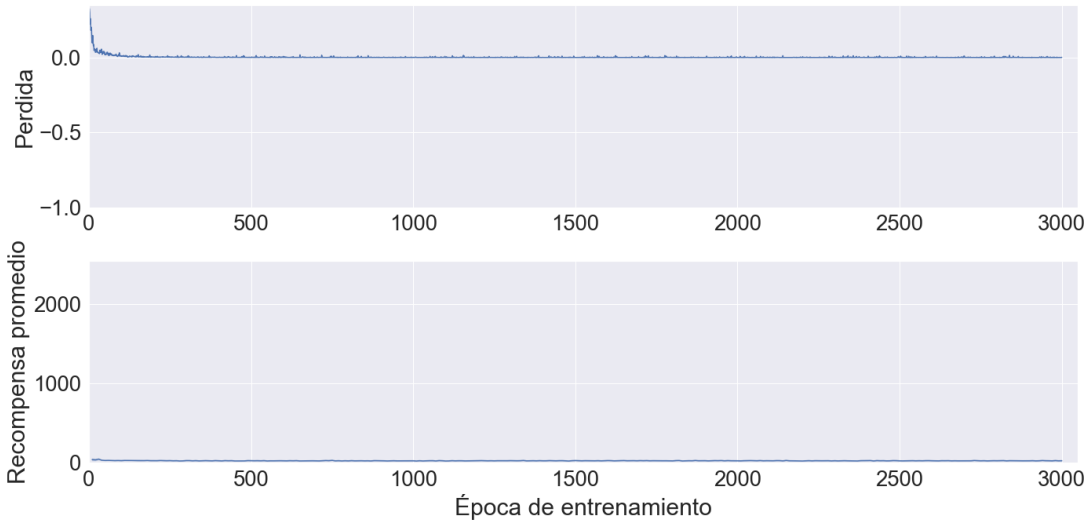

Figura 4.23: Pérdida y recompensas obtenidas para el modelo DQL

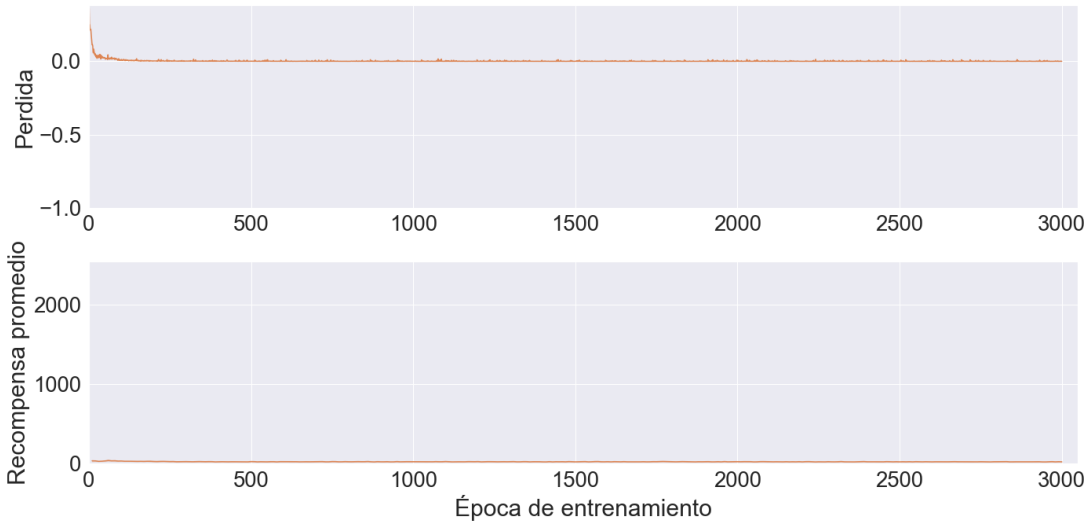

Figura 4.24: Pérdida y recompensas obtenidas para el modelo Double DQL

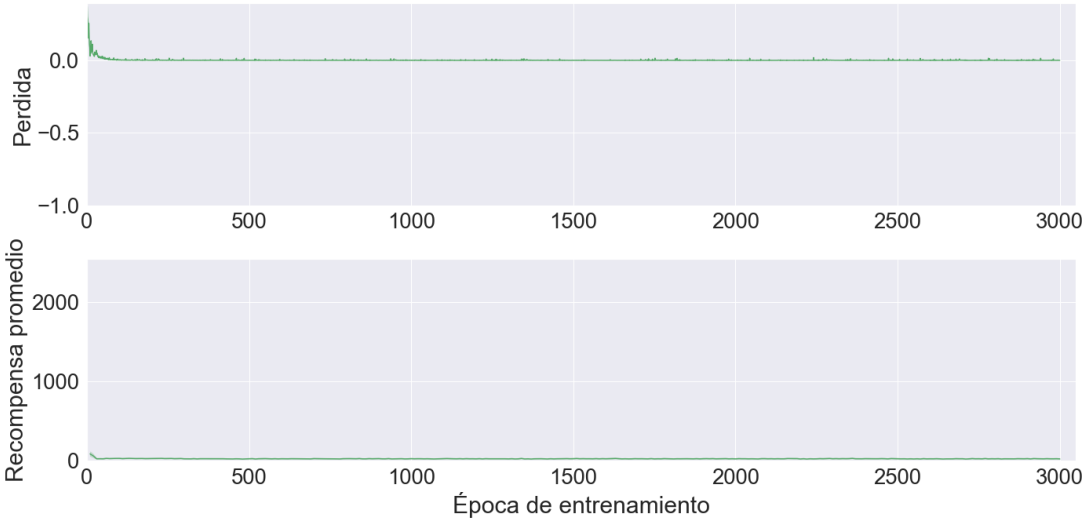

Figura 4.25: Pérdida y recompensas obtenidas para el modelo Dueling DQL

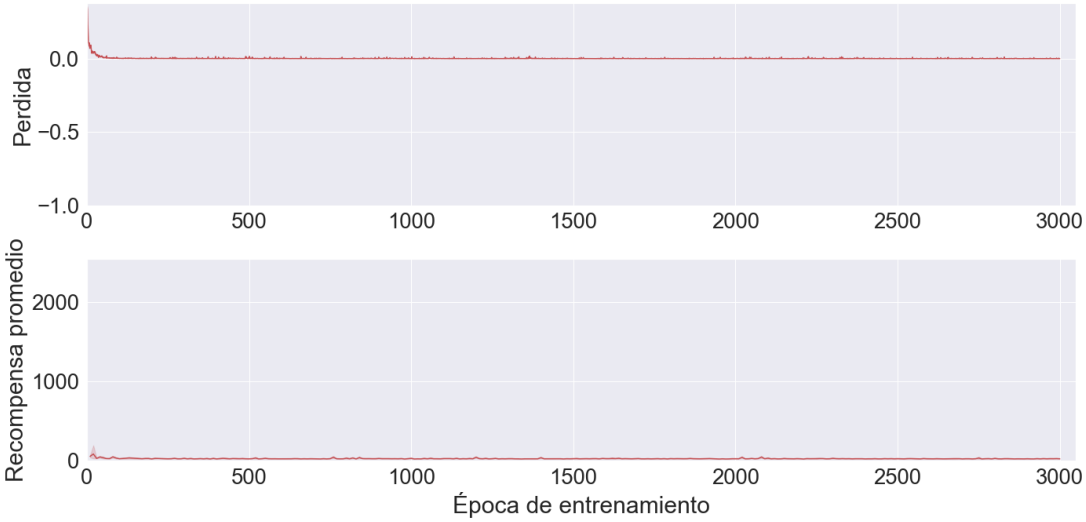

Figura 4.26: Pérdida y recompensas obtenidas para el modelo Double Dueling DQL

Los resultados muestran, que este algoritmo falla nuevamente, aún empleando datos con una función de recompensa más precisa, al ser continua.

4.1.2.2. Deep Q-Learning from Demostration

En las figuras 4.27, 4.28, 4.29 y 4.30 se visualizan las funciones de pérdida y recompensas obtenidas de entrenar y evaluar los agentes que implementaron el algoritmo Deep Q-Learning from Demostration, junto con sus variantes Double DQfD, Dueling DQfD y Double Dueling DQfD respectivamente, para el experimento de balancear el péndulo invertido.

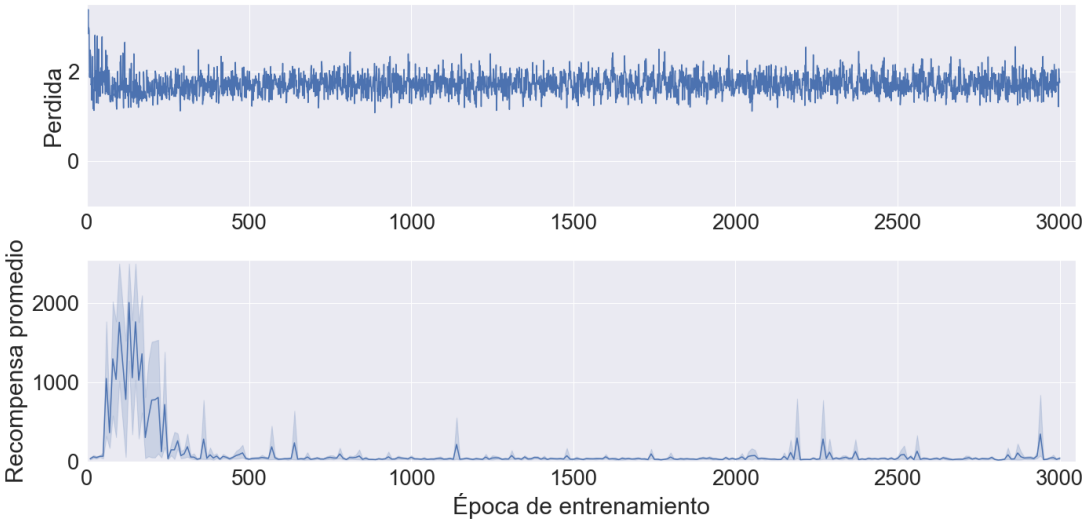

Figura 4.27: Pérdida y recompensas obtenidas para el modelo DQfD

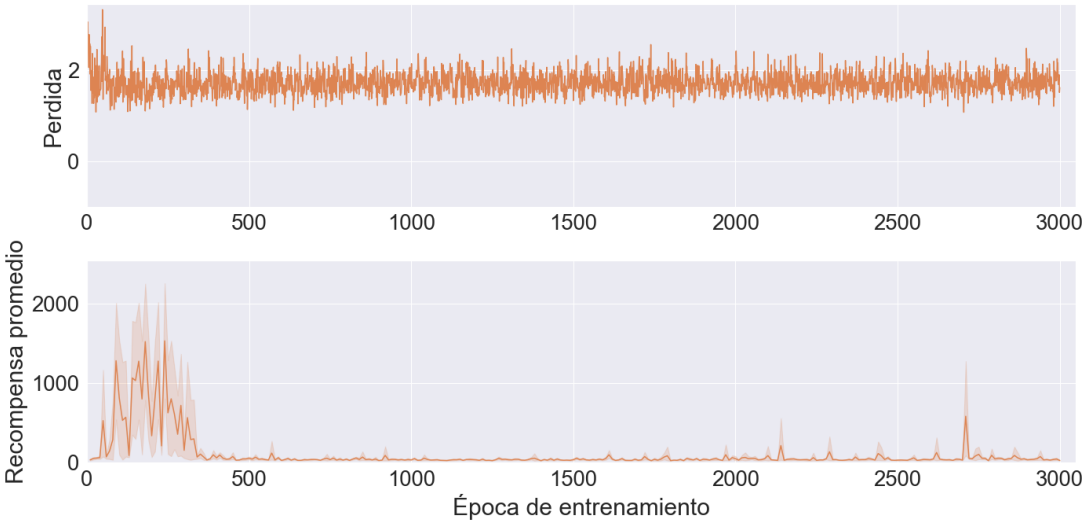

Figura 4.28: Pérdida y recompensas obtenidas para el modelo Double DQfD

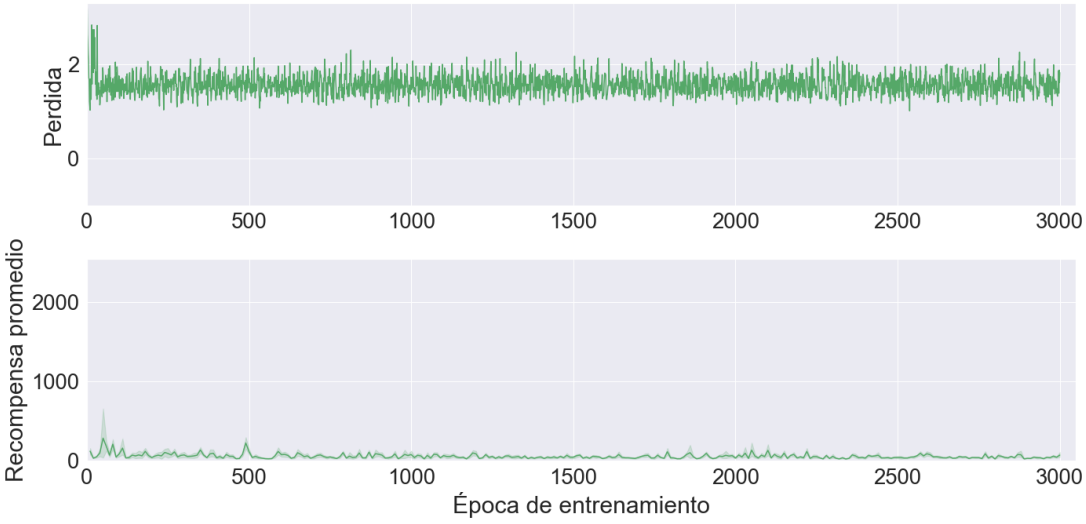

Figura 4.29: Pérdida y recompensas obtenidas para el modelo Dueling DQfD

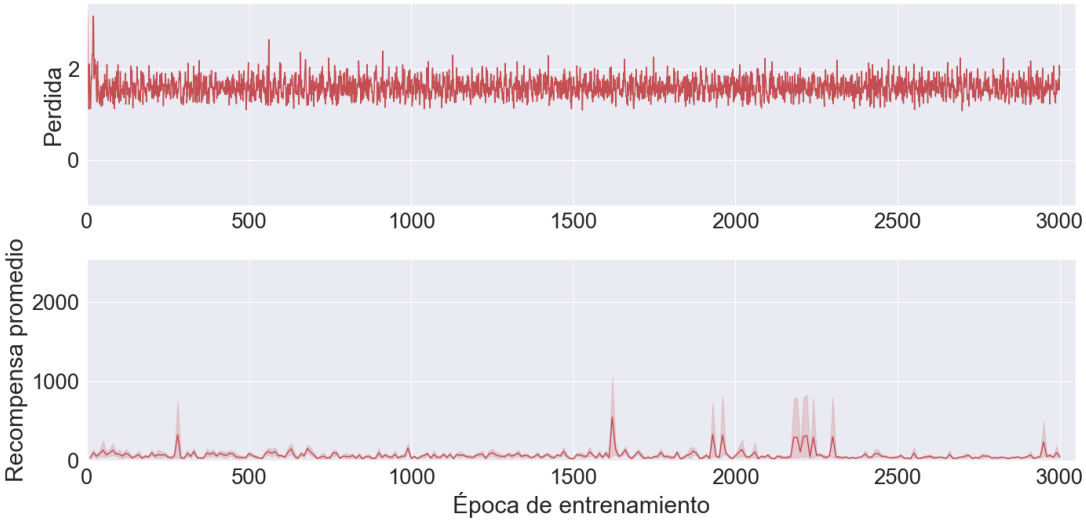

Figura 4.30: Pérdida y recompensas obtenidas para el modelo Double Dueling DQfD

Los resultados de los gráficos muestran, nuevamente, que este algoritmo no se beneficia de la arquitectura de las *Dueling Networks*, y que al contrario, afecta negativamente a su desempeño.

4.1.2.3. Neural Fitted Q Iteration

En las figuras 4.31 y 4.32 se visualizan las funciones de pérdida y recompensas obtenidas de entrenar y evaluar los agentes que implementaron el algoritmo Neural Fitted Q Iteration, junto con su variante Double NFQ, para el experimento de balancear el péndulo invertido.

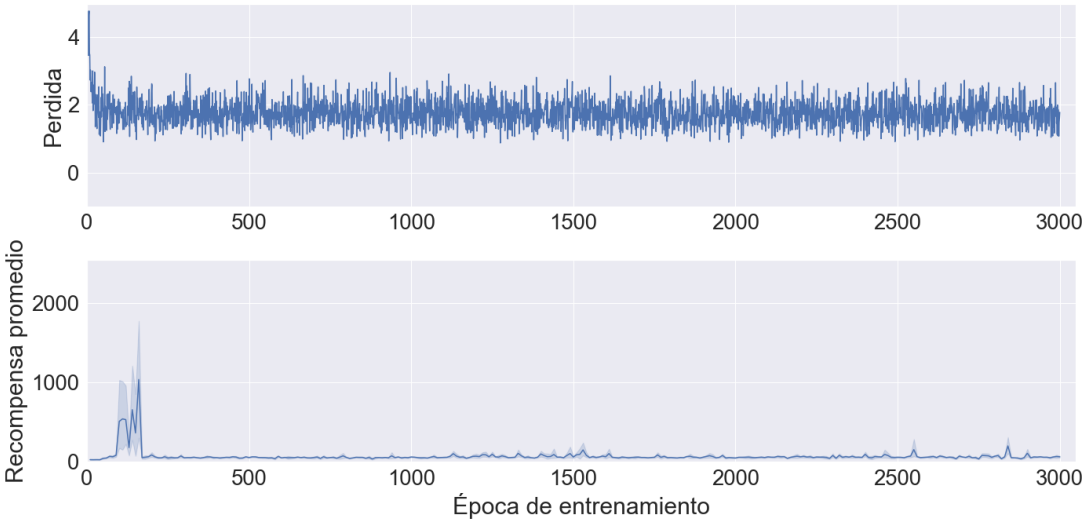

Figura 4.31: Pérdida y recompensas obtenidas para el modelo NFQ

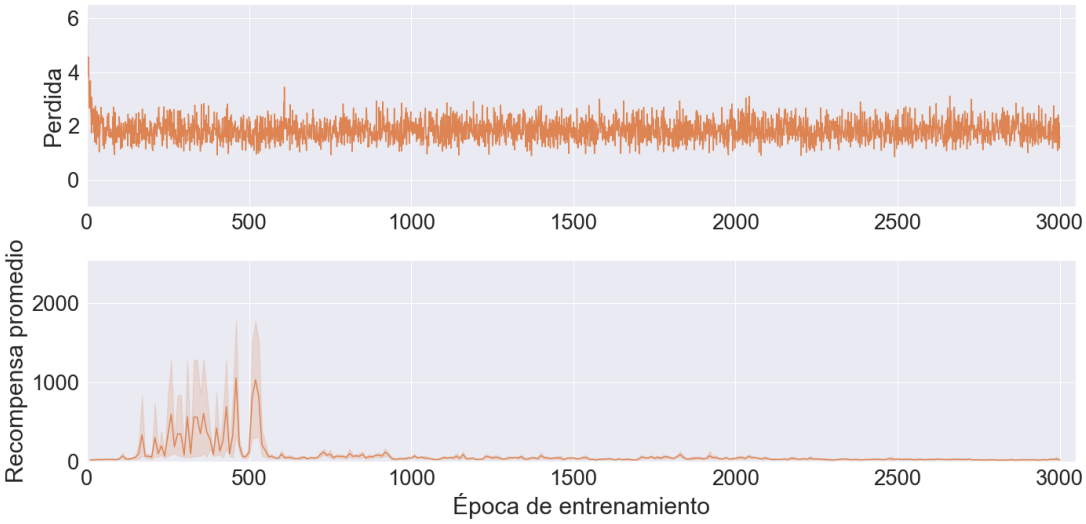

Figura 4.32: Pérdida y recompensas obtenidas para el modelo Double NFQ

Los resultados de los gráficos muestran, que el cambio de función de recompensa, empleado en los datos resulta infructuoso, para este algoritmo.

4.1.2.4. Conservative Q-Learning

En las figuras 4.33, 4.34, 4.35 y 4.36 se visualizan las funciones de pérdida y recompensas obtenidas de entrenar y evaluar los agentes que implementaron el algoritmo Conservative Q-Learning, junto con sus variantes Double CQL, Dueling CQL y Double Dueling CQL respectivamente, para el experimento de balancear el péndulo invertido.

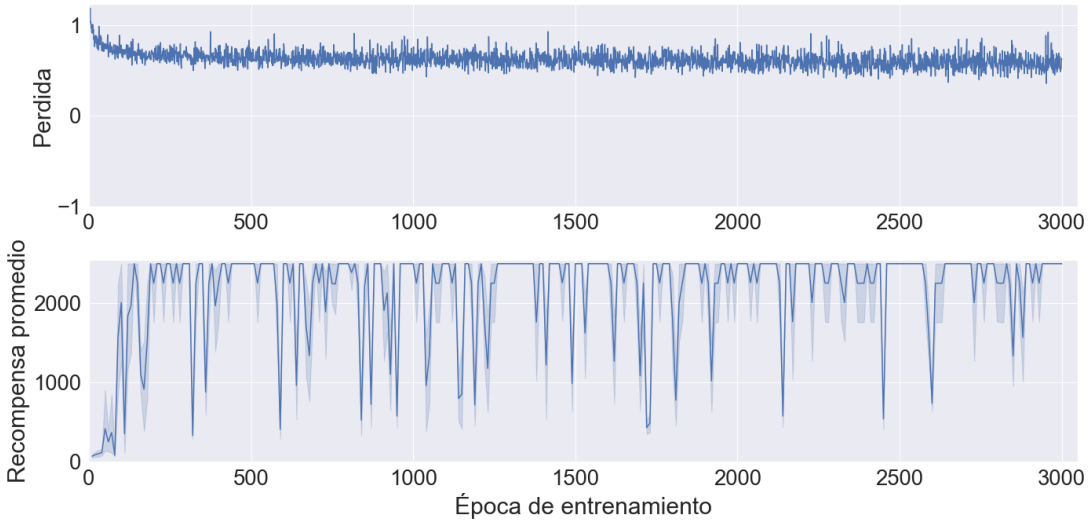

Figura 4.33: Pérdida y recompensas obtenidas para el modelo CQL

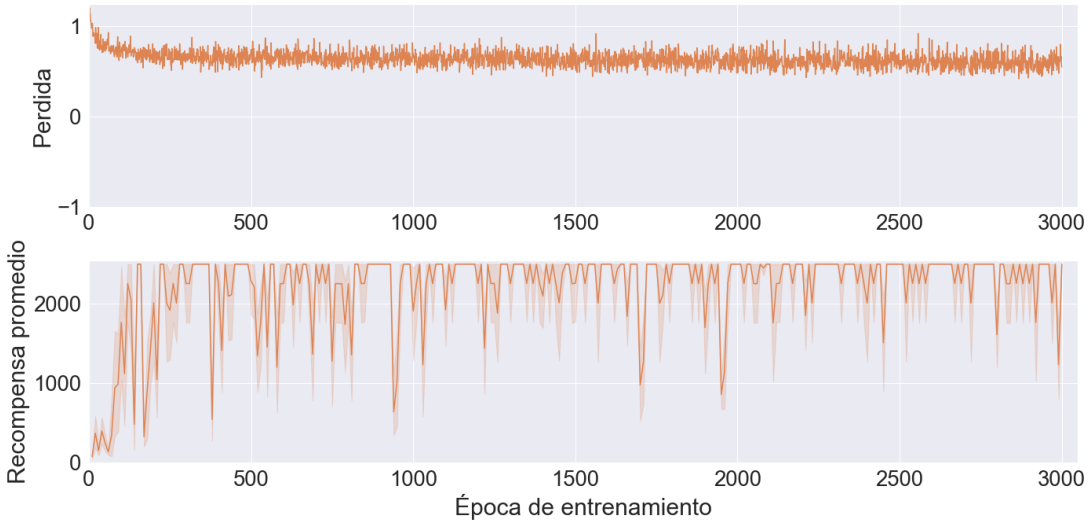

Figura 4.34: Pérdida y recompensas obtenidas para el modelo Double CQL

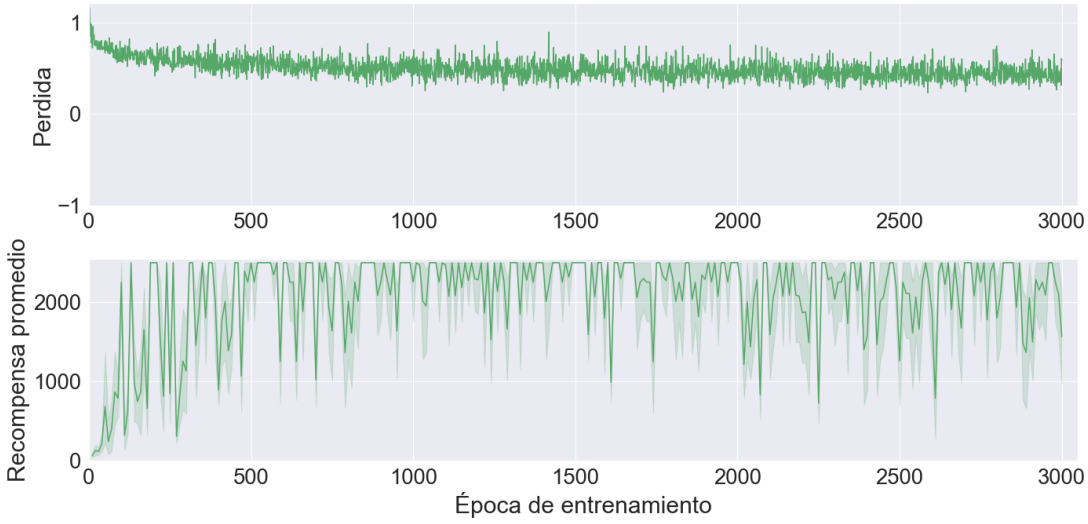

Figura 4.35: Pérdida y recompensas obtenidas para el modelo Dueling CQL

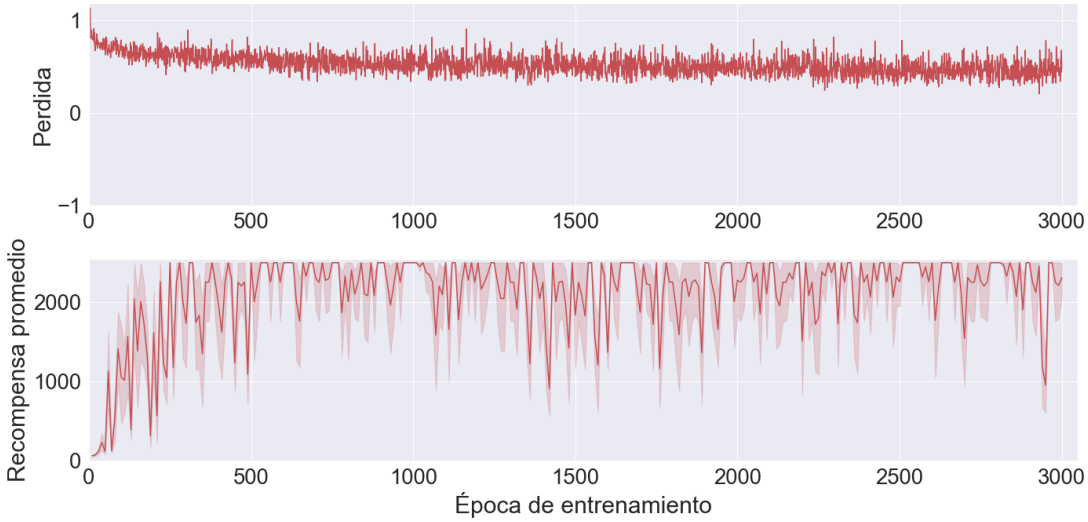

Figura 4.36: Pérdida y recompensas obtenidas para el modelo Double Dueling CQL

Los resultados de los gráficos, muestran como el cambio de función de recompensa del conjunto de datos de entrenamiento, enlentece la velocidad de aprendizaje al emplear *Dueling Networks*. Nuevamente se aprecia para este algoritmo, una alta oscilación de la recompensa, atribuible a un sobreajuste a los datos. También se distingue mayor oscilación de la recompensa en las *Dueling Networks*.

4.1.2.5. Implicit Q-Learning

En las figuras 4.37, 4.38, 4.39 y 4.40 se visualizan las funciones de pérdida y recompensas obtenidas de entrenar y evaluar los agentes que implementaron el algoritmo Implicit Q-Learning, junto con sus variantes Double IQL, Dueling IQL y Double Dueling IQL respectivamente, para el experimento de balancear el péndulo invertido.

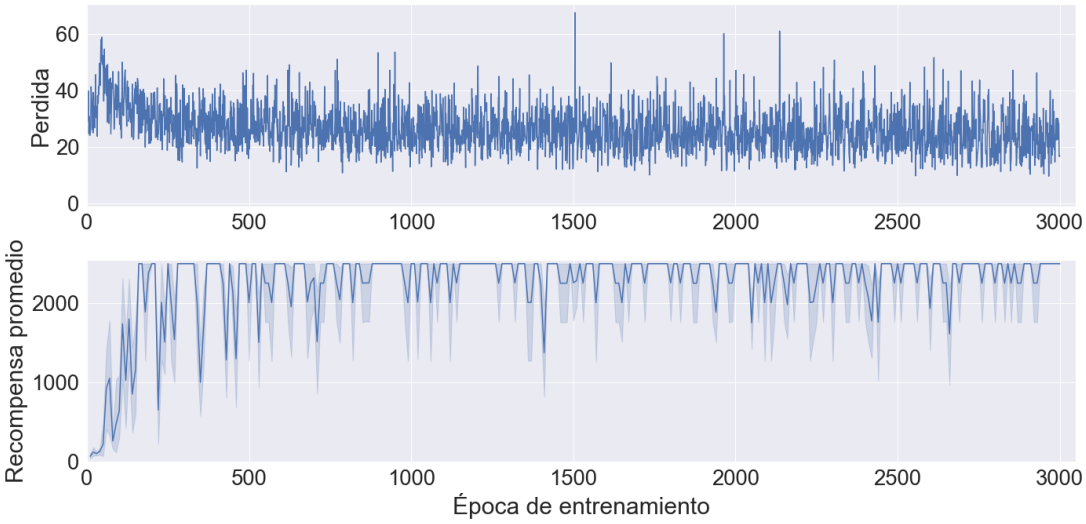

Figura 4.37: Pérdida y recompensas obtenidas para el modelo IQL

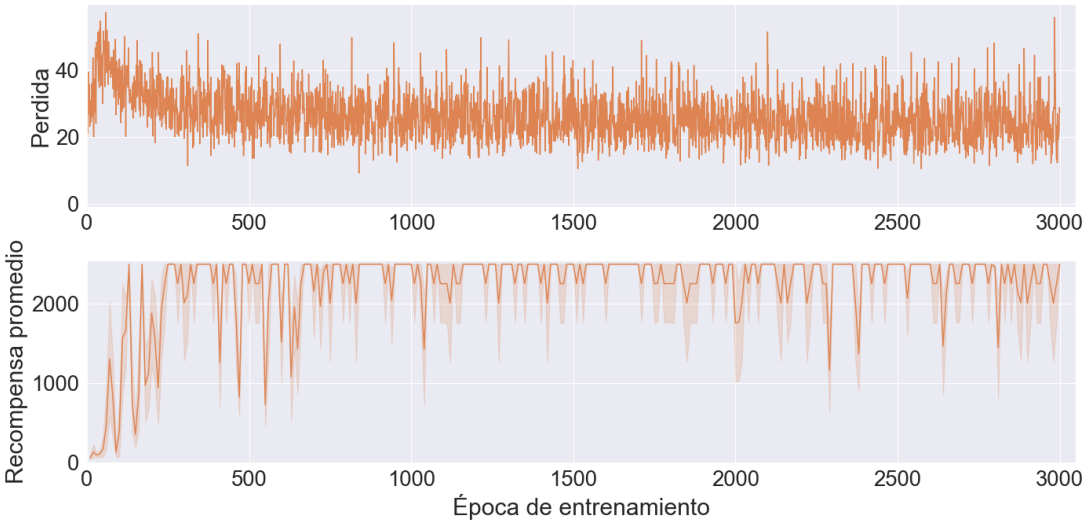

Figura 4.38: Pérdida y recompensas obtenidas para el modelo Double IQL

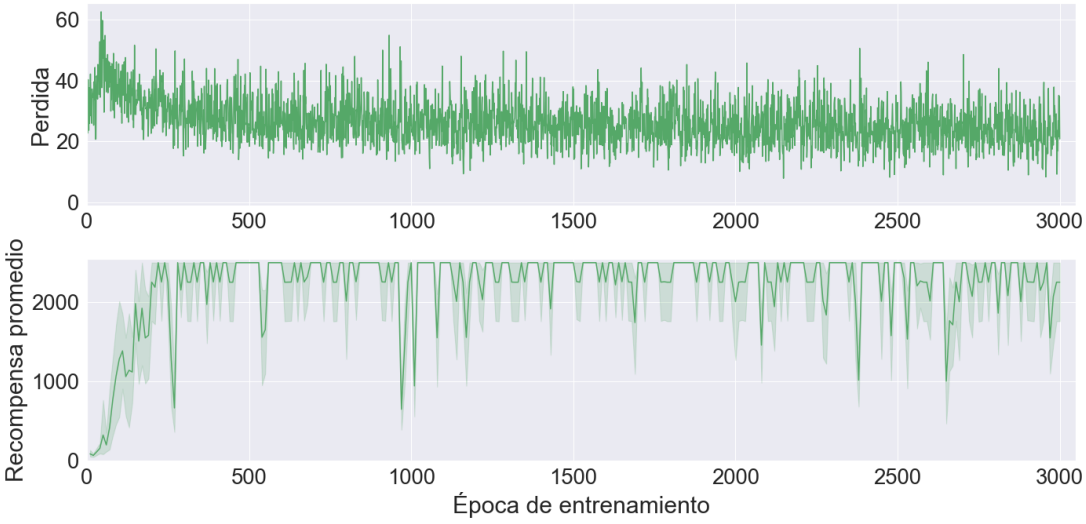

Figura 4.39: Pérdida y recompensas obtenidas para el modelo Dueling IQL

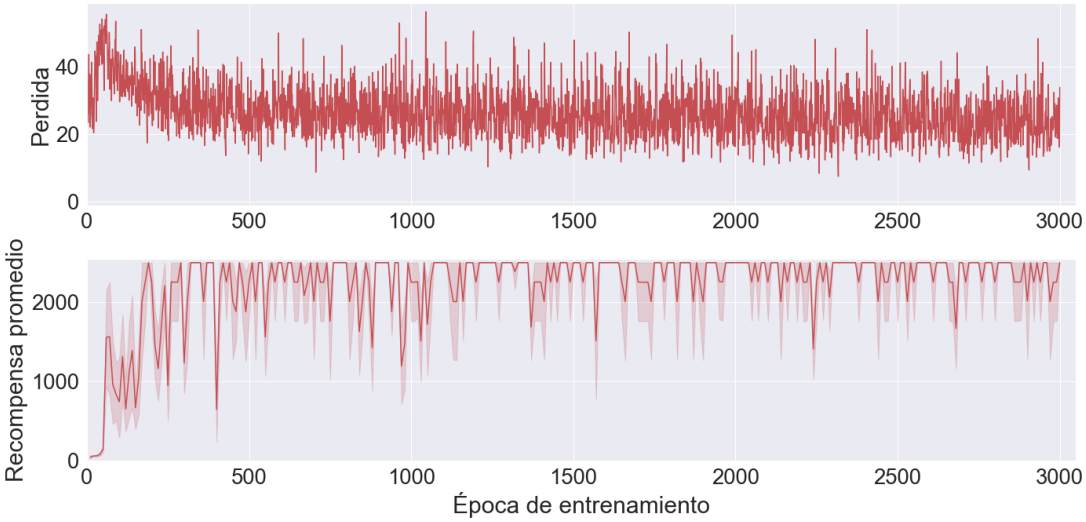

Figura 4.40: Pérdida y recompensas obtenidas para el modelo Double Dueling IQL

Para esta configuración de recompensa customizada| del conjunto de datos, los resultados de los gráficos resultan similares a los obtenidos por este mismo algoritmo, cuando empleó datos generados con la fnción de recompensa default.

4.1.2.6. Batch-Constrained Deep Q-Learning

En las figuras 4.41, 4.42, 4.43 y 4.44 se visualizan las funciones de pérdida y recompensas obtenidas de entrenar y evaluar los agentes que implementaron el algoritmo Batch-Constrained Deep Q-Learning, junto con sus variantes Double BCQ, Dueling Double BCQ y Double Dueling BCQ respectivamente, para el experimento de balancear el péndulo invertido.

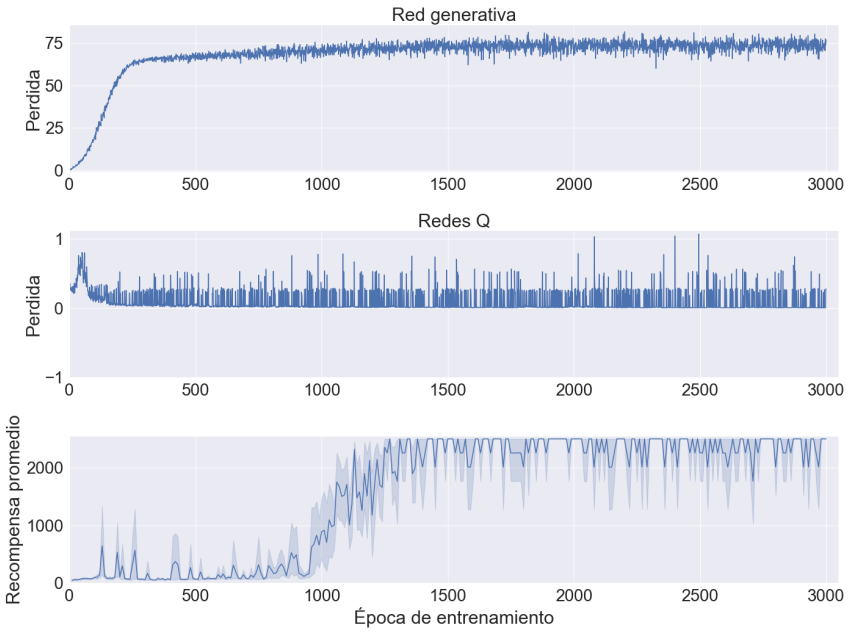

Figura 4.41: Pérdidas y recompensas obtenidas para el modelo BCQ

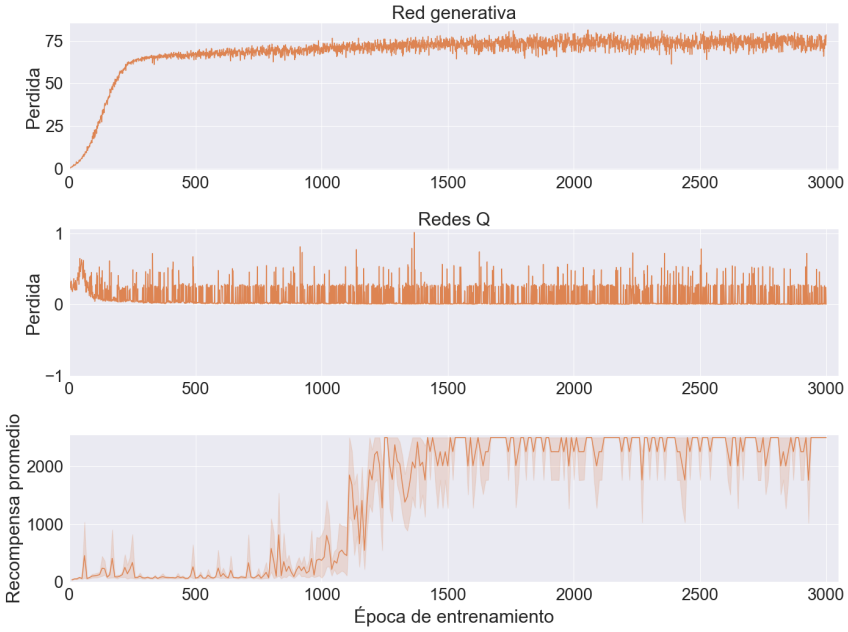

Figura 4.42: Pérdidas y recompensas obtenidas para el modelo Double BCQ

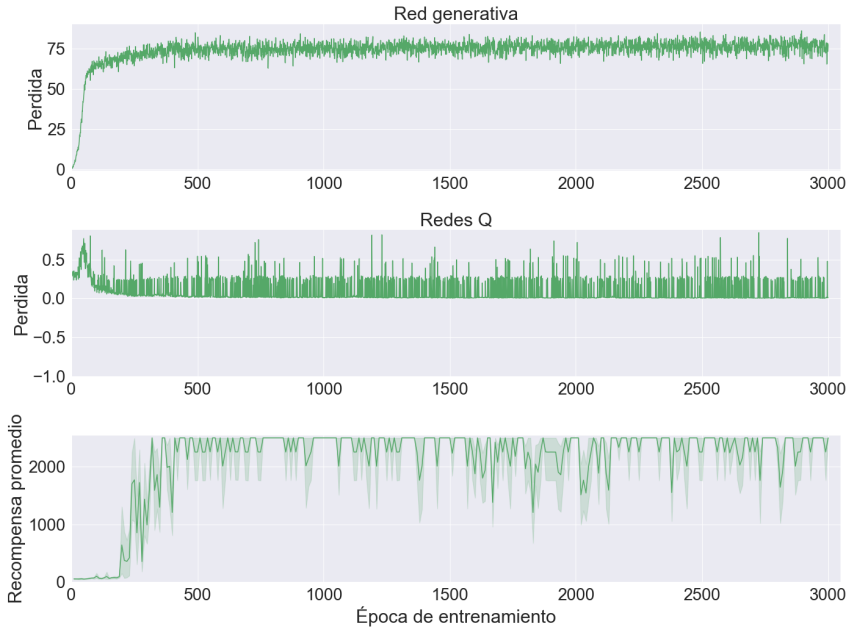

Figura 4.43: Pérdidas y recompensas obtenidas para el modelo Dueling BCQ

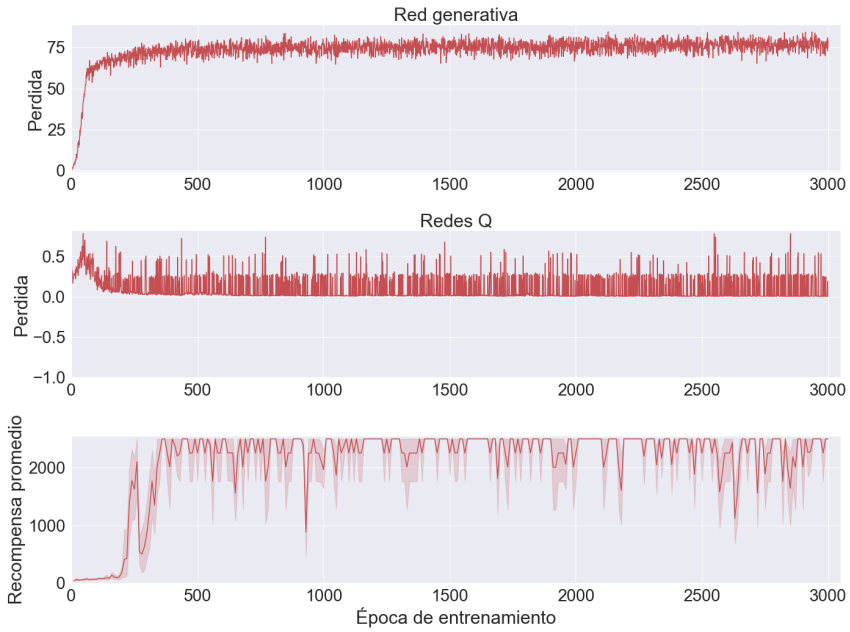

Figura 4.44: Pérdidas y recompensas obtenidas para el modelo Double Dueling BCQ

Los resultados de los gráficos anteriores, reiteran que para este algoritmo, el uso de *Dueling Networks* reduce la velocidad de aprendizaje.

4.1.3. Entrenamiento con base de datos generada por política mixta y función de recompensa default

4.1.3.1. Deep Q-Learning

En las figuras 4.45, 4.46, 4.47 y 4.48 se visualizan las funciones de pérdida y recompensas obtenidas de entrenar y evaluar los agentes que implementaron el algoritmo Deep Q-Learning, junto con sus variantes Double DQL, Dueling DQL y Double Dueling DQL, para el experimento de balancear el péndulo invertido.

Figura 4.45: Pérdida y recompensas obtenidas para el modelo DQL

Figura 4.46: Pérdida y recompensas obtenidas para el modelo Double DQL

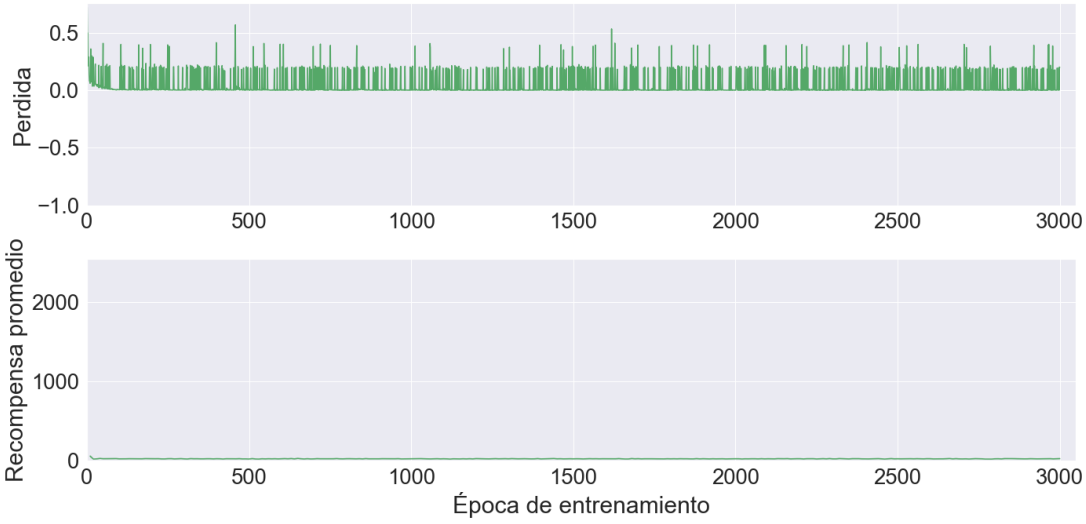

Figura 4.47: Pérdida y recompensas obtenidas para el modelo Dueling DQL

Figura 4.48: Pérdida y recompensas obtenidas para el modelo Double Dueling DQL

Los resultados muestran nuevamente, que este algoritmo fracasa al implementarse en un proceso de aprendizaje *offline*.

4.1.3.2. Deep Q-Learning from Demostration

En las figuras 4.49, 4.50, 4.51 y 4.52 se visualizan las funciones de pérdida y recompensas obtenidas de entrenar y evaluar los agentes que implementaron el algoritmo Deep Q-Learning from Demostration, junto con sus variantes Double DQfD, Dueling DQfD y Double Dueling DQfD respectivamente, para el experimento de balancear el péndulo invertido.

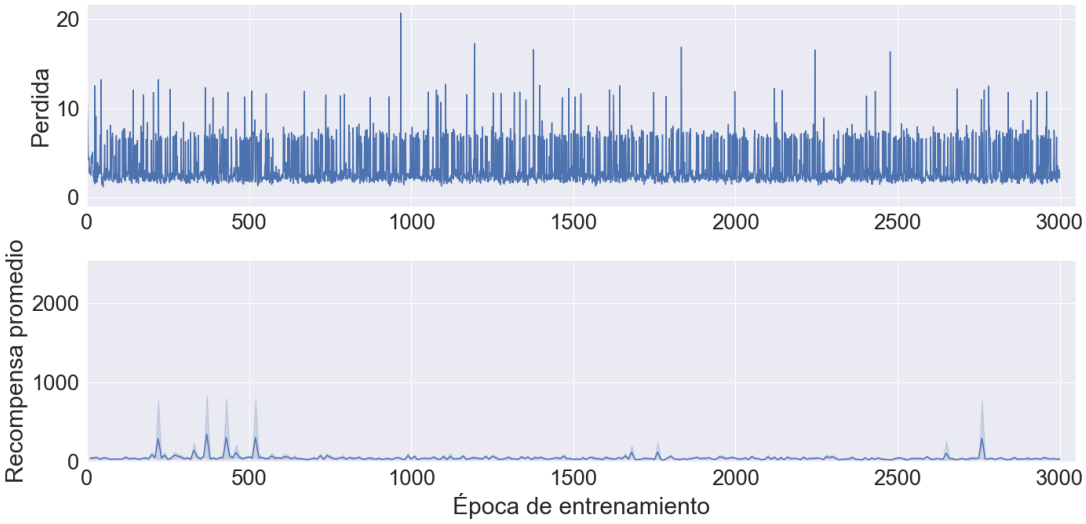

Figura 4.49: Pérdida y recompensas obtenidas para el modelo DQfD

Figura 4.50: Pérdida y recompensas obtenidas para el modelo Double DQfD

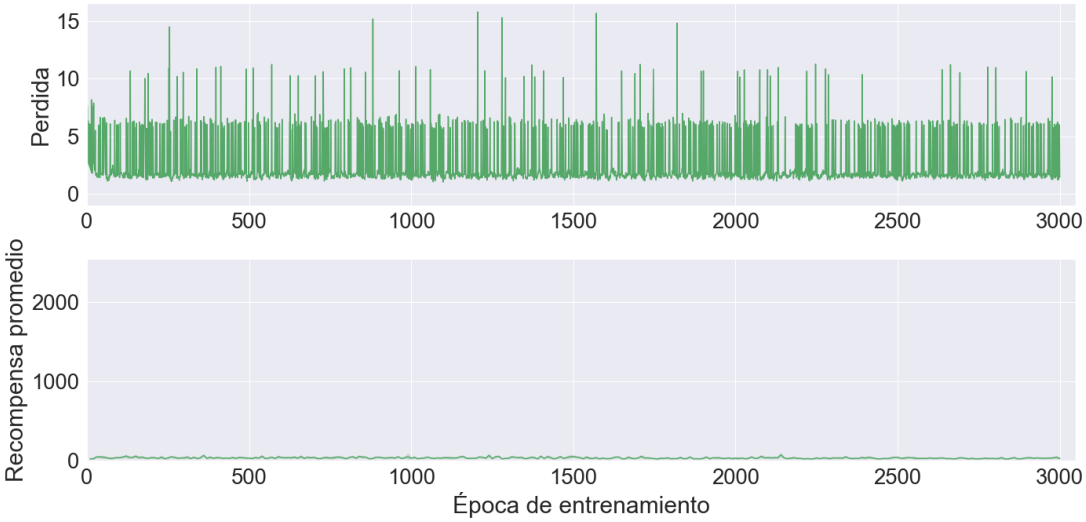

Figura 4.51: Pérdida y recompensas obtenidas para el modelo Dueling DQfD

Figura 4.52: Pérdida y recompensas obtenidas para el modelo Double Dueling DQfD

Los resultados de los gráficos anteriores, muestran que este algoritmo no logra ser eficaz, si los datos que emplea para entrenar no son explcusivamente de origen experto.

4.1.3.3. Neural Fitted Q Iteration

En las figuras 4.53 y 4.54 se visualizan las funciones de pérdida y recompensas obtenidas de entrenar y evaluar los agentes que implementaron el algoritmo Neural Fitted Q Iteration, junto con su variante Double NFQ, para el experimento de balancear el péndulo invertido.

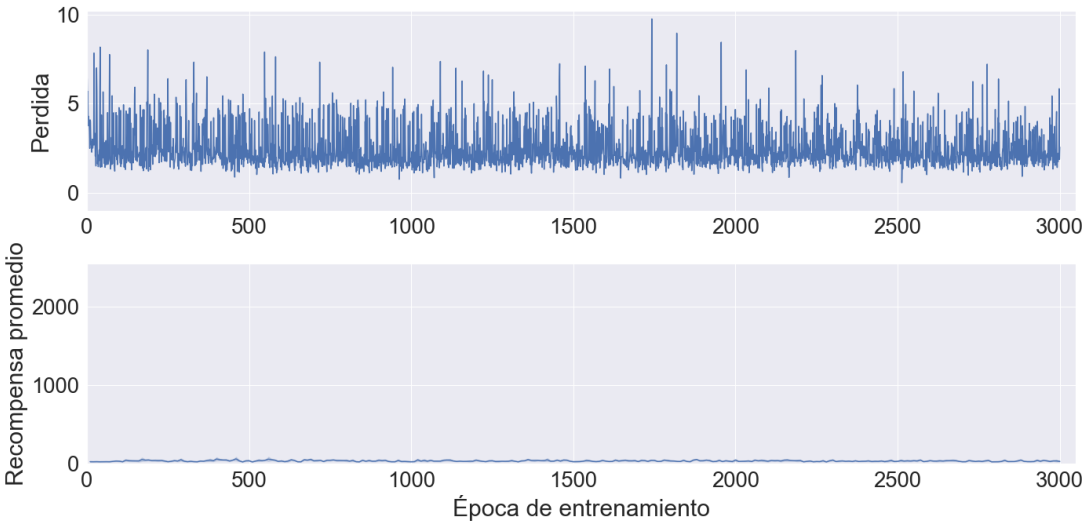

Figura 4.53: Pérdida y recompensas obtenidas para el modelo NFQ

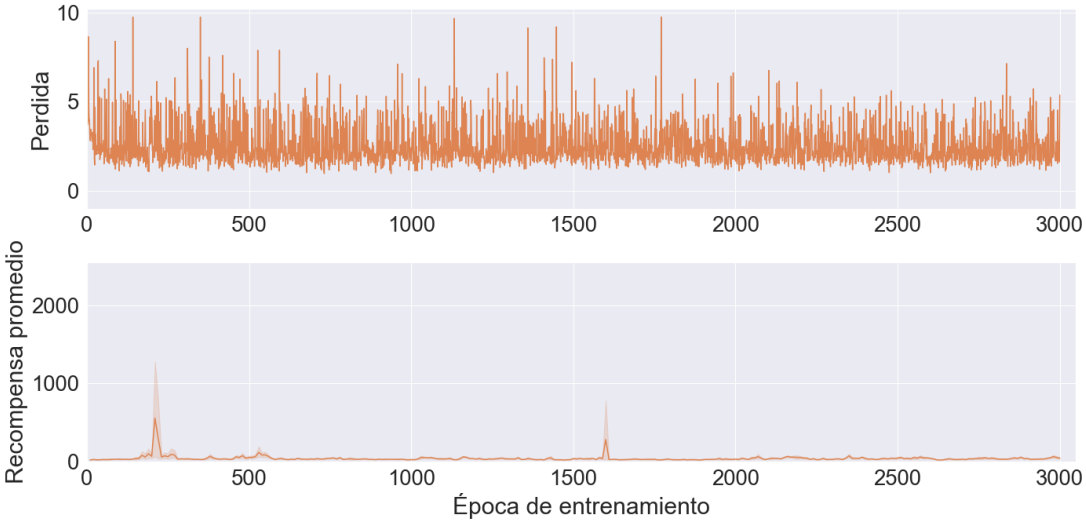

Figura 4.54: Pérdida y recompensas obtenidas para el modelo Double NFQ

Los resultados muestran, que este algoritmo también se ve afectado, por emplear datos que no presenten exclusivamente políticas óptimas.

4.1.3.4. Conservative Q-Learning

En las figuras 4.55, 4.56, 4.57 y 4.58 se visualizan las funciones de pérdida y recompensas obtenidas de entrenar y evaluar los agentes que implementaron el algoritmo Conservative Q-Learning, junto con sus variantes Double CQL, Dueling CQL y Double Dueling CQL respectivamente, para el experimento de balancear el péndulo invertido.

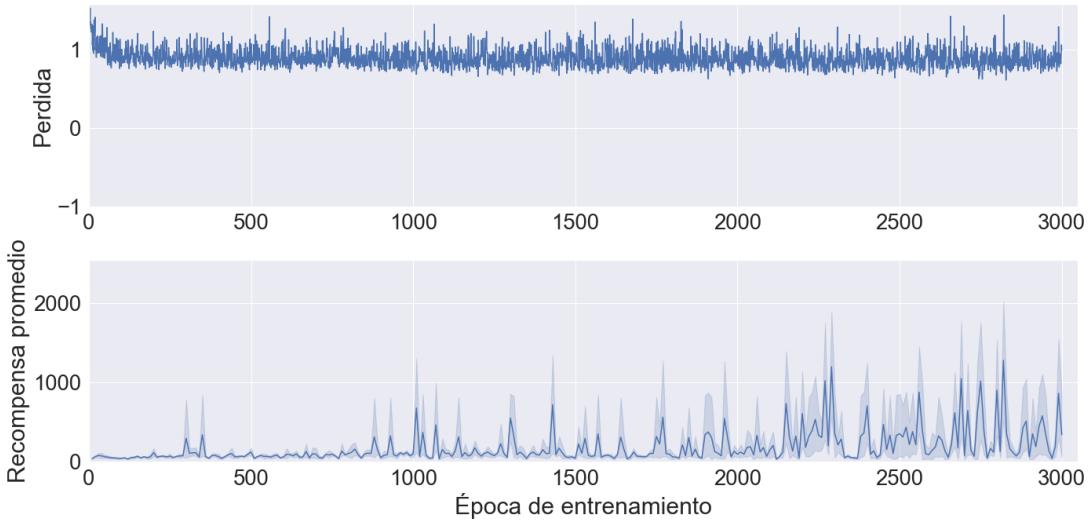

Figura 4.55: Pérdida y recompensas obtenidas para el modelo CQL

Figura 4.56: Pérdida y recompensas obtenidas para el modelo Double CQL

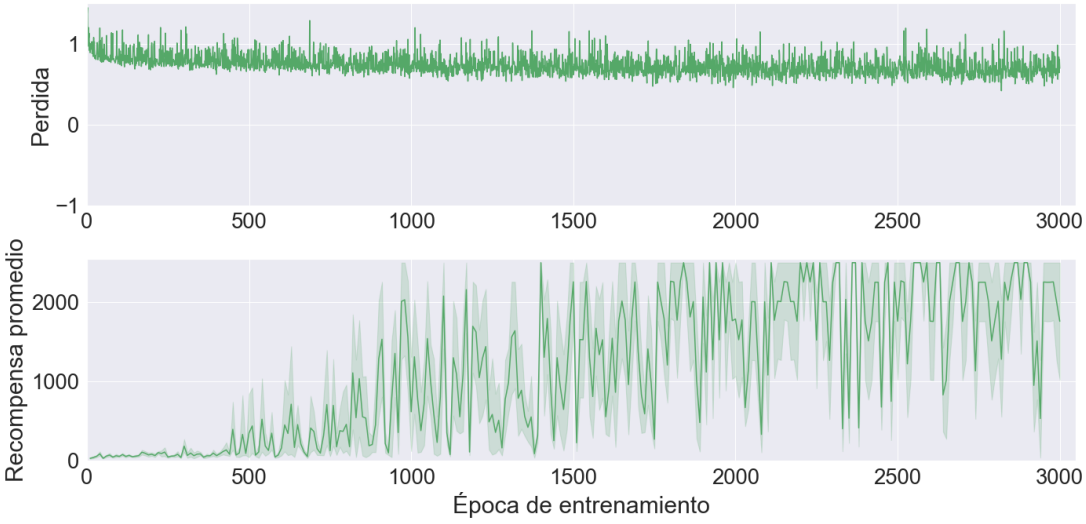

Figura 4.57: Pérdida y recompensas obtenidas para el modelo Dueling CQL

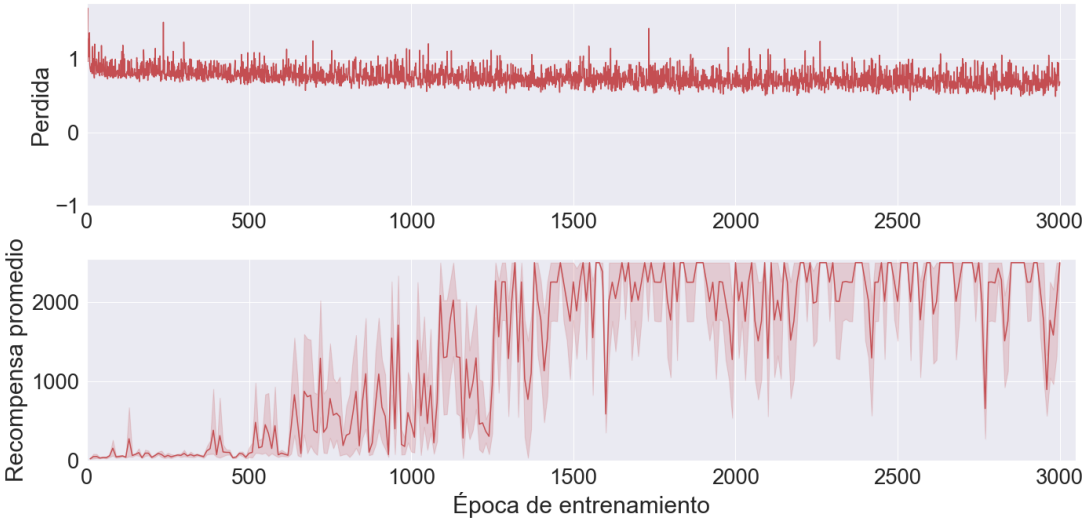

Figura 4.58: Pérdida y recompensas obtenidas para el modelo Double Dueling CQL

Los resultados obtenidos, muestran que este algoritmo se ve fuertemente afectado, al aprender con datos que no contengan exclusivamente políticas óptimas. Sin embargo, se aprecia que el empleo de *Dueling Networks* permite superar este inconveniente.

4.1.3.5. Implicit Q-Learning

En las figuras 4.59, 4.60, 4.61 y 4.62 se visualizan las funciones de pérdida y recompensas obtenidas de entrenar y evaluar los agentes que implementaron el algoritmo Implicit Q-Learning, junto con sus variantes Double IQL, Dueling IQL y Double Dueling IQL respectivamente, para el experimento de balancear el péndulo invertido.

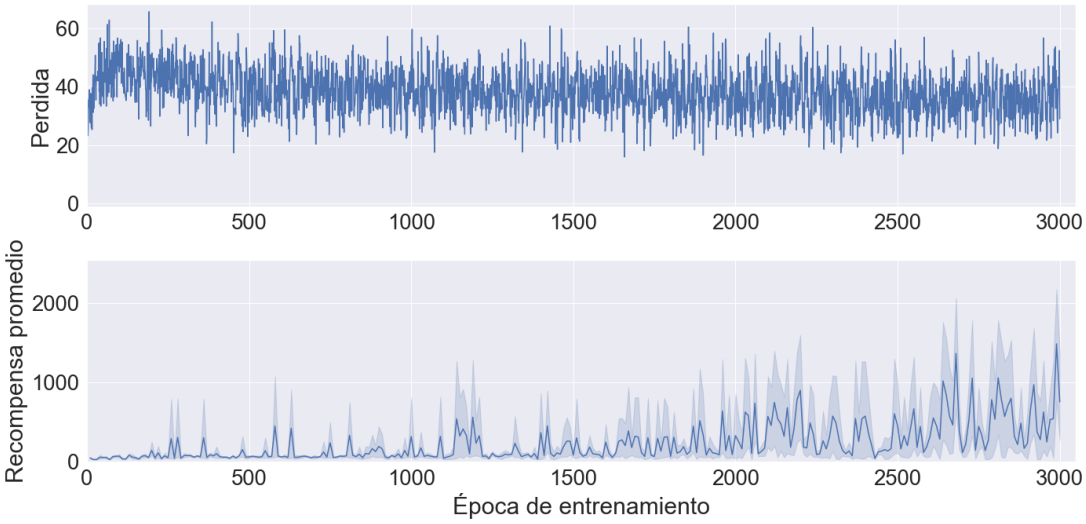

Figura 4.59: Pérdida y recompensas obtenidas para el modelo IQL

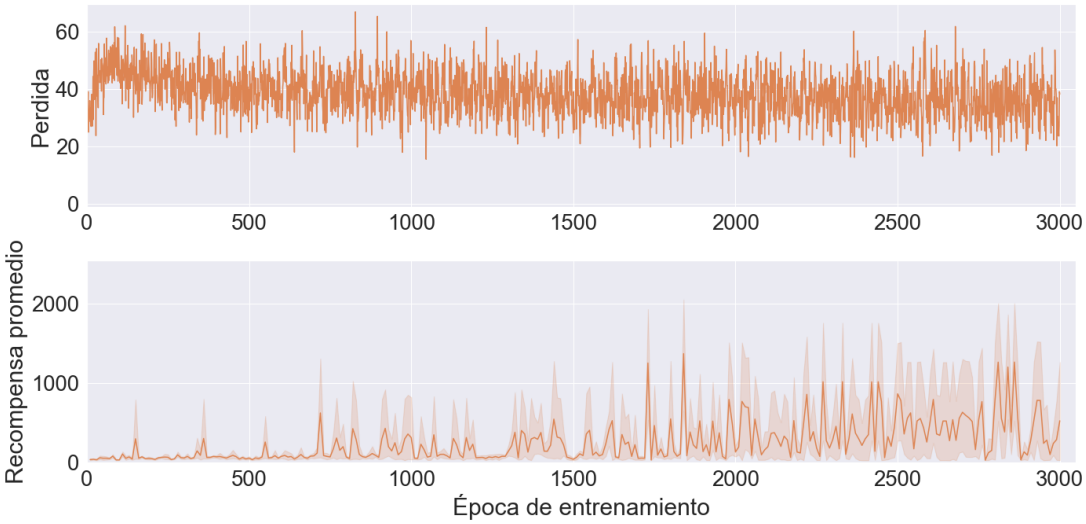

Figura 4.60: Pérdida y recompensas obtenidas para el modelo Double IQL

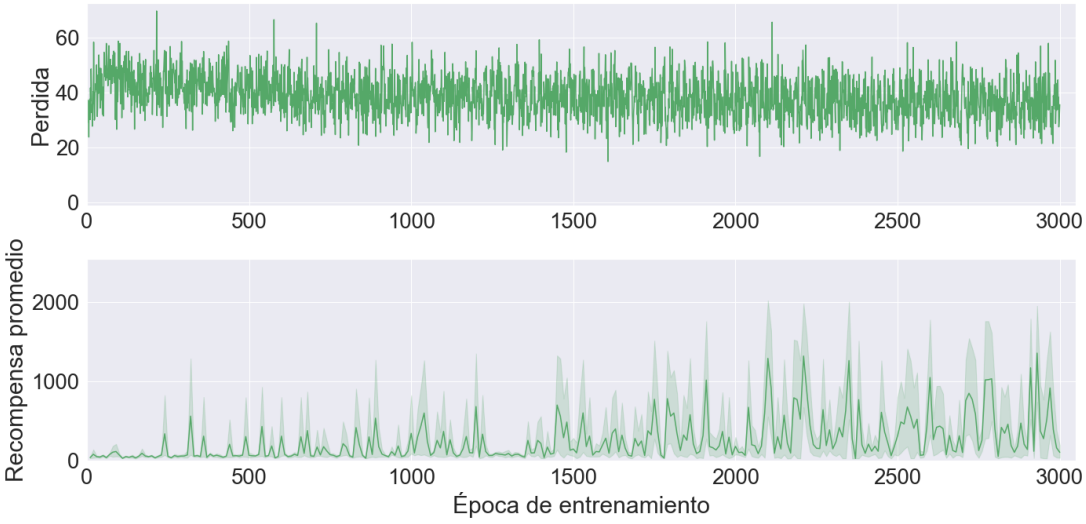

Figura 4.61: Pérdida y recompensas obtenidas para el modelo Dueling IQL

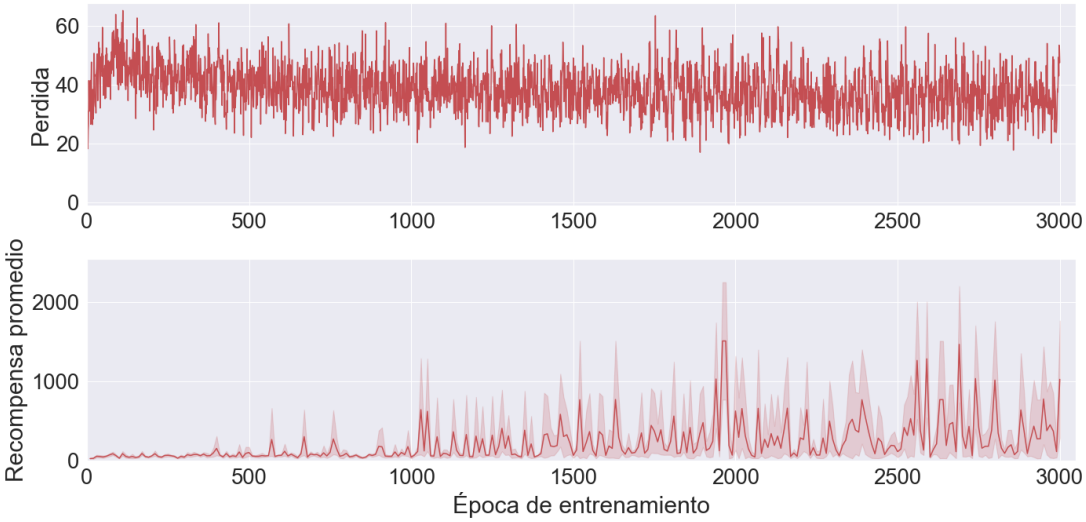

Figura 4.62: Pérdida y recompensas obtenidas para el modelo Double Dueling IQL

Los resultados obtenidos, muestran que este algoritmo se ve aún más afectado que CQL, cuando es entrenado con datos que puedan contener políticas sub-óptimas.

4.1.3.6. Batch-Constrained Deep Q-Learning

En las figuras 4.63, 4.64, 4.65 y 4.66 se visualizan las funciones de pérdida y recompensas obtenidas de entrenar y evaluar los agentes que implementaron el algoritmo Batch-Constrained Deep Q-Learning, junto con sus variantes Double BCQ, Dueling Double BCQ y Double Dueling BCQ respectivamente, para el experimento de balancear el péndulo invertido.

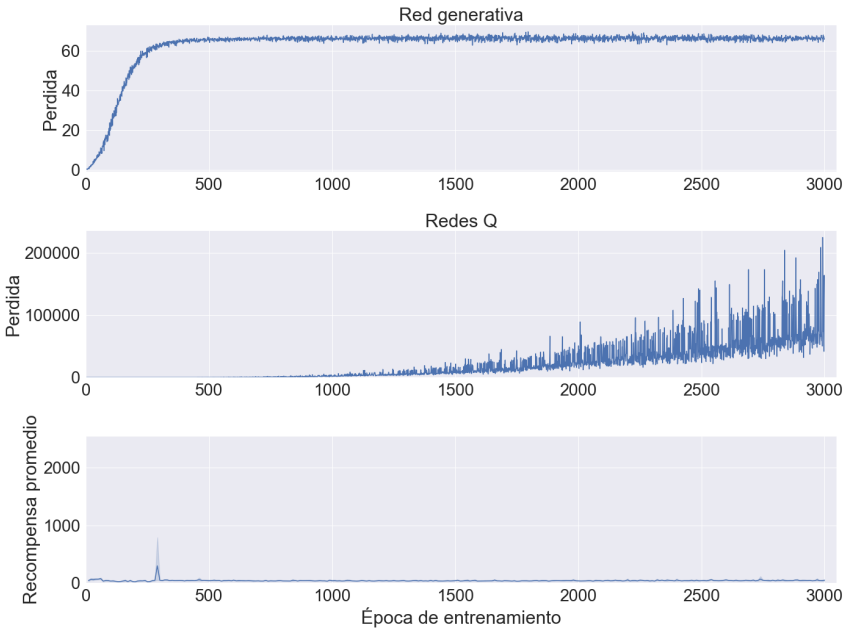

Figura 4.63: Pérdidas y recompensas obtenidas para el modelo BCQ

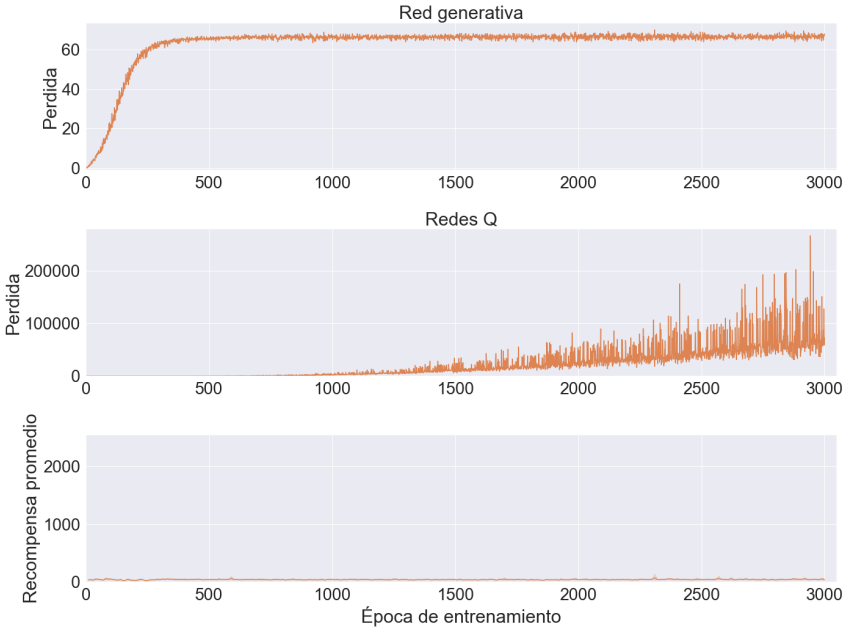

Figura 4.64: Pérdidas y recompensas obtenidas para el modelo Double BCQ

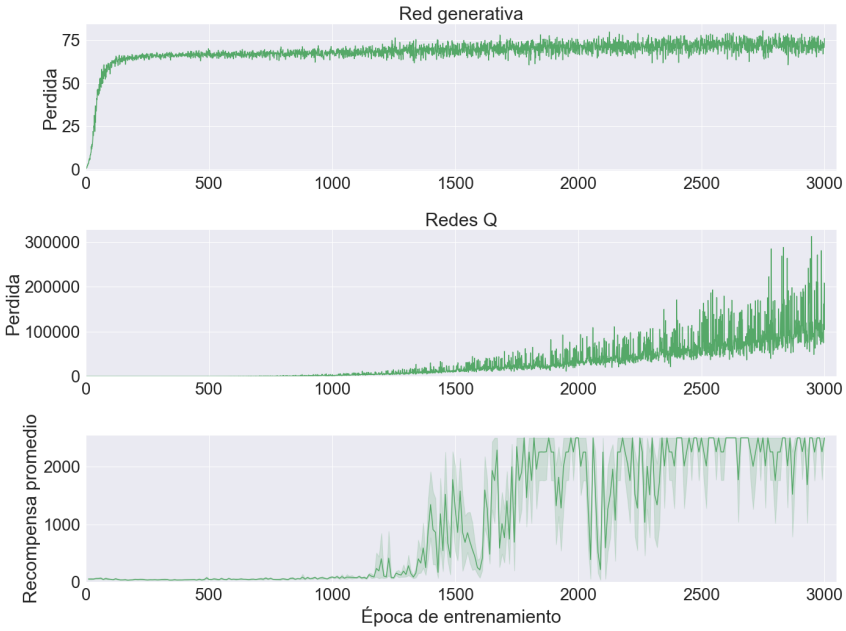

Figura 4.65: Pérdidas y recompensas obtenidas para el modelo Dueling BCQ

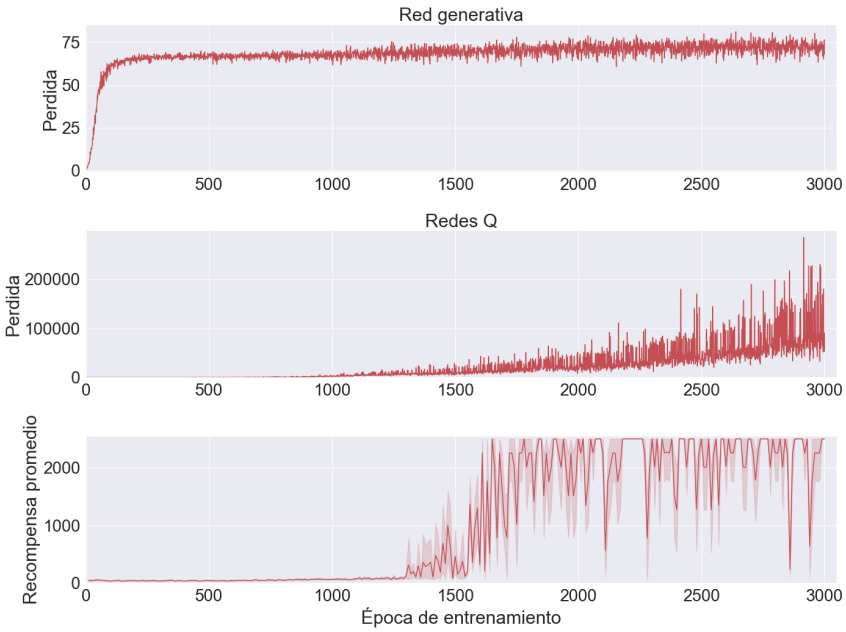

Figura 4.66: Pérdidas y recompensas obtenidas para el modelo Double Dueling BCQ

Los resultados muestran, que este algoritmo se ve aún más afectado que CQL y IQL, al entrenar con datos que no sean exclusivamente generados por una política óptima.

4.1.4. Entrenamiento con base de datos generada por política mixta y función de recompensa customizada

4.1.4.1. Deep Q-Learning

En las figuras 4.67, 4.68, 4.69 y 4.70 se visualizan las funciones de pérdida y recompensas obtenidas de entrenar y evaluar los agentes que implementaron el algoritmo Deep Q-Learning, junto con sus variantes Double DQL, Dueling DQL y Double Dueling DQL respectivamente, para el experimento de balancear el péndulo invertido.

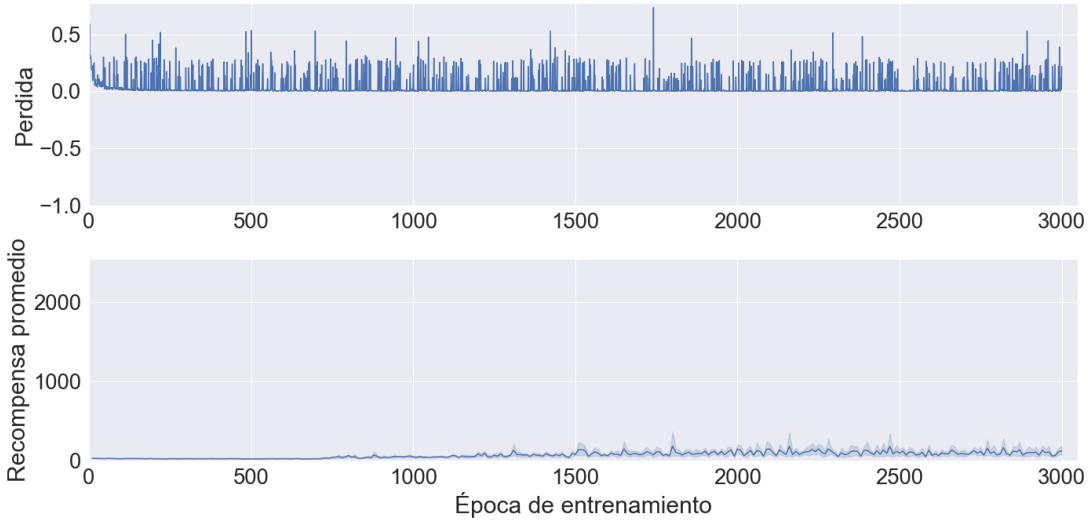

Figura 4.67: Pérdida y recompensas obtenidas para el modelo DQL

Figura 4.68: Pérdida y recompensas obtenidas para el modelo Double DQL

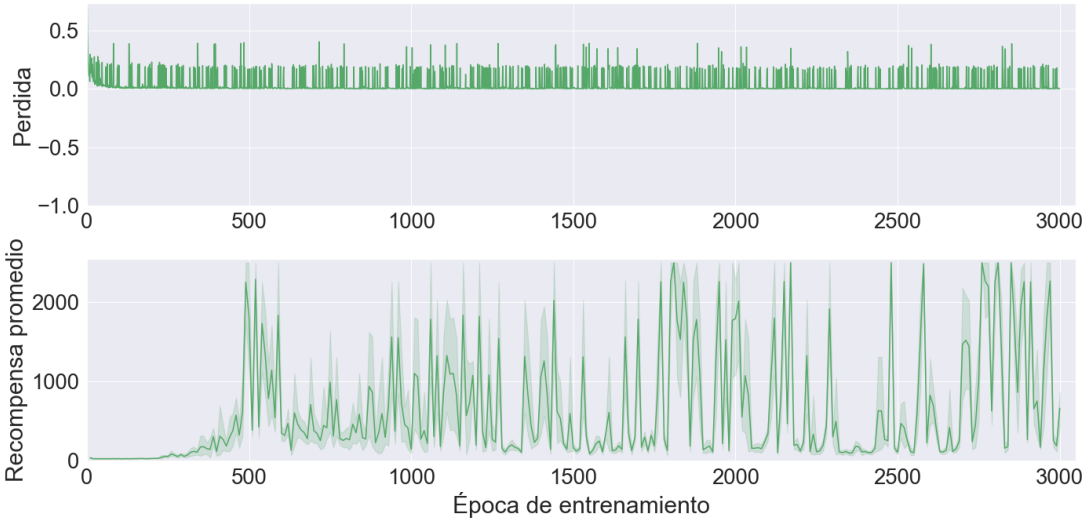

Figura 4.69: Pérdida y recompensas obtenidas para el modelo Dueling DQL

Figura 4.70: Pérdida y recompensas obtenidas para el modelo Double Dueling DQL

Los resultados muestran nuevamente el mal desempeño de este algoritmo al entrenar de manera *offline*, salvo por la particularidad de que, para la configuración empleando una *Dueling Network*, logra llegar a alcanzar de manera inesperada, un desempeño óptimo.

4.1.4.2. Deep Q-Learning from Demostration

En las figuras 4.71, 4.72, 4.73 y 4.74 se visualizan las funciones de pérdida y recompensas obtenidas de entrenar y evaluar los agentes que implementaron el algoritmo Deep Q-Learning from Demostration, junto con sus variantes Double DQfD, Dueling DQfD y Double Dueling DQfD respectivamente, para el experimento de balancear el péndulo invertido.

Figura 4.71: Pérdida y recompensas obtenidas para el modelo DQfD

Figura 4.72: Pérdida y recompensas obtenidas para el modelo Double DQfD

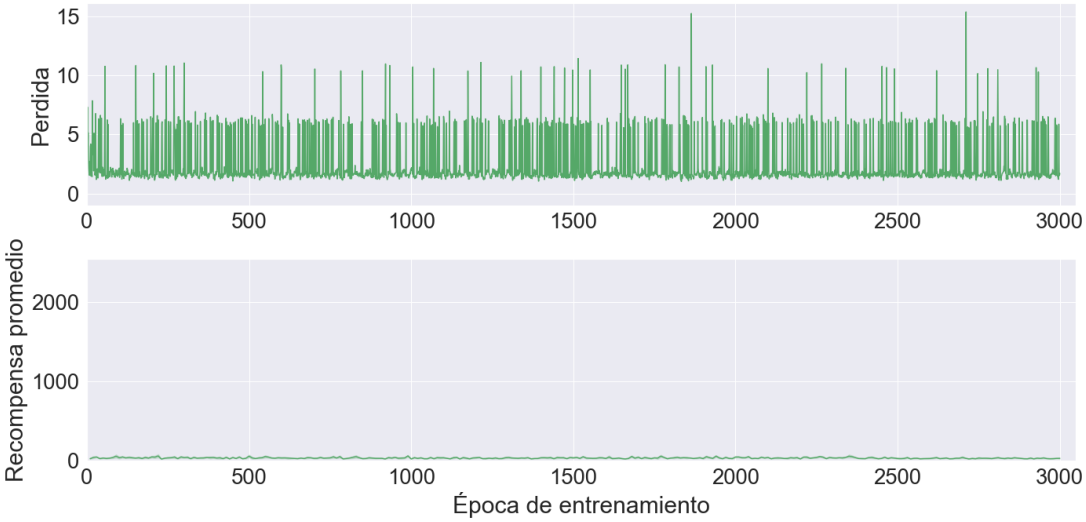

Figura 4.73: Pérdida y recompensas obtenidas para el modelo Dueling DQfD

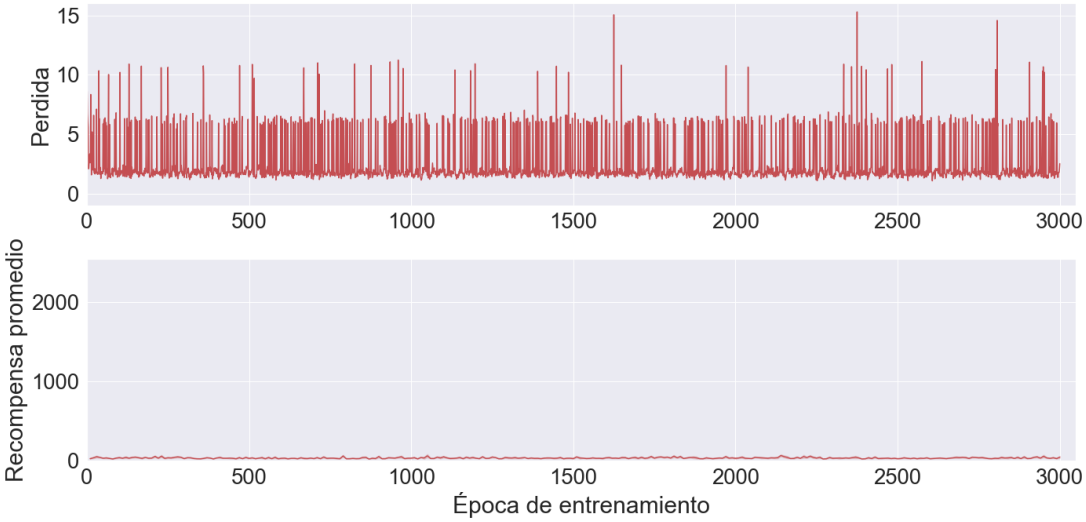

Figura 4.74: Pérdida y recompensas obtenidas para el modelo Double Dueling DQfD

Los resultados obtenidos reiteran que, este algoritmo se ve drásticamente afectado, si es entrenado con datos que no son exclusivamente de origen experto.

4.1.4.3. Neural Fitted Q Iteration

En las figuras 4.75 y 4.76 se visualizan las funciones de pérdida y recompensas obtenidas de entrenar y evaluar los agentes que implementaron el algoritmo Neural Fitted Q Iteration, junto con su variante Double NFQ, para el experimento de balancear el péndulo invertido.

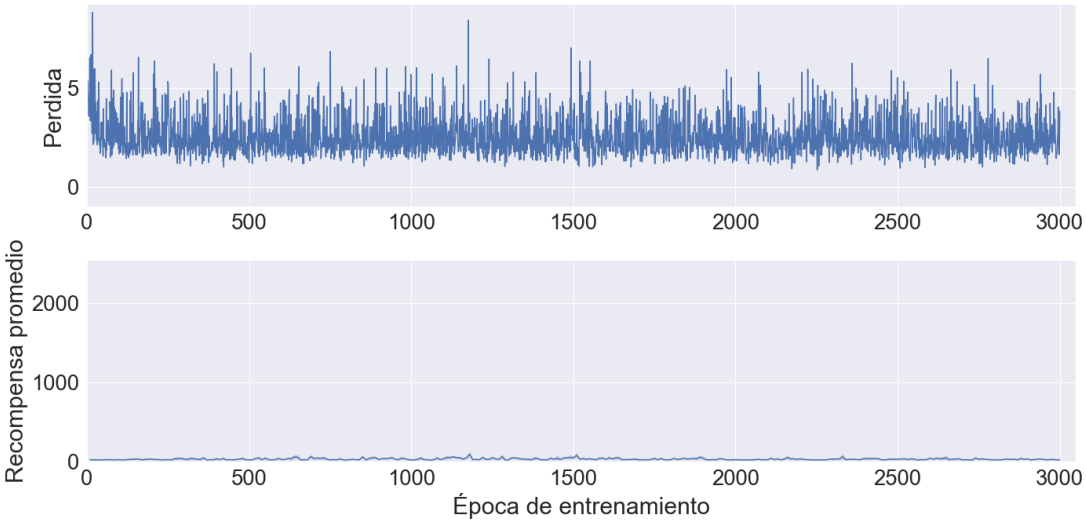

Figura 4.75: Pérdida y recompensas obtenidas para el modelo NFQ

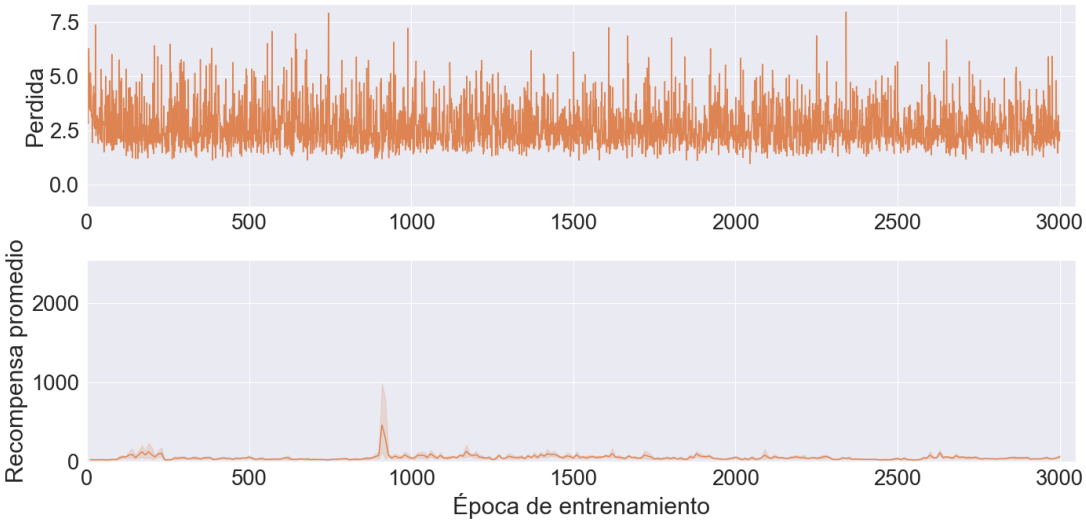

Figura 4.76: Pérdida y recompensas obtenidas para el modelo Double NFQ

Los resultados de este algoritmo muestran que, este también necesita aprender a partir de datos generados de experiencias expertas.

4.1.4.4. Conservative Q-Learning

En las figuras 4.77, 4.78, 4.79 y 4.80 se visualizan las funciones de pérdida y recompensas obtenidas de entrenar y evaluar los agentes que implementaron el algoritmo Conservative Q-Learning, junto con sus variantes Double CQL, Dueling CQL y Double Dueling CQL respectivamente, para el experimento de balancear el péndulo invertido.

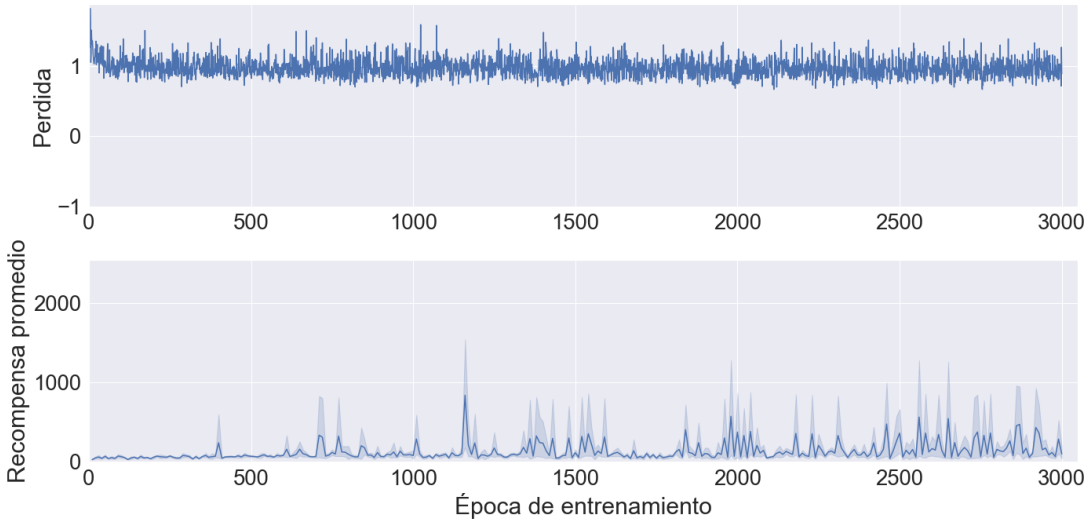

Figura 4.77: Pérdida y recompensas obtenidas para el modelo CQL

Figura 4.78: Pérdida y recompensas obtenidas para el modelo Double CQL

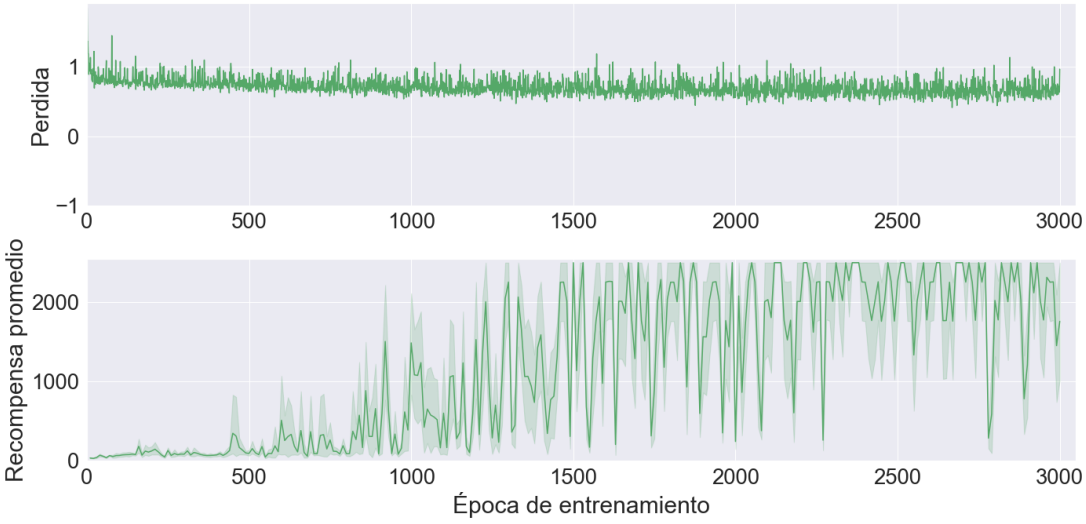

Figura 4.79: Pérdida y recompensas obtenidas para el modelo Dueling CQL

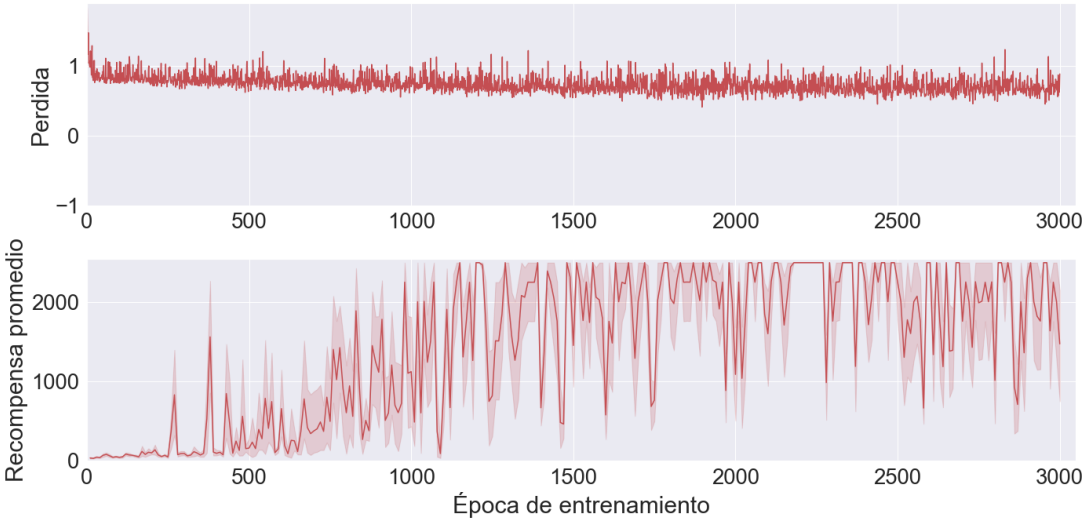

Figura 4.80: Pérdida y recompensas obtenidas para el modelo Double Dueling CQL

Los resultados presentados en los gráficos, nuevamente reiteran que este algoritmo puede aprovechar las *Dueling Networks*, para sortear los efectos de políticas subóptimas en los datos de entrenamiento. Aún así, se vuelve a apreciar un posible sobreajuste a los datos, por la oscilación de la recompensa obtenida con el avanze de las épocas.

4.1.4.5. Implicit Q-Learning

En las figuras 4.81, 4.82, 4.83 y 4.84 se visualizan las funciones de pérdida y recompensas obtenidas de entrenar y evaluar los agentes que implementaron el algoritmo Implicit Q-Learning, junto con sus variantes Double IQL, Dueling IQL y Double Dueling IQL respectivamente, para el experimento de balancear el péndulo invertido.

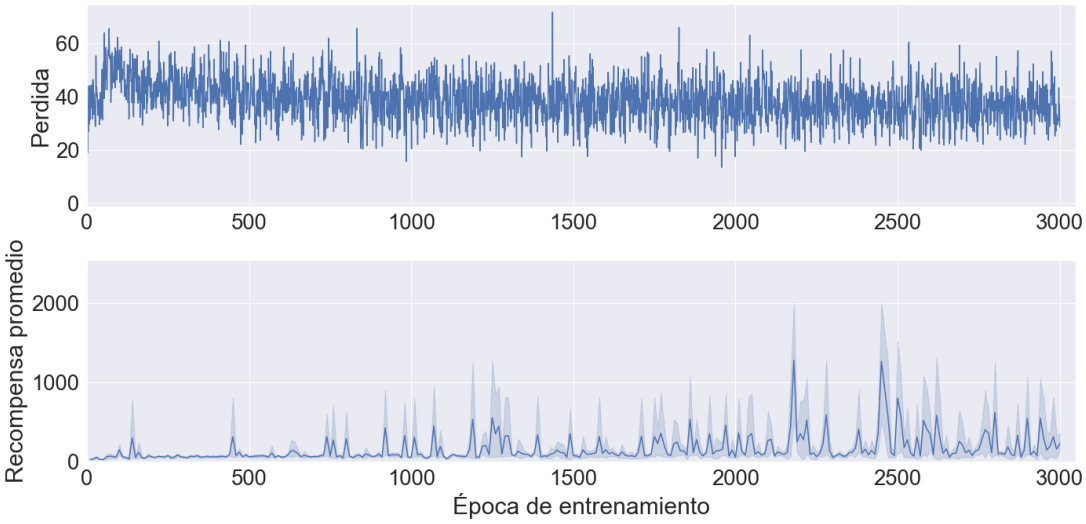

Figura 4.81: Pérdida y recompensas obtenidas para el modelo IQL

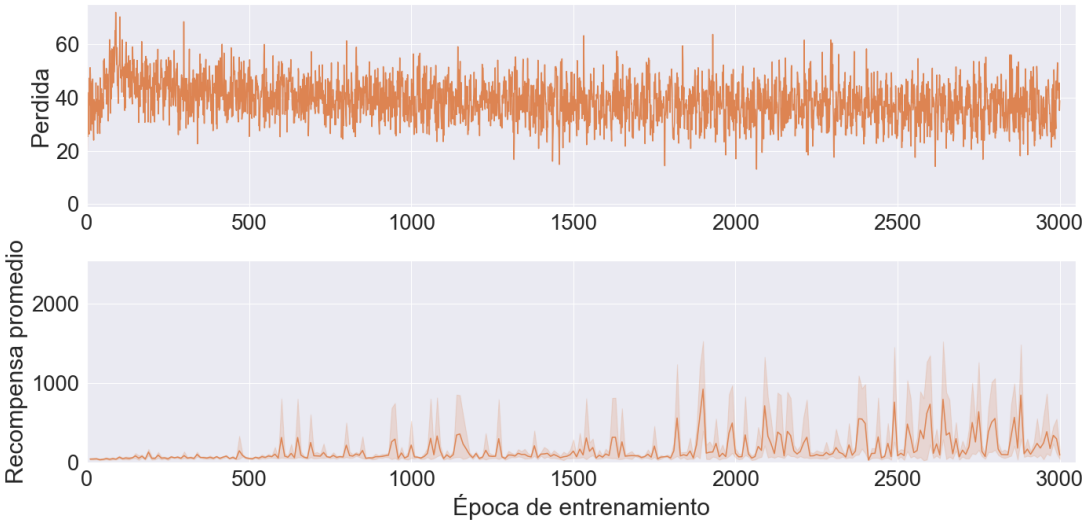

Figura 4.82: Pérdida y recompensas obtenidas para el modelo Double IQL

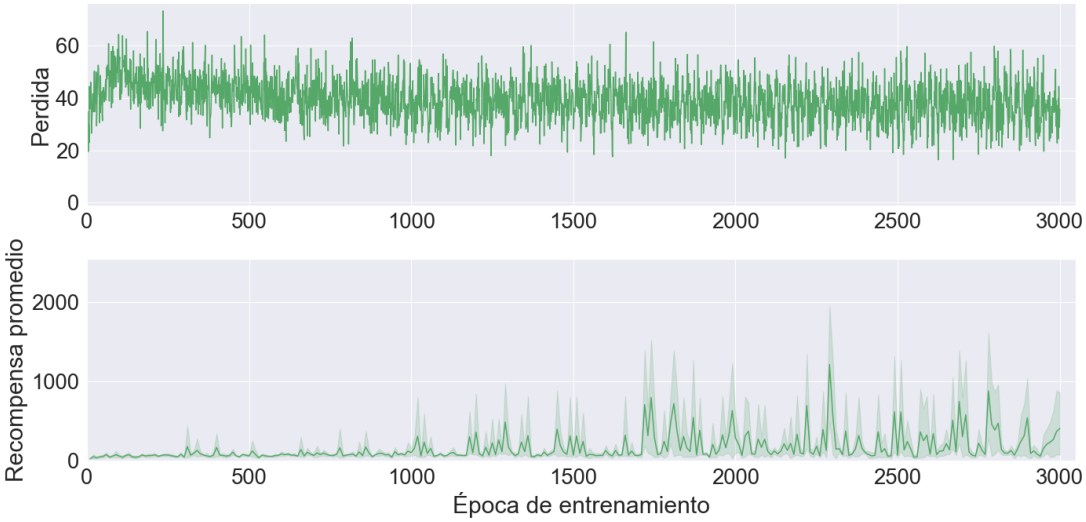

Figura 4.83: Pérdida y recompensas obtenidas para el modelo Dueling IQL

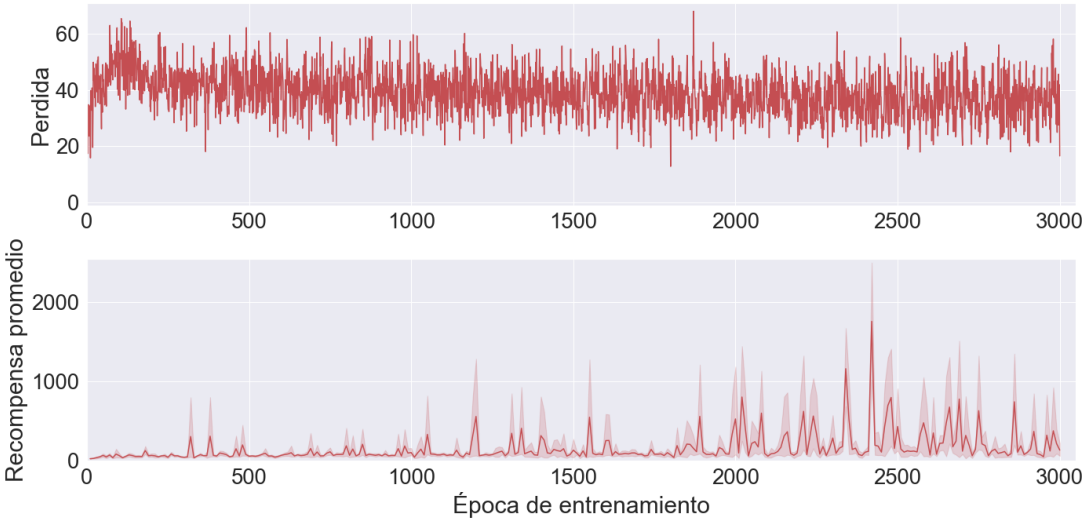

Figura 4.84: Pérdida y recompensas obtenidas para el modelo Double Dueling IQL

Los resultados reiteran que, este algoritmo requiere datos exclusivamente de origen experto, para aprender de manera óptima.

4.1.4.6. Batch-Constrained Deep Q-Learning

En las figuras 4.85, 4.86, 4.87 y 4.88 se visualizan las funciones de pérdida y recompensas obtenidas de entrenar y evaluar los agentes que implementaron el algoritmo Batch-Constrained Deep Q-Learning, junto con sus variantes Double BCQ, Dueling Double BCQ y Double Dueling BCQ respectivamente, para el experimento de balancear el péndulo invertido.

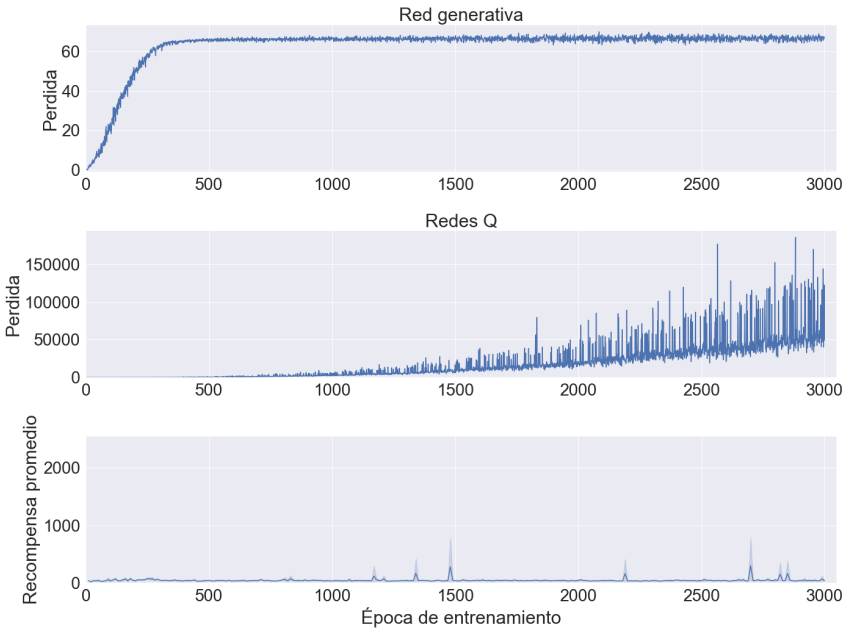

Figura 4.85: Pérdidas y recompensas obtenidas para el modelo BCQ

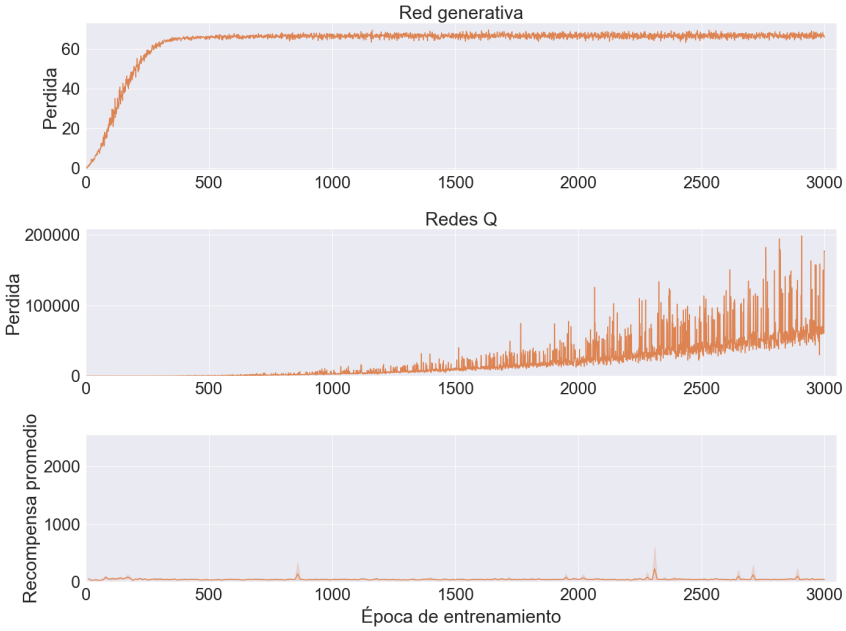

Figura 4.86: Pérdidas y recompensas obtenidas para el modelo Double BCQ

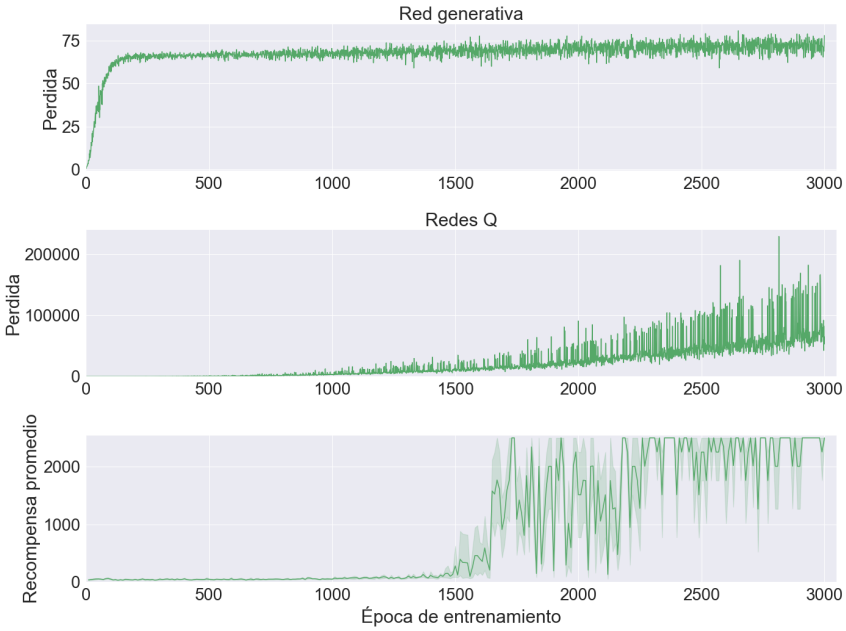

Figura 4.87: Pérdidas y recompensas obtenidas para el modelo Dueling BCQ

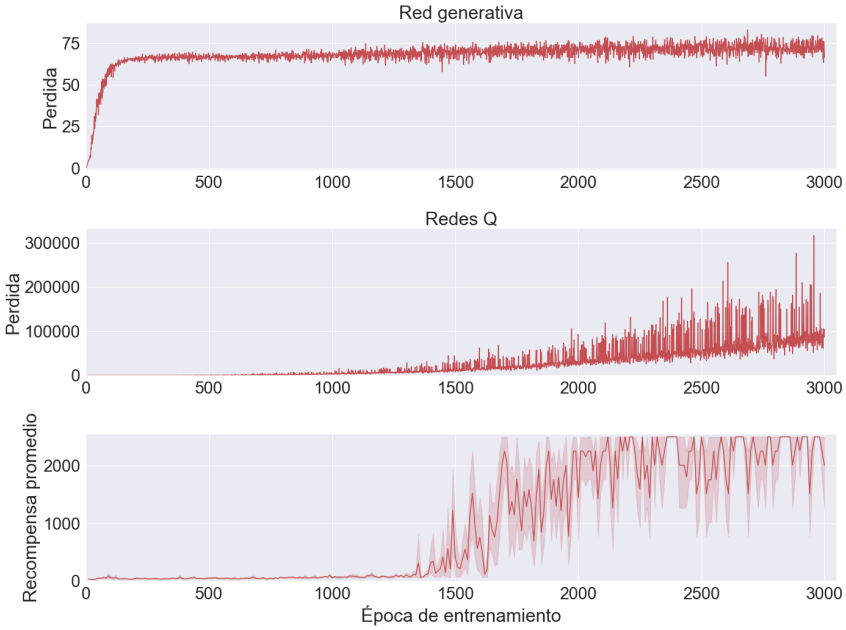

Figura 4.88: Pérdidas y recompensas obtenidas para el modelo Double Dueling BCQ

Nuevamente los resultados arrojan que, las *Dueling Networks* son útiles cuando se trabaja con datos, que pueden contener políticas subóptimas en ellos.

4.1.5. Análisis y comparativa de algoritmos

A fin de de facilitar el análisis de los resultados obtenidos en el experimento de balancear el péndulo, se realiza en las tablas 4.1, 4.2, 4.3, 4.4, 4.5 y 4.6 un resumen de los resultados de cada algoritmo y sus variantes, para los distintos conjuntos de datos empleados, presentando la época tras la que alcanzan su mejor desempeño, junto con la recompensa promedio obtenida en esa evaluación del algoritmo. Tomando en cuenta que las evaluaciones realizadas en el ambiente virtual tenían una extensión máxima de 2500 pasos, lo que definía la recompensa máxima posible de 2500, se consideró para los casos en que un algoritmo llegó a este máximo en repetidas instancias, presentar la primera época tras la que lo hizo.

Tabla 4.1: Resumen de resultados del experimento de balancear el péndulo invertido, para el algoritmo Deep Q-Learning y sus variantes

Algoritmo	Deep Q-Learning				Double Deep Q-Learning				Dueling Deep Q-Learning				Double Dueling Deep Q-Learning			
Función de recompensa	Default		Customizada		Default		Customizada		Default		Customizada		Default		Customizada	
Tipo de data	Experta	Mixta	Experta	Mixta	Experta	Mixta	Experta	Mixta	Experta	Mixta	Experta	Mixta	Experta	Mixta	Experta	Mixta
Época de mayor recompensa promedio	40	10	30	1800	10	10	60	50	10	10	10	1810	40	40	20	60
Mayor recompensa promedio	40.5	38.2	44.5	183.6	40.0	42.5	37.7	28.3	68.6	57.0	84.5	2500.0	70.3	63.4	83.6	54.2

Los resultados de la 4.1, reflejan que el algoritmo DQL no fue desarrollado para implementarse de manera *offline*, por el insatisfactorio desempeño que obtuvo. No obstante resulta curioso que, para una configuración del algoritmo y la base de datos en particular, logra de alguna manera obtener un desempeño óptimo. Una conjetura de porqué se da este caso particular, es que la mezcla de emplear la función de recompensa continua, junto con poder aprender tanto de datos de comportamientos óptimos como deficientes, permitieron aprovechar las capacidades de generalización, de las *Dueling Networks*.

Tabla 4.2: Resumen de resultados del experimento de balancear el péndulo invertido, para el algoritmo Deep Q-Learning from Demonstration y sus variantes

Algoritmo	Deep Q-Learning from Demonstration				Double Deep Q-Learning from Demonstration				Dueling Deep Q-Learning from Demonstration				Double Dueling Deep Q-Learning from Demonstration			
Función de recompensa	Default		Customizada		Default		Customizada		Default		Customizada		Default		Customizada	
Tipo de data	Experta	Mixta	Experta	Mixta	Experta	Mixta	Experta	Mixta	Experta	Mixta	Experta	Mixta	Experta	Mixta	Experta	Mixta
Época de mayor recompensa promedio	150	370	130	660	120	990	240	2960	1730	2140	50	990	70	800	1620	2140
Mayor recompensa promedio	1780.1	346.3	2009.0	206.3	1781.9	303.0	1533.1	283.5	216.8	72.2	283.3	59.5	533.9	69.9	550.3	64.4

En cuanto al algoritmo DQfD, la Tabla 4.2 muestra que este algoritmo posee un mejor desempeño promedio que DQN. No obstante, los resultados obtenidos por este algoritmo, muestran que requiere ser entrenado exclusivamente con datos expertos, y que solo puede ser entrenado de manera *offline* por un breve periodo.

Tabla 4.3: Resumen de resultados del experimento de balancear el péndulo invertido, para el algoritmo Neural Fitted Q Iteration y su variante

Algoritmo	Neural Fitted Q Iteration				Double Neural Fitted Q Iteration			
Función de recompensa	Default		Customizada		Default		Customizada	
Tipo de data	Experta	Mixta	Experta	Mixta	Experta	Mixta	Experta	Mixta
Época de mayor recompensa promedio	2520	460	160	1180	330	210	460	910
Mayor recompensa promedio	1039.8	60.4	1036.9	91.6	1790.9	556.1	1055.8	462.7

La Tabla 4.3 muestra que, el algoritmo NFQ también requiere ser entrenado exclusivamente con datos expertos, y que su proceso de entrenamiento resulta ser más lento. También este algoritmo se beneficia notablemente, de emplear *Double Q-Networks*.

Tabla 4.4: Resumen de resultados del experimento de balancear el péndulo invertido, para el algoritmo Conservative Q-Learning y sus variantes

Algoritmo	Conservative Q-Learning				Double Conservative Q-Learning				Dueling Conservative Q-Learning				Double Dueling Conservative Q-Learning			
Función de recompensa	Default		Customizada		Default		Customizada		Default		Customizada		Default		Customizada	
Tipo de data	Experta	Mixta	Experta	Mixta	Experta	Mixta	Experta	Mixta	Experta	Mixta	Experta	Mixta	Experta	Mixta	Experta	Mixta
Época de mayor recompensa promedio	120	2820	140	1160	140	2630	150	1920	120	1400	130	1500	210	1320	250	1150
Mayor recompensa promedio	2500.0	1279.1	2500.0	838.0	2500.0	1511.1	2500.0	1778.8	2500.0	2500.0	2500.0	2500.0	2500.0	2500.0	2500.0	2500.0

En cuanto al algoritmo CQL, la Tabla 4.4 refleja como este requiere poco entrenamiento para alcanzar un gran desempeño. También la tabla remarca como, este algoritmo aprende mejor y más rápido con datos exclusivamente de origen experto. Se aprecia también que las *Dueling Networks* son útiles para este modelo, cuando aprende con datos que puedan contener políticas subóptimas.

Tabla 4.5: Resumen de resultados del experimento de balancear el péndulo invertido, para el algoritmo Implicit Q-Learning y sus variantes.

Algoritmo	Implicit Q-Learning				Double Implicit Q-Learning				Dueling Implicit Q-Learning				Double Dueling Implicit Q-Learning			
Función de recompensa	Default		Customizada		Default		Customizada		Default		Customizada		Default		Customizada	
Tipo de data	Experta	Mixta	Experta	Mixta	Experta	Mixta	Experta	Mixta	Experta	Mixta	Experta	Mixta	Experta	Mixta	Experta	Mixta
Época de mayor recompensa promedio	190	2990	160	2180	170	1840	130	1900	130	2930	220	2290	220	1960	190	2420
Mayor recompensa promedio	2500.0	1486.7	2500.0	1279.6	2500.0	1371.5	2500.0	923.9	2500.0	1357.6	2500.0	1213.2	2500.0	1511.2	2500.0	1758.6

La Tabla 4.5 muestra que el algoritmo IQL se desempeña de manera similar a CQL, salvo que este primero no logra aprovechar los beneficios de las *Dueling Networks* como el segundo.

Tabla 4.6: Resumen de resultados del experimento de balancear el péndulo invertido, para el algoritmo Batch-Constrained Q-Learning y sus variantes.

Algoritmo	Batch-Constrained Q-Learning				Double Batch Constrained Q-Learning				Dueling Batch Constrained Q-Learning				Double Dueling Batch Constrained Q-Learning			
Función de recompensa	Default		Customizada		Default		Customizada		Default		Customizada		Default		Customizada	
Tipo de data	Experta	Mixta	Experta	Mixta	Experta	Mixta	Experta	Mixta	Experta	Mixta	Experta	Mixta	Experta	Mixta	Experta	Mixta
Época de mayor recompensa promedio	1370	290	1270	2700	1420	2310	1250	2310	370	1780	320	1730	280	1650	360	2120
Mayor recompensa promedio	2500.0	296.8	2500.0	301.6	2500.0	70.5	2500.0	229.4	2500.0	2500.0	2500.0	2500.0	2500.0	2500.0	2500.0	2500.0

Los resultados de la Tabla 4.6 muestran que, si bien, resulta más lento en su velocidad de aprendizaje que CQL y IQL, es capaz de alcanzar similares buenos desempeños. También, al igual que CQL, le saca notorio provecho a las *Dueling Networks*, al aprender de datos de origen variado.

Los resultados anteriores muestran con claridad, que los esquemas *offline* aprenden de forma más efectiva y rápida, que los esquemas de carácter *online*, logrando alcanzar muy buenos desempeños en el balanceo del péndulo. También, se distinguen los efectos de la calidad de los datos que se utilizan, para el entrenamiento de los distintos agentes implementados, ya que en todos los algoritmos se apreció que, utilizar datos de origen exclusivamente experto, resultaba ser mucho más beneficioso. Por otro lado, se pudo observar que la función de recompensa empleada para generar los datos, no hacia cambios relevantes en el proceso de aprendizaje. Sobre esto solo se puede comentar que se requieren experimentar más con las funciones de recompensa, para entender de mejor manera la injerencia que esta tiene en los agentes de aprendizaje reforzado. Es importante destacar que, de los tres esquemas *offline* estudiados, el método CQL presentó reiteradas veces altas oscilaciones en su recompensa obtenida, lo que probablemente se deba a un sobreajuste a los datos de entrenamiento. Cabe señalar que este algoritmo solo entrena una función Q, a diferencia de los otros dos métodos que emplean soft actor critic y utilizan más redes neuronales en su proceso de aprendizaje. Es posible que el uso de redes neuronales más complejas en estos métodos sea una de las razones por las cuales no sufren de este sobreajuste que afecta a CQL.

4.2. Experimento 2: Elevación del péndulo invertido

En esta sección se presentarán los resultados de las pruebas realizadas para el experimento 2, detallado en la Subsubsección 3.5.2.2. En este experimento se entrenó cada agente con la base de datos construida mediante el controlador MPC, que contaba con un tamaño de 10000 registros, se entrenó durante 3000 épocas, y se empleó *batches* de 32 registros. Para evaluar los agentes, se fijó el estado inicial del péndulo en $X = 0$ y $\theta = \pi$ $[rad]$, se asignó un horizonte máximo de 500 *steps* o pasos para cada episodio y se registró la mayor cantidad de *steps* seguidos en los cuales el agente era capaz de elevar el péndulo y mantener su inclinación dentro del rango $-0.3 < \theta < 0.3$ $[rad]$. Los resultados se expondrán mediante la presentación de las funciones de pérdida obtenidas durante el entrenamiento, y las mediciones de desempeño obtenidas al evaluar cada uno de los algoritmos implementados, junto con sus diferentes variaciones arquitectónicas, detalladas en la Sección 3.4.

4.2.1. Deep Q-Learning

A continuación se presentan en las figuras 4.89, 4.90, 4.91 y 4.92 los resultados obtenidos al entrenar y evaluar el algoritmo Deep Q-Learning, junto con sus variantes Double DQL, Dueling DQL y Double Dueling DQL respectivamente, para el experimento de elevar el péndulo invertido.

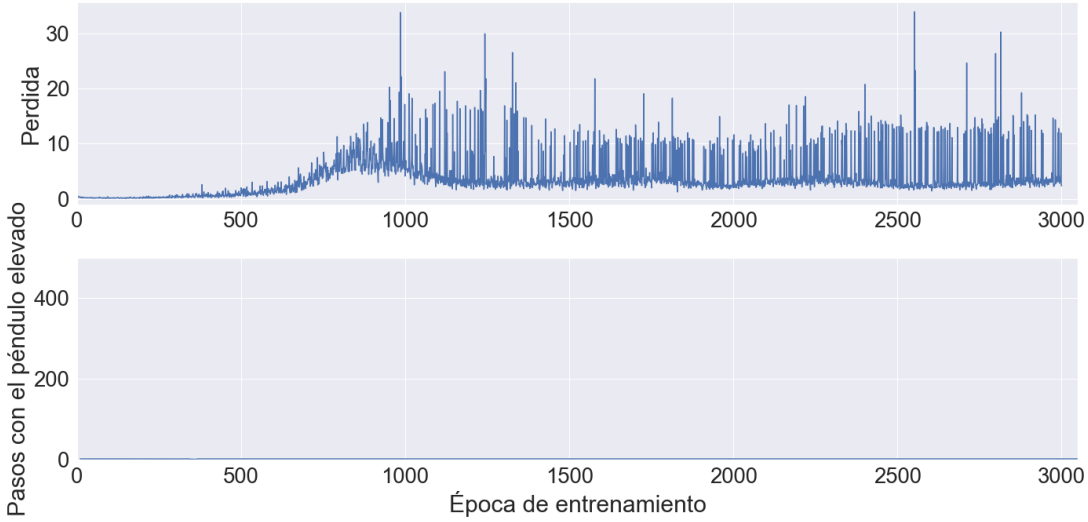

Figura 4.89: Pérdida y recompensas obtenidas para el modelo DQL entrenado para la elevación del péndulo

Figura 4.90: Pérdida y recompensas obtenidas para el modelo Double DQL entrenado para la elevación del péndulo

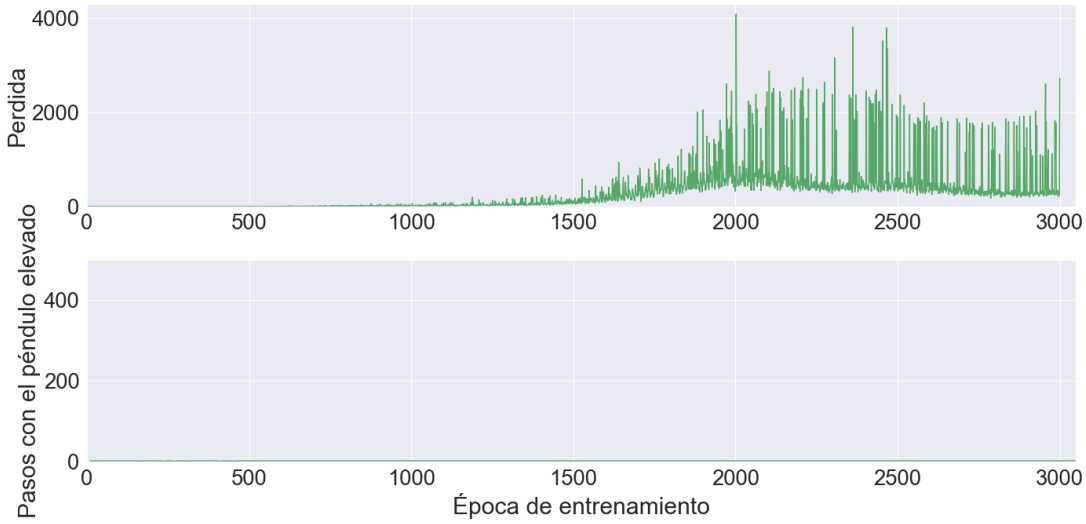

Figura 4.91: Pérdida y recompensas obtenidas para el modelo Dueling DQL entrenado para la elevación del péndulo

Figura 4.92: Pérdida y recompensas obtenidas para el modelo Double Dueling DQL entrenado para la elevación del péndulo

4.2.2. Deep Q-Learning from Demostration

A continuación se presentan en las figuras 4.93, 4.94, 4.95 y 4.96 los resultados obtenidos al entrenar y evaluar el algoritmo Deep Q-Learning from Demostration, junto con sus variantes Double DQfD, Dueling DQfD y Double Dueling DQfD respectivamente, para el experimento de elevar el péndulo invertido.

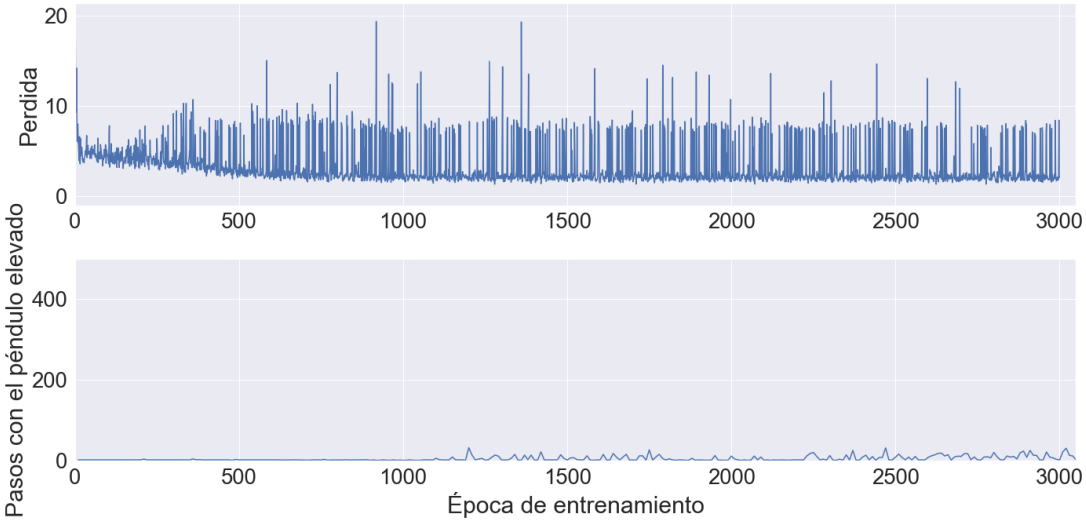

Figura 4.93: Pérdida y recompensas obtenidas para el modelo DQfD entrenado para la elevación del péndulo

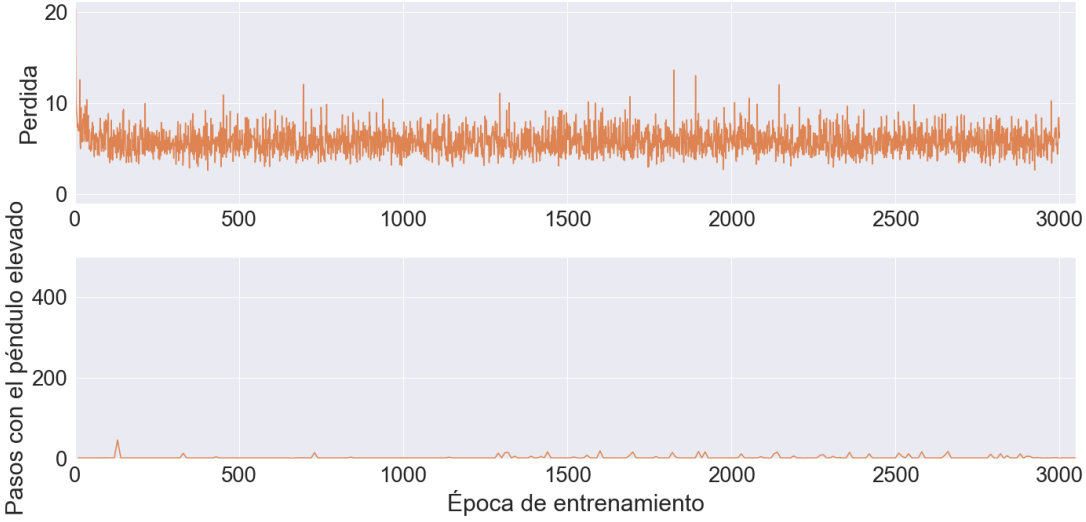

Figura 4.94: Pérdida y recompensas obtenidas para el modelo Double DQfD entrenado para la elevación del péndulo

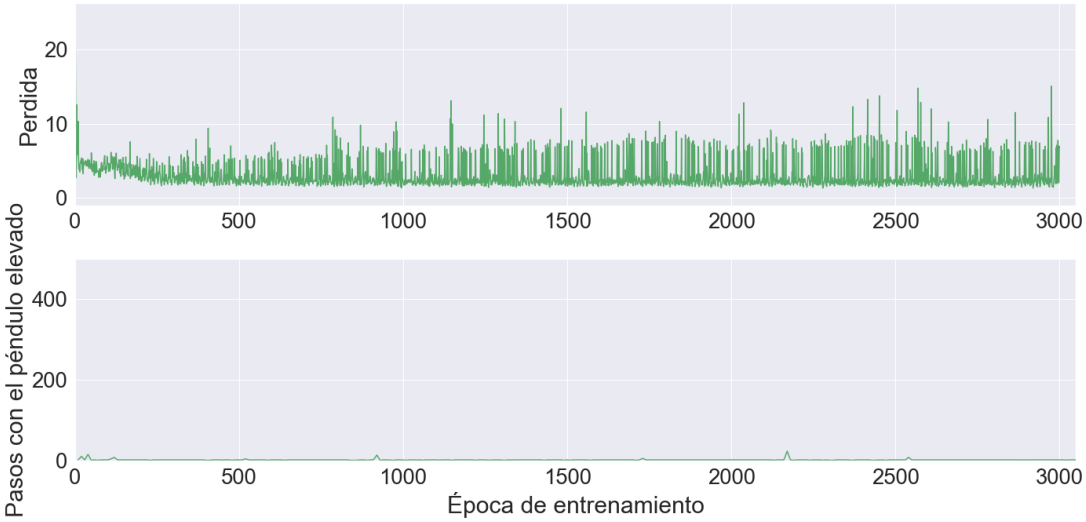

Figura 4.95: Pérdida y recompensas obtenidas para el modelo Dueling DQfD entrenado para la elevación del péndulo

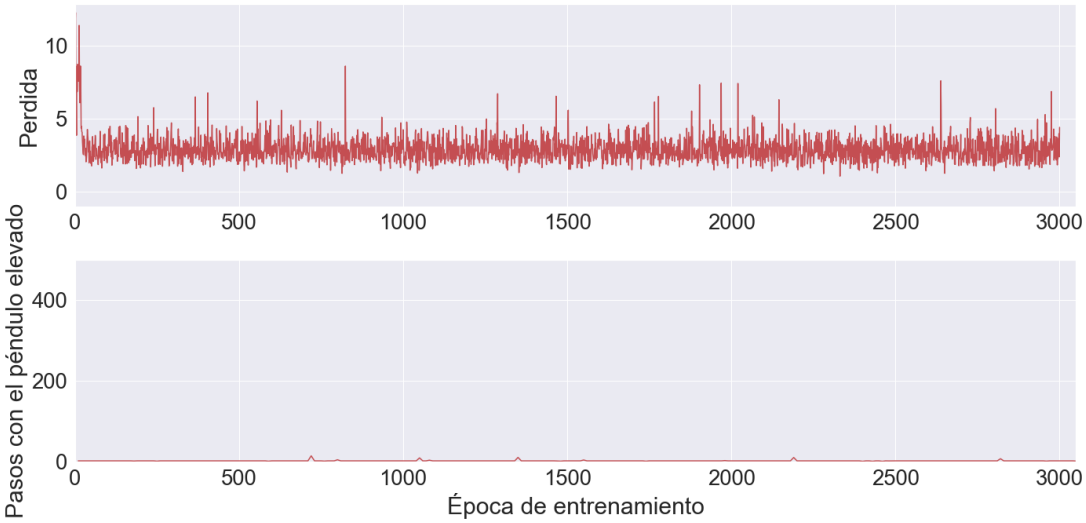

Figura 4.96: Pérdida y recompensas obtenidas para el modelo Double Dueling DQfD entrenado para la elevación del péndulo

4.2.3. Neural Fitted Q Iteration

A continuación se presentan en las figuras 4.97 y 4.98 los resultados obtenidos al entrenar y evaluar el algoritmo Neural Fitted Q Iteration, junto con su variante Double NFQ, para el experimento de elevar el péndulo invertido.

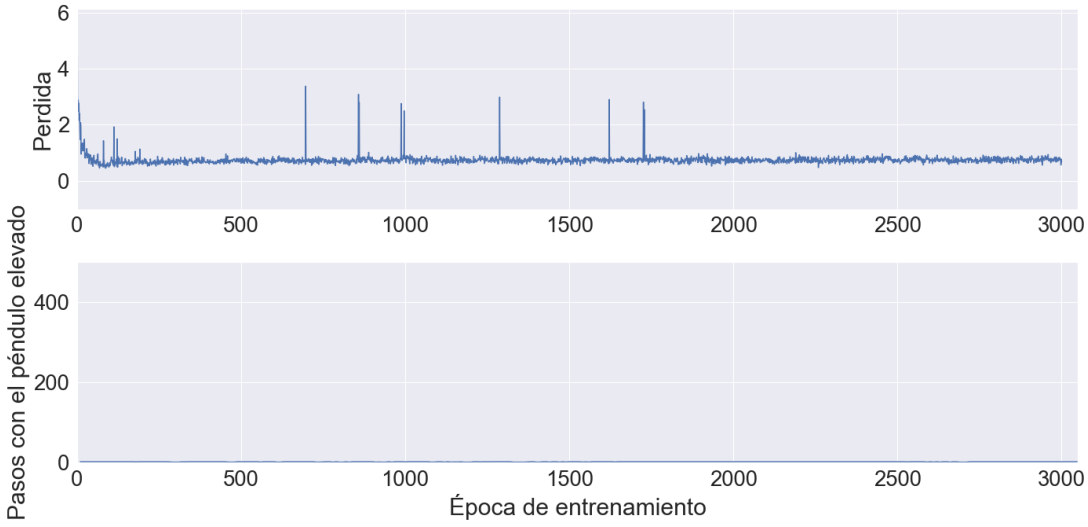

Figura 4.97: Pérdida y recompensas obtenidas para el modelo NFQ entrenado para la elevación del péndulo

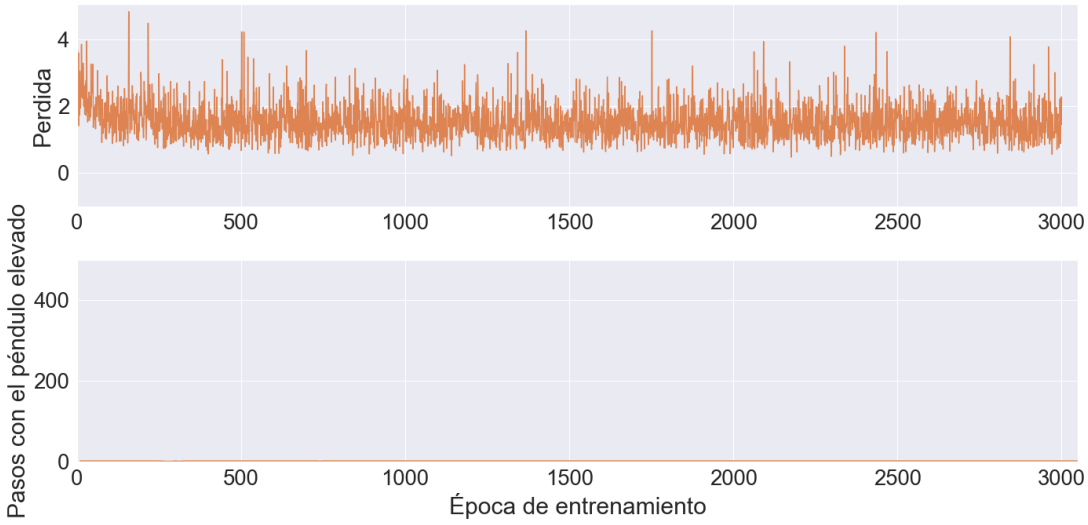

Figura 4.98: Pérdida y recompensas obtenidas para el modelo Double NFQ entrenado para la elevación del péndulo

4.2.4. Conservative Q-Learning

A continuación se presentan en las figuras 4.99, 4.100, 4.101 y 4.102 los resultados obtenidos al entrenar y evaluar el algoritmo Conservative Q-Learning, junto con sus variantes Double CQL, Dueling CQL y Double Dueling CQL respectivamente, para el experimento de elevar el péndulo invertido.

Figura 4.99: Pérdida y recompensas obtenidas para el modelo CQL entrenado para la elevación del péndulo

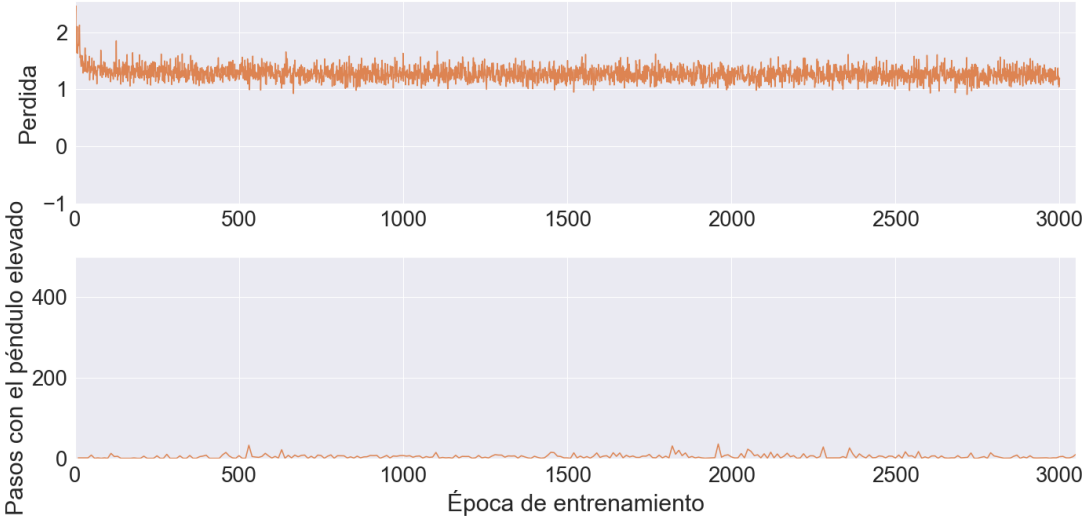

Figura 4.100: Pérdida y recompensas obtenidas para el modelo Double CQL entrenado para la elevación del péndulo

Figura 4.101: Pérdida y recompensas obtenidas para el modelo Dueling CQL entrenado para la elevación del péndulo

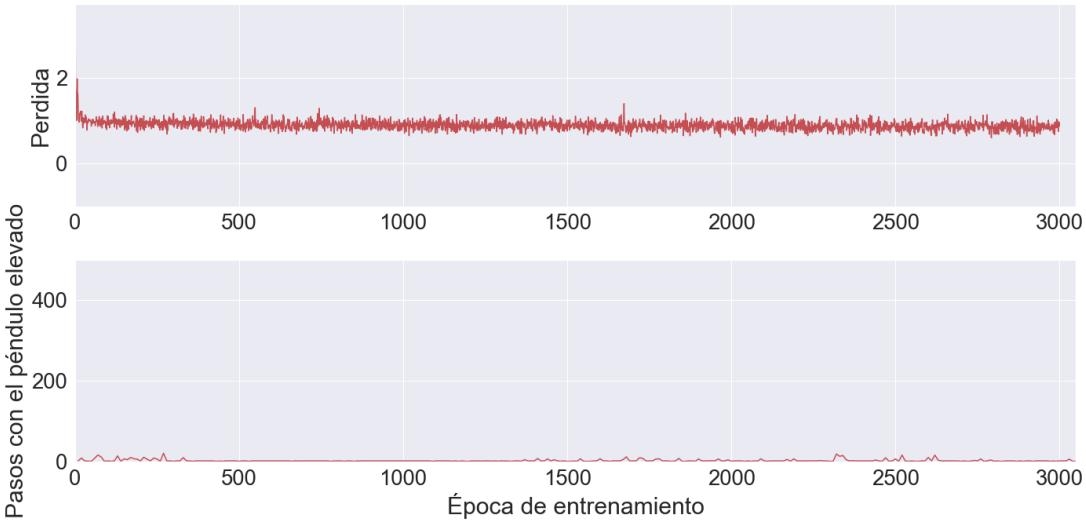

Figura 4.102: Pérdida y recompensas obtenidas para el modelo Double Dueling CQL entrenado para la elevación del péndulo

4.2.5. Implicit Q-Learning

A continuación se presentan en las figuras 4.103, 4.104, 4.105 y 4.106 los resultados obtenidos al entrenar y evaluar el algoritmo Implicit Q-Learning, junto con sus variantes Double IQL, Dueling IQL y Double Dueling IQL respectivamente, para el experimento de elevar el péndulo invertido.

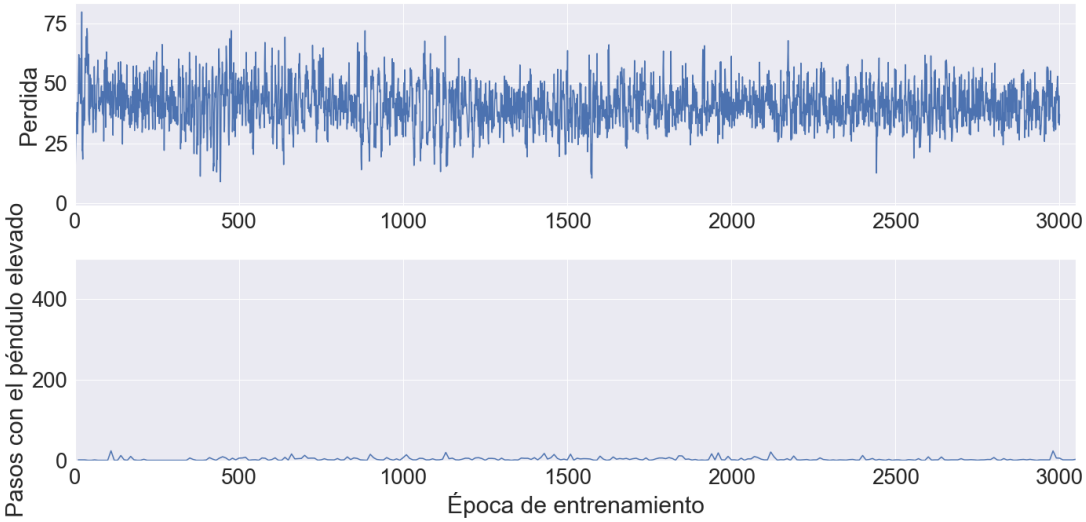

Figura 4.103: Pérdida y recompensas obtenidas para el modelo IQL entrenado para la elevación del péndulo

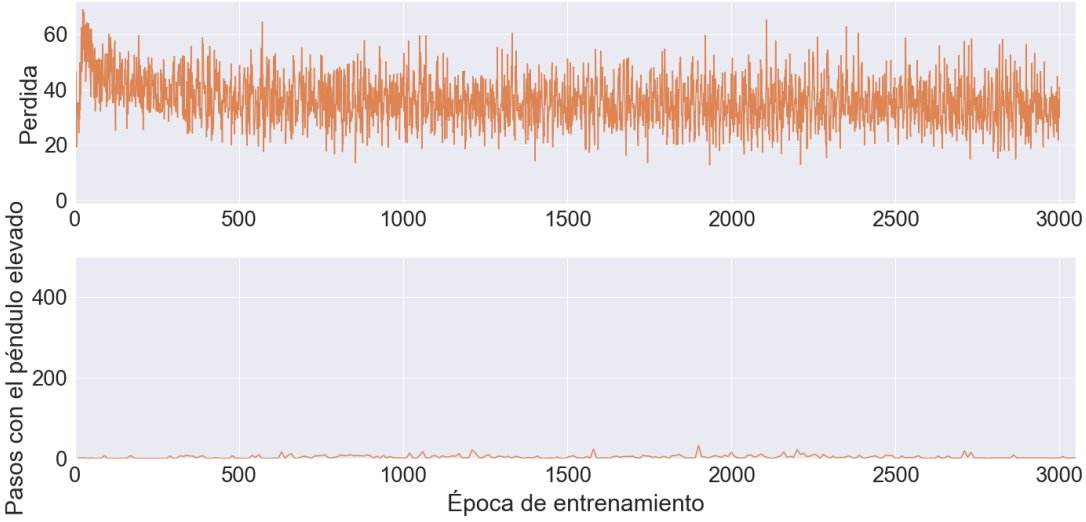

Figura 4.104: Pérdida y recompensas obtenidas para el modelo Double IQL entrenado para la elevación del péndulo

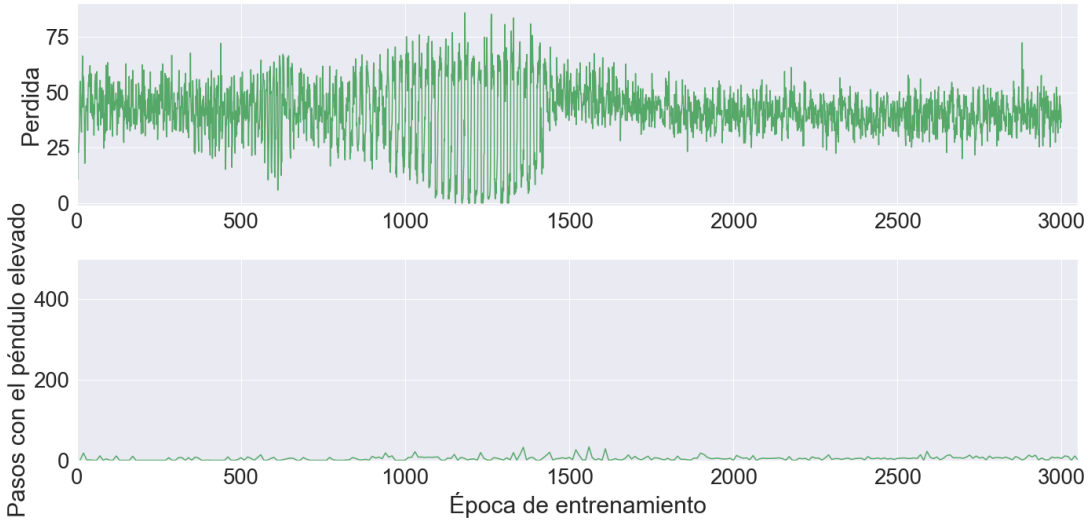

Figura 4.105: Pérdida y recompensas obtenidas para el modelo Dueling IQL entrenado para la elevación del péndulo

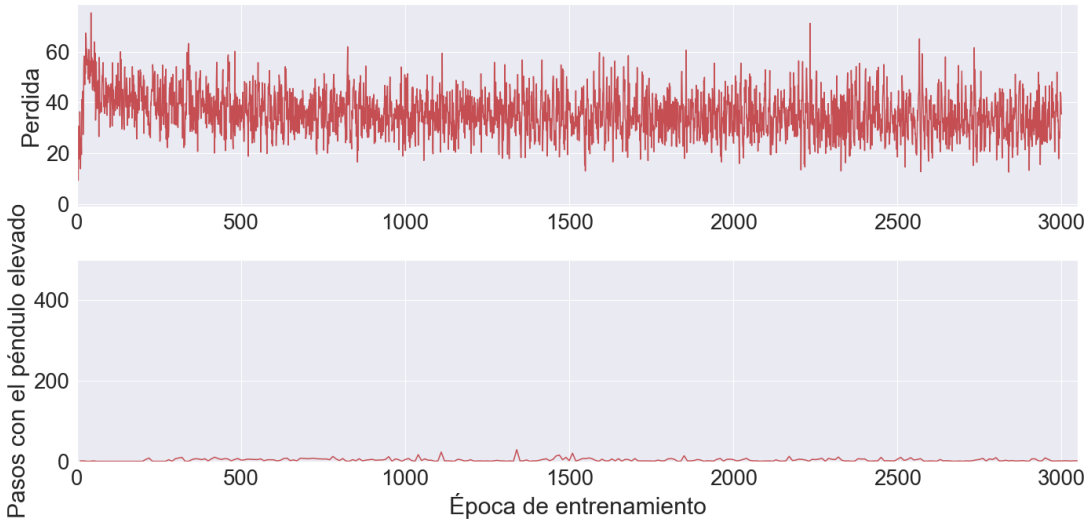

Figura 4.106: Pérdida y recompensas obtenidas para el modelo Double Dueling IQL entrenado para la elevación del péndulo

4.2.6. Batch-Constrained Deep Q-Learning

A continuación se presentan en las figuras 4.107, 4.108, 4.109 y 4.110 los resultados obtenidos al entrenar y evaluar el algoritmo Batch-Constrained Deep Q-Learning, junto con sus variantes Double BCQ, Dueling Double BCQ y Double Dueling BCQ respectivamente, para el experimento de elevar el péndulo invertido.

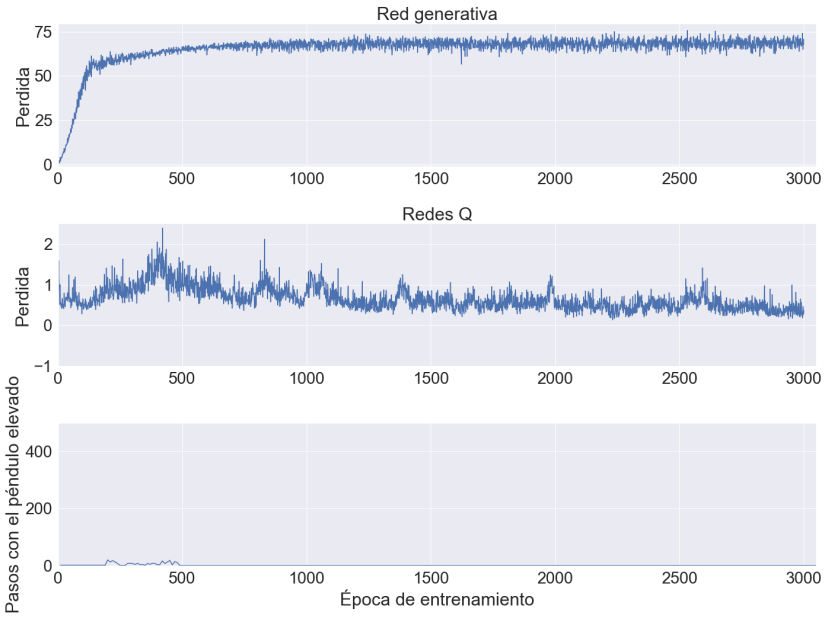

Figura 4.107: Pérdidas y recompensas obtenidas para el modelo BCQ entrenado para la elevación del péndulo

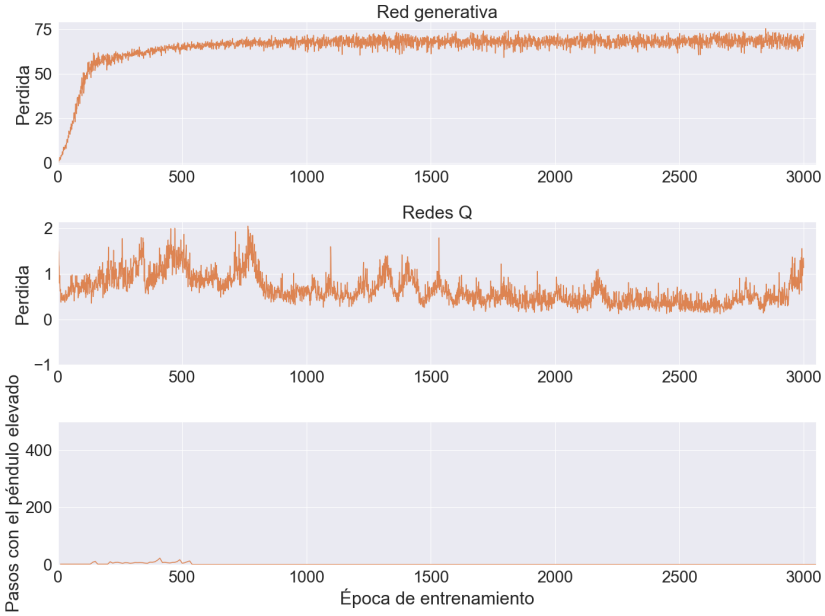

Figura 4.108: Pérdidas y recompensas obtenidas para el modelo Double BCQ entrenado para la elevación del péndulo

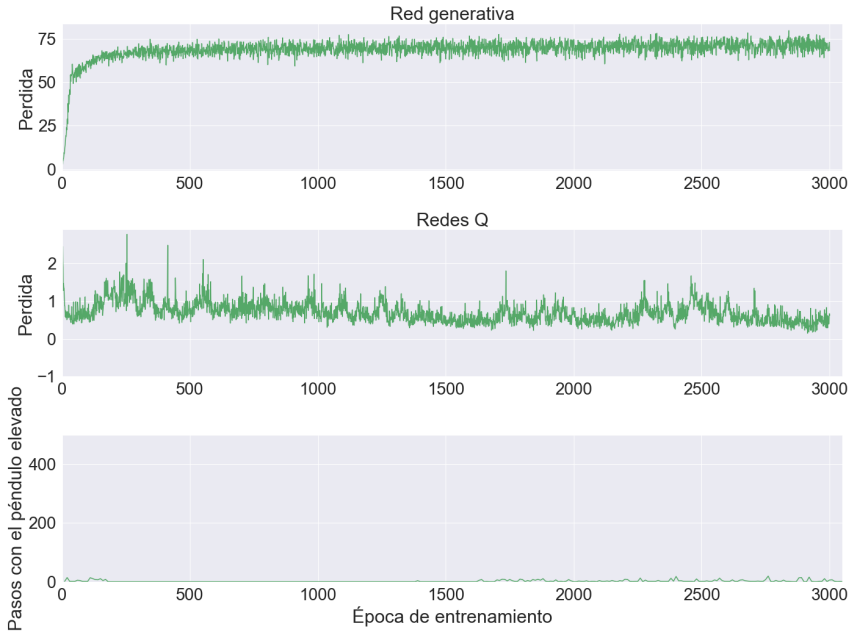

Figura 4.109: Pérdidas y recompensas obtenidas para el modelo Dueling BCQ entrenado para la elevación del péndulo

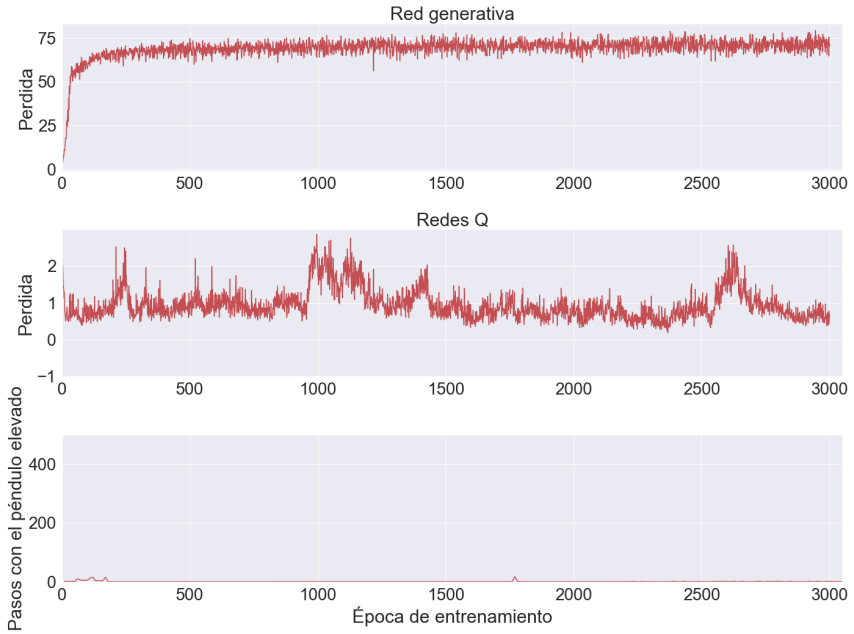

Figura 4.110: Pérdidas y recompensas obtenidas para el modelo Double Dueling BCQ entrenado para la elevación del péndulo

4.2.7. Análisis y comparativa de algoritmos

A partir de los resultados presentados del experimento 2, se aprecia que ninguno de los algoritmos es capaz de mantener elevado el péndulo por un periodo notable de tiempo, considerando que la ventana de evaluación empleada fue de un máximo de 500 pasos. Para poder examinar con mayor detalle los resultados obtenidos, se registran en la Tabla 4.7 para cada algoritmo y sus variantes, la época de entrenamiento y evaluación, tras el que se alcanzó a mantener elevado por más tiempo, en promedio, el péndulo, junto con este tiempo logrado.

Tabla 4.7: Resumen de resultados del experimento de elevar el péndulo invertido

Algoritmo	**Época de mayor tiempo con el péndulo elevando**	**Mayor tiempo con el péndulo elevado**
Deep Q-Learning	10	1
Double Deep Q-Learning	10	1
Dueling Deep Q-Learning	10	1
Double Dueling Deep Q-Learning	1330	25
Deep Q-Learning from Demonstration	1200	31
Double Deep Q-Learning from Demonstration	130	45
Dueling Deep Q-Learning from Demonstration	2170	23
Double Dueling Deep Q-Learning from Demonstration	720	14
Neural Fitted Q Iteration	10	1
Double Neural Fitted Q Iteration	10	1
Conservative Q-Learning	1970	39
Double Conservative Q-Learning	1960	35
Dueling Conservative Q-Learning	2020	16
Double Dueling Conservative Q-Learning	270	20
Implicit Q-Learning	110	24
Double Implicit Q-Learning	1900	32
Dueling Implicit Q-Learning	1560	33
Double Dueling Implicit Q-Learning	1340	29
Batch-Constrained Q-Learning	200	20
Double Batch-Constrained Q-Learning	410	22
Dueling Batch-Constrained Q-Learning	2760	19
Double Dueling Batch-Constrained Q-Learning	1770	17

Los resultados indican que ninguno de los algoritmos logró emplearse satisfactoriamente para aprender a elevar el péndulo invertido y mantenerlo alzado. Pese a esto, se distingue que los algoritmos de aprendizaje reforzado fuera de linea CQL, IQL y BCQ, junto con DQfD que también fue desarrollado para poder ser empleado parcialmente de manera *offline*, lograron elevar momentáneamente el péndulo, a diferencia de los algoritmos DQL y NFQ, que fracasaron rotundamente.

Se conjetura que la tarea de elevar el péndulo resulta ser, en gran medida, más compleja que solo mantenerlo balanceado, y que la función de recompensa customizada propuesta, no es suficiente para reflejar en los datos generados, las políticas correctas de comportamiento. Se sospecha, que una función de recompensa adecuada, para generar datos útiles para el aprendizaje de los agentes, deba tener también en consideración las variables de velocidad del sistema.

Capítulo 5

Conclusiones

Este informe presenta la implementación y evaluación *offline* de seis algoritmos de aprendizaje reforzado. Aunque algunos de ellos fueron diseñados para operar en línea, su adaptación a entornos *offline* ha dado lugar a la creación de nuevos algoritmos. Por esta razón, se evaluaron todos los algoritmos en un enfoque *offline* para poder contrastar sus capacidades de aprendizaje. Los algoritmos trabajados se caracterizaron por ser libres de modelo, basarse en *Q-Learning*, y implementar redes profundas, ya que se deseaba evaluar si los métodos de aprendizaje reforzado, son capaces de controlar un sistema, sin la necesidad de modelarlo matemáticamente. Estos algoritmos se codificaron en *Python*, a fin de que pudieran operar sobre un entorno virtual de un péndulo invertido, desarrollado en la API *Gym* de este lenguaje.

Para el entrenamiento *offline* de estos algoritmos, se generaron set de datos de interacciones en el entorno del péndulo invertido de *Gym*, llamado *CartPole*. Estudiando los efectos de la calidad de datos, estos se generaron empleando soluciones de control ya existentes, para el problema del péndulo invertido, y también empleando interacciones aleatorias que agregaran ruido a los conjuntos de datos. Además, estos set de datos empleados para el entrenamiento, también fueron generados utilizando dos funciones de recompensa distinta, con la intención de estudiar los efectos de entrenar agentes de aprendizaje reforzado, con distintas funciones. Estos algoritmos entrenados se testearon en el ambiente *CartPole*, evaluando sus capacidades de aprender a balancear y alzar el péndulo.

Los resultados obtenidos a partir de estas evaluaciones corresponden a los presentados en este informe, los que permitieron constatar las distintas capacidades de los algoritmos de aprendizaje reforzado implementados. Estos resultados mostraron que de los 6 algoritmos trabajados, 3 de ellos correspondientes a *Deep Q-Learning*, *Deep Q-Learning from Demostration* y *Neural Fitted Q Iteration*, no cuentan con buenas capacidades para emplearse de manera *offline*. Esto se debió a su bajo desempeño en las distintas evaluaciones, en contraste de los otros 3 algoritmos trabajados, correspondientes a *Conservative Q-Learning*, *Implicit Q-Learning* y *Batch Constrained Q-Learning*, que, sin mayores dificultades, lograron aprender a balancear un péndulo de manera óptima. Estos últimos algoritmos mencionados, corresponden a esquemas desarrollados para ser implementados de manera *offline*, lo que da cuenta de las capacidades que poseen estos tipos de esquemas, para dirigir esta área del *Machine Learning* a una orientada por datos.

Otras conclusiones que pudieron obtenerse a partir de los resultados, fue que en el plan-

teamiento *offline*, la calidad de los datos con los que se entrenan los agentes, resulta muy importante. Esto puede entenderse porque, los esquemas de aprendizaje reforzado *offline* buscan extraer comportamientos que sean útiles, que puedan generalizarse, y permitan generar mejores políticas de decisión que las presentes en los datos. También otra conclusión obtenida, es que cada uno de estos métodos busca implementar nuevos conceptos y principios, sobre una misma base, lo que da espacio a experimentar y probar nuevas técnicas de aprendizaje reforzado, que logren aprovechar los distintos desarrollos que se han hecho en los últimos años.

Finalizando los últimos comentarios de esta investigación, el trabajo realizado no estuvo exento de dificultades ni contrariedades, puesto que uno de los experimentos desarrollados no logró conseguir el desempeño esperado. No obstante, este final inesperado da paso a proponer trabajos futuros, que involucren el estudio de las funciones de recompensa con las que trabajan estos algoritmos, y el cómo pueden ayudar a entrenar agentes a solucionar problemas más complejos.

Bibliografía

[1] Sutton, R. S. y Barto, A. G., Reinforcement Learning: An Introduction. The MIT Press, second ed., 2018, http://incompleteideas.net/book/the-book-2nd.html.

[2] Silver, D., Huang, A., Maddison, C., Guez, A., Sifre, L., Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., y Hassabis, D., "Mastering the game of go with deep neural networks and tree search," Nature, vol. 529, pp. 484–489, 2016, doi:10.1038/nature16961.

[3] Tesauro, G., "Td-gammon, a self-teaching backgammon program, achieves master-level play," Neural Computation, vol. 6, no. 2, pp. 215–219, 1994, doi:10.1162/neco.1994.6.2.215.

[4] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., y Riedmiller, M. A., "Playing atari with deep reinforcement learning," CoRR, vol. abs/1312.5602, 2013, http://arxiv.org/abs/1312.5602.

[5] Heess, N., TB, D., Sriram, S., Lemmon, J., Merel, J., Wayne, G., Tassa, Y., Erez, T., Wang, Z., Eslami, S. M. A., Riedmiller, M. A., y Silver, D., "Emergence of locomotion behaviours in rich environments," CoRR, vol. abs/1707.02286, 2017, http://arxiv.org/abs/1707.02286.

[6] OpenAI, Andrychowicz, M., Baker, B., Chociej, M., Józefowicz, R., McGrew, B., Pachocki, J. W., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., Schneider, J., Sidor, S., Tobin, J., Welinder, P., Weng, L., y Zaremba, W., "Learning dexterous in-hand manipulation," CoRR, vol. abs/1808.00177, 2018, http://arxiv.org/abs/1808.00177.

[7] Levine, S., Kumar, A., Tucker, G., y Fu, J., "Offline reinforcement learning: Tutorial, review, and perspectives on open problems," CoRR, vol. abs/2005.01643, 2020, https://arxiv.org/abs/2005.01643.

[8] Brunton, S. L. y Kutz, J. N., Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control. USA: Cambridge University Press, 1st ed., 2019.

[9] Tehrani, K. A. y Mpanda, A., "Pid control theory," en Introduction to PID Controllers (Panda, R. C., ed.), cap. 9, Rijeka: IntechOpen, 2012, doi:10.5772/34364.

[10] Wikipedia contributors, "Pid controller — Wikipedia, the free encyclopedia," 2022, https://en.wikipedia.org/wiki/PID_controller. [Online; accessed 16-December-2022].

[11] Setyawan, G. E., Kurniawan, W., y Gaol, A. C. L., "Linear quadratic regulator controller (lqr) for ar. drone's safe landing," en 2019 International Conference on Sustainable Information Engineering and Technology (SIET), pp. 228–233, 2019, doi:10.1109/SIET48054.2019.8986078.

[12] Rana, M. S., Pota, H. R., y Petersen, I. R., "Model predictive control of atomic force microscope for fast image scanning," en 2012 IEEE 51st IEEE Conference on Decision and Control (CDC), pp. 2477–2482, 2012, doi:10.1109/CDC.2012.6426103.

[13] Kumar, A. y Levine, S., "Offline reinforcement learning from algorithms to practical challenges." https://drive.google.com/file/d/1_aJxnlwLsJYup-___qKi-ZnujQho6ibDk/view, 2020. Accessed: 2021–12-13.

[14] Watkins, C. J. C. H. y Dayan, P., "Q-learning," Mach. Learn., vol. 8, pp. 279–292, 1992.

[15] Konda, V. y Tsitsiklis, J., "Actor-critic algorithms," en Advances in Neural Information Processing Systems (Solla, S., Leen, T., y Müller, K., eds.), vol. 12, MIT Press, 1999, https://proceedings.neurips.cc/paper/1999/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf.

[16] van Hasselt, H., Guez, A., y Silver, D., "Deep reinforcement learning with double q-learning," 2015, doi:10.48550/ARXIV.1509.06461.

[17] Hasselt, H. v., "Double q-learning," en Proceedings of the 23rd International Conference on Neural Information Processing Systems - Volume 2, NIPS'10, (Red Hook, NY, USA), p. 2613–2621, Curran Associates Inc., 2010.

[18] Wang, Z., Schaul, T., Hessel, M., van Hasselt, H., Lanctot, M., y de Freitas, N., "Dueling network architectures for deep reinforcement learning," 2015, doi:10.48550/ARXIV.1511.06581.

[19] Hester, T., Vecerík, M., Pietquin, O., Lanctot, M., Schaul, T., Piot, B., Sendonaris, A., Dulac-Arnold, G., Osband, I., Agapiou, J. P., Leibo, J. Z., y Gruslys, A., "Learning from demonstrations for real world reinforcement learning," CoRR, vol. abs/1704.03732, 2017, http://arxiv.org/abs/1704.03732.

[20] Ernst, D., Geurts, P., y Wehenkel, L., "Tree-based batch mode reinforcement learning," J. Mach. Learn. Res., vol. 6, p. 503–556, 2005.

[21] Riedmiller, M., "Neural fitted q iteration – first experiences with a data efficient neural reinforcement learning method," en Machine Learning: ECML 2005 (Gama, J., Camacho, R., Brazdil, P. B., Jorge, A. M., y Torgo, L., eds.), (Berlin, Heidelberg), pp. 317–328, Springer Berlin Heidelberg, 2005.

[22] Kumar, A., Zhou, A., Tucker, G., y Levine, S., "Conservative q-learning for offline reinforcement learning," 2020, doi:10.48550/ARXIV.2006.04779.

[23] Kostrikov, I., Nair, A., y Levine, S., "Offline reinforcement learning with implicit q-learning," 2021, doi:10.48550/ARXIV.2110.06169.

[24] Fujimoto, S., Meger, D., y Precup, D., "Off-policy deep reinforcement learning without exploration," 2018, doi:10.48550/ARXIV.1812.02900.

[25] Barto, A. G., Sutton, R. S., y Anderson, C. W., "Neuronlike adaptive elements that can solve difficult learning control problems," IEEE Transactions on Systems, Man, and Cybernetics, vol. SMC-13, no. 5, pp. 834–846, 1983, doi:10.1109/TSMC.1983.6313077.

[26] Wikipedia contributors, "Inverted pendulum — Wikipedia, the free encyclopedia," 2022, https://en.wikipedia.org/wiki/Inverted_pendulum. [Online; accessed 16-December-2022].

[27] Schweighofer, K., Hofmarcher, M., Dinu, M., Renz, P., Bitto-Nemling, A., Patil, V. P.,

y Hochreiter, S., "Understanding the effects of dataset characteristics on offline reinforcement learning," CoRR, vol. abs/2111.04714, 2021, https://arxiv.org/abs/2111.04714.