ESCUELA TÉCNICA SUPERIOR DE INGENIEROS
INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA

Proyecto Fin de Carrera

DESARROLLO DE UNA PLATAFORMA DE INVESTIGACIÓN DE "DEEP REINFORCEMENT LEARNING"

(Development of a Research Framefork for Deep Reinforcement Learning)

Para acceder al Titulo de

INGENIERO DE TELECOMUNICACIÓN

Autor: Fernando Arbeiza Guerra

09 - 2017

UNIVERSIDAD DE CANTABRIA

E.T.S. DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACION

INGENIERÍA DE TELECOMUNICACIÓN

CALIFICACIÓN DEL PROYECTO FIN DE CARRERA

Realizado por: ***Fernando Arbeiza Guerra***
Director del PFC: **Tomás Fernández Ibáñez**
Título: "Desarrollo de una plataforma de investigación de "Deep Reinforcement Learning""
Title: "Development of a Research Framefork for Deep Reinforcement Learning"

Presentado a examen el día:

para acceder al Título de

INGENIERO DE TELECOMUNICACIÓN

Composición del Tribunal:
Presidente (Apellidos, Nombre):
Secretario (Apellidos, Nombre):
Vocal (Apellidos, Nombre):

Este Tribunal ha resuelto otorgar la calificación de: .....

Fdo.: El Presidente

Fdo.: El Secretario

Fdo.: El Vocal

Fdo.: El Director del PFC
(sólo si es distinto del Secretario)

E.T.S. DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACION

Vº Bº del Subdirector

Proyecto Fin de Carrera Nº
(a asignar por Secretaría)

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

Esta introducción presenta el contexto en el que se enmarca el presente proyecto, tanto en el campo de conocimiento en el que se ha centrado, como en el entorno empresarial que lo ha motivado. También destaca los objetivos que se pretenden con el resultado del proyecto y, finalmente, explica la estructura del documento completo para su posterior consulta.

Aprendizaje por refuerzo profundo

En esta sección se presenta brevemente el Aprendizaje por Refuerzo Profundo, campo de conocimiento al que el presente proyecto se aplica; comenzando desde el área más general hasta lo más concreto. Finalmente, se destaca la motivación del proyecto respecto a este campo.

Como introducción, diremos que la **Inteligencia Artificial** («Artificial Intelligence» o AI) es el campo de conocimiento más general donde se engloba el proyecto. Genéricamente, la Inteligencia Artificial estudia los agentes inteligentes, es decir, los dispositivos capaces de percibir su entorno y tomar decisiones sobre la acción a tomar para maximizar las posibilidades de cumplir su objetivo. El término inteligencia fue tomado a propósito, intentando buscar una similitud entre la parametrización y el proceso de decisión de estos dispositivos, y las habilidades netamente humanas de aprender, abstraer, razonar y decidir para la resolución de problemas; sobre todo aquellos problemas donde la dificultad es tan grande que la intuición humana se demuestra como determinante a la hora de optimizar la solución a tal problema.

Dentro de la Inteligencia Artificial, el **Aprendizaje Automático** («Machine Learning» o ML) es la rama que estudia las técnicas que permiten a los dispositivos, sin estar explícitamente programados para afrontar un problema, ser entrenados para resolver el problema de forma óptima. Concre-

tamente, trata de crear sistemas que puedan generalizar el comportamiento para optimizar la respuesta a un problema, basándose en información suministrada como ejemplos de soluciones a ese problema. Al igual que en la Inteligencia Artificial, el término Aprendizaje busca la analogía entre el entrenamiento de estos dispositivos y la capacidad del ser humano de aprender con ejemplos, y abordar situaciones nuevas con éxito basadas en situaciones diferentes vividas anteriormente.

Más específicamente, el **Aprendizaje por Refuerzo** («Reinforcement Learning» o RL), plantea el Aprendizaje Automático de forma que el dispositivo o agente es entrenado únicamente con el resultado de las acciones que decide, de alguna manera, ejecutar en un entorno; y que recibe en forma de observación del estado de tal entorno y una recompensa como medición del éxito en la tarea que debe realizar [23]. Habitualmente, la acción tomada no afecta solamente a la recompensa más inmediata recibida, sino a la que se puede obtener después de sucesivas acciones, lo que hace que descubrir la acción que maximiza la recompensa sea más exigente.

Por lo tanto, en el Aprendizaje por Refuerzo, al agente nunca se le presentan las acciones correctas, ni se le corrige explícitamente; sino que, en este proceso de entrenamiento, es el propio agente el que decide la acción a tomar para observar el resultado (resultado expresado, tanto en el estado del entorno resultado de la acción como en la recompensa obtenida). Para tomar estas decisiones durante el entrenamiento, el agente debe seguir una política que debe equilibrar dos conceptos:

Exploración: para tomar acciones poco observadas, pero que que puedan llevar a nuevas soluciones aún mejores que las ya conocidas

Explotación: que se basa en el conocimiento actual de la acción más correcta para intentar optimizar la respuesta

Por todo ello, el Aprendizaje por Refuerzo es aplicable cuando el agente puede ser entrenado con entornos interactivos en los que dicho agente puede ejecutar acciones y comprobar su resultado, y así aprender de su propia experiencia; y en los que no se cuenta, o que no es práctico obtener ejemplos que son tanto correctos como representativos de todas y cada una de las situaciones que el agente puede encontrar, al contrario que en el caso del Aprendizaje Supervisado, otro campo del Aprendizaje Automático en el que sí se cuenta con tales ejemplos para entrenar al sistema.

Finalmente, el **Aprendizaje por Refuerzo Profundo** («Deep Reinforcement Learning» o DRL) es, simplemente, la aplicación de, típicamente,

múltiples capas de unidades de procesado no lineal (tales como Redes Neuronales), a los procesos y algoritmos de Aprendizaje por Refuerzo; para el proceso de entrenamiento y decisión.

Dado el avance que han experimentado estás técnicas con la investigación que se está llevando a cabo en las últimas fechas en estos campos, están siendo aplicadas con éxito en toda clase de problemas que eran difícilmente solubles anteriormente. Como muestra, podemos nombrar varios ejemplos donde se aplica el Aprendizaje por Refuerzo, normalmente combinado con otras técnicas:

Robótica: Aunque en tareas repetitivas ya está completamente generalizado el uso de robots, éstos no son aplicables cuando la tarea no está perfectamente definida de antemano. Tareas tan triviales para el ser humano como agarrar objetos de diferentes formas y tamaños, son imposibles para un robot autónomo. El Aprendizaje por Refuerzo está siendo aplicado, desde para detectar si el robot realmente ha agarrado algo [14], hasta para conseguir que el robot autónomamente agarre el objeto deseado.

Conducción autónoma: Pese a que el Aprendizaje Supervisado mediante la generación de ejemplos de conducción puede ser generado por humanos, el Aprendizaje por Refuerzo también es aplicado en los dispositivos de conducción autónoma [7], principalmente mediante simuladores del entorno de conducción. Esto permite que se pueda entrenar a los sistemas con muchas más situaciones de las que se puedan dar en un entorno real o de forma mucho más segura.

Juegos de mesa: Algunos de ellos eran muy difíciles de solucionar mediante técnicas de programación tradicionales porque poseían un gran número de combinaciones o el éxito podía estar muy lejano en tiempo (como el ajedrez o, últimamente, por su complicación, el Go). Sin embargo, están siendo solucionados aplicando el Aprendizaje por Refuerzo [21].

Pese a que el campo del Aprendizaje Automático no es nuevo, no ha sido hasta últimamente cuando se ha producido un enorme avance, consiguiéndose aplicar estas técnicas a problemas cada vez más complejos. Sin embargo, debido a esta novedad, no han surgido todavía plataformas suficientemente estables y maduras y que cuenten con estos últimos avances en los algoritmos y con las herramientas de análisis necesarias para ser utilizadas en la investigación y experimentación; lo que ha motivado este proyecto.

Monkey from the Future

Esta sección presenta el contexto empresarial en el que se enmarca el proyecto, concluyendo con la motivación que propició su ejecución.

Monkey from the Future es una iniciativa empresarial creada en 2017 por un investigador en Inteligencia Artificial y el autor de este proyecto, con el objetivo de investigar en el campo del Aprendizaje Automático y aplicar todos los avances comentados en la sección anterior a problemas reales.

Por lo tanto, el presente proyecto se enmarca en un contexto de una iniciativa empresarial emergente de reciente creación, y que cuenta con un equipo pequeño formado por los dos socios y otros dos investigadores colaboradores con la iniciativa. Ello permite que el grupo sea muy dinámico y con gran facilidad para pivotar, buscando tales aplicaciones del Aprendizaje Automático.

Y, precisamente, para la investigación en el campo del Aprendizaje por Refuerzo, de forma que los experimentos pudiesen ser llevados a cabo por el equipo de la forma más eficiente posible, surge la necesidad de realización del presente proyecto.

Objetivos

La presente sección detalla los objetivos a alcanzar por el resultado del proyecto.

Como ya se detalló en las anteriores secciones, existía una necesidad en el equipo de una plataforma para facilitar la investigación, desarrollo y prueba de los distintos algoritmos de Aprendizaje por Refuerzo. La plataforma, resultado del presente proyecto, debía contar con las características reflejadas en las siguientes secciones.

Ejecución de experimentos

Uno de los pilares en la investigación de algoritmos de Aprendizaje por Refuerzo es la ejecución de experimentos para probar los distintos algoritmos en diferentes entornos y variando los hiperparámetros para comprobar sus efectos en el entrenamiento.

Los investigadores, usuarios finales de la plataforma, deben invertir el tiempo mínimo para la preparación y ejecución de los experimentos, para no interferir en su labor de investigación. Por lo tanto, la ejecución de nuevos experimentos debe ser sencilla; asimismo, se debe posibilitar esta ejecución

en distintas plataformas y arquitecturas; tanto locales como remotas o en la nube.

Múltiples entornos de problema

El principal objetivo de la plataforma es investigar la aplicación de los algoritmos de Aprendizaje por Refuerzo en nuevos problemas; por lo tanto, debe ser posible cambiar los entornos de problema en los que se ejecutan los experimentos, y que pueden ser de la más variada naturaleza: motores físicos, simuladores, juegos, motores 3D...

Múltiples algoritmos

El sistema debe contar con múltiples algoritmos de Aprendizaje por Refuerzo, perteneciendo a diferentes familias de algoritmos, para que puedan ser experimentados en los distintos entornos y analizar cual de ellos es más aplicable al entorno y problema en cuestión. Adicionalmente, la plataforma debe ser diseñada para añadir, en el futuro, nuevos algoritmos para su análisis.

Modificación y almacenado de los hiperparámetros

Los hiperparámetros son aquellos parámetros de los algoritmos de Aprendizaje por Refuerzo que modifican el proceso de aprendizaje y explotación de tal algoritmo. Como tales, uno de los objetivos de la experimentación sobre algoritmos de Aprendizaje por Refuerzo es la búsqueda de los valores de tales hiperparámetros necesarios para que el algoritmo realice el entrenamiento y la explotación de manera efectiva.

Estos hiperparámetros requieren una cuidadosa configuración, puesto que el desempeño de los algoritmos depende, en gran medida, de tales hiperparámetros y de las condiciones iniciales del sistema. Habitualmente, estos valores se buscan mediante búsqueda manual («Manual Search»), en la que es el investigador el que selecciona, personalmente, los valores antes de ejecutar cualquier experimento.

Por todo ello, el sistema tiene que estar preparado para poder modificar tales parámetros de forma amigable y, al mismo tiempo, permitir el registro y almacenado de los hiperparámetros utilizados en cada experimento, para un futuro análisis y para la replicabilidad de los experimentos.

Análisis del desarrollo y resultado del experimento

Las implementaciones disponibles, hasta el momento, de los algoritmos de Aprendizaje Profundo, sólo analizan limitados indicadores del resultado de los experimentos. Habitualmente, esto se limita a la recompensa obtenida por el agente durante el entrenamiento del sistema, pero no otros indicadores que pueden ser muy útiles para el análisis de los experimentos, sobre todo de aquellos que no han tenido éxito.

Por lo tanto, el sistema debe contar con un sistema de análisis que permita observar el mayor número de indicadores y medidas relevantes posible. Además, es deseable que tal análisis pueda efectuarse mientras los experimentos estén desarrollándose pues, de esta forma, y dada la longitud de esta clase de experimentos, puede decidirse si un experimento debe continuar o cancelarlo de manera anticipada para liberar recursos.

Estructura del documento

Después de esta introdución, en los siguientes capítulos de esta memoria de proyecto se presenta: inicialmente, la metodología seguida para el desarrollo del proyecto para posteriormente presentar los algoritmos principales implementados y las herramientas utilizadas en su implementación. Dicha implementación y el despliegue realizado de la plataforma se detallan en los siguientes capítulos.

Finalmente, se concretan las conclusiones a las que se llega en el proyecto y posibles desarrollos futuros.

Además, se incluye un apéndice con ejemplos de ejecución de la plataforma y el análisis realizado posteriormente.

Capítulo 2

Metodología de desarrollo

El presente capítulo presenta la metodología escogida para el desarrollo del proyecto y la motivación para su elección.

Metodología ágil

Como ya se resaltó, el presente proyecto se encuadra en un contexto dinámico de empresa emergente, donde es importante la gestión de requisitos cambiantes, según las necesidades que vayan surgiendo; y la entrega constante de funcionalidad, para su inmediata utilización en la experimentación. Además, el pequeño tamaño de los grupos de trabajo, impone que el esfuerzo sea dirigido mayoritariamente en el diseño y desarrollo de esta funcionalidad; y reduciendo, en todo lo posible, cualquier actividad no imprescindible que pueda derivar esfuerzo.

Por lo tanto, dentro de este contexto, las metodologías ágiles para el desarrollo de software encajan a la perfección; como puede verse por las principales características de estas metodologías [4]:

Entrega de versiones funcionales temprana y continua: Uno de los principios de las metodologías ágiles es que la entrega de versiones funcionales debe ser continua y cada poco tiempo dentro del desarrollo del proyecto. Por lo tanto, aplicando estas metodologías, se cuenta con una versión funcional, usable, o incluso lanzable de forma continua. En el presente proyecto, esto permite que cada una de estas versiones resultantes pueda ser inmediatamente utilizada para avanzar en la experimentación con diversos entornos de problema, y aprovechadas las diferentes características que se hayan añadido.

Adaptación a cambios en los requisitos: Adicionalmente, las metodologías ágiles permiten que cualquier cambio en los requisitos pueda ser rápidamente incorporado y analizado para su implementación y, en el caso del presente proyecto, su posterior uso en la experimentación.

Colaboración y comunicación continua: La comunicación cara a cara es continua, lo que favorece la colaboración entre los miembros del equipo. Asimismo, se reduce el riesgo de ineficiencias, tanto en esfuerzo duplicado, como en requisitos no completamente comprendidos en la implementación.

Sencilla medida del progreso del proyecto: El disponer de versiones completas cada poco tiempo permite que se puedan utilizar dichas versiones para la medida del progreso, simplemente analizando las características ya incluidas en la versión y las restantes planificadas.

Scrum

Por todo lo mencionado en el apartado anterior, la metodología ágil de desarrollo de software finalmente seleccionada para este proyecto fue Scrum [20], pues cumplía todo lo necesario para encajar con el contexto, equipo y objetivos de proyecto.

Sin embargo, en el presente proyecto, la metodología Scrum fue todavía más simplificada, para adaptarse al pequeño tamaño del equipo.

Las siguientes secciones describen tanto el planteamiento original de la metodología Scrum, como las simplificaciones realizadas para adaptarlas al presente proyecto.

Planteamiento original

Esta sección describe el planteamiento original de la metodología de desarrollo de software Scrum, definiendo los principales conceptos que se manejan en ella.

Equipo

La siguiente lista detalla los diferentes roles que tendrán las personas del equipo, y que deben coordinarse entre sí para el desarrollo del proyecto:

Dueño de Producto (Product Owner): Representa la visión de negocio de los principales interesados en el producto: por lo tanto, es el punto

de comunicación entre los usuarios finales y el Equipo de Desarrollo. Su tarea principal es mantener la Lista de Tareas del Producto; tanto añadiendo nuevas tareas que surjan en la comunicación con los usuarios finales, como priorizando las tareas existentes o clarificando el significado de las tareas pendientes al Equipo de Desarrollo.

Maestro de Scrum (Scrum Master): Supervisa que el proceso se realiza según la metodología expuesta. Organiza las distintas actividades del proceso y maneja y trata de eliminar cualquier impedimento al rendimiento del Equipo de Desarrollo.

Equipo de Desarrollo (Development Team): Es el que se encarga, efectivamente, del desarrollo del proyecto en sí, luego es el que produce todos los entregables del proyecto.

Documentos

En la metodología Scrum, los documentos tratan de reducirse al máximo, para que todo el esfuerzo sea dirigido a la producción de valor en el proyecto. Los documentos principales son dos: la Lista de Tareas de Producto y la Lista de Tareas de Sprint.

Lista de Tareas del Producto (Product Backlog): Que lista, en orden de prioridad, todos los requisitos que debe cumplir el producto; usualmente, en forma de historias desde el punto de vista del usuario. El Dueño de Producto es el encargado de actualizar y priorizar esta lista según los requisitos que extrae de su comunicación con el usuario final, y la estimación de esfuerzo que le transmite el equipo de desarrollo.

Lista de Tareas del Sprint (Sprint Backlog): Una vez se ha escogido de la Lista de Tareas del Producto la siguiente tarea a realizar, se subdivide, a su vez, en tareas más pequeñas, de forma que no sobrepasen una o dos jornadas de trabajo cada una, y se añaden a la Lista de Tareas del Sprint. Es de esta lista de donde los integrantes del Equipo de Desarrollo seleccionan la tarea a realizar durante el desarrollo del Sprint.

Flujo de Trabajo

La presente sección describe el flujo de trabajo que se sigue en la metodología Scrum, con las diferentes actividades a realizar para implementarla.

Sprint En la metodología Scrum, el proyecto es desarrollado incrementalmente en Sprints. Cada Sprint es un periodo determinado de tiempo (típicamente, 2 a 4 semanas), durante las que el Equipo de Desarrollo trabaja en la versión actual del proyecto, para sacar una nueva versión realizando las tareas especificadas en la Lista de Tareas del Sprint. Al final del Sprint, se debe generar una nueva versión funcional y usable del proyecto, que puede ser, incluso, entregada al cliente. Un nuevo Sprint comienza inmediatamente ha finalizado el Sprint anterior.

Durante este flujo de trabajo, existen varios eventos que se deben organizar en cada fase del Sprint. Estos eventos están definidos de antemano, tanto en regularidad como en tiempo máximo, y tratan de ser lo suficientemente raros y cortos como para no interferir en el desarrollo de proyecto; pero lo suficientemente frecuentes como para minimizar cualquier otra reunión ajena al proceso. En la metodología Scrum, se definen los siguientes eventos principales:

Planificación de Sprint (Sprint Planning): En donde se planifica el trabajo a realizar durante el Sprint. Todos los miembros del equipo participarán en esta fase. El objetivo de esta reunión es:

- Seleccionar las tareas de la Lista de Tareas de Producto que se realizarán durante el Sprint.
- Dividir las tareas en otras más pequeñas y estimar el esfuerzo necesario
- Crear la Lista de Tareas de Sprint con estas tareas

Reunión Diaria (Daily Scrum): Es una reunión diaria de reducida duración, para planificar la siguiente jornada de desarrollo. En ella se exponen los resultados de la jornada anterior, se comunican las tareas en las que se va a invertir esfuerzos en la jornada siguiente y se comenta cualquier problema que se haya observado durante el desarrollo. Todo ello con el propósito de mejorar la comunicación regular, pero mitigando la necesidad de otras reuniones que puedan interferir en el desarrollo del proyecto; e intentar identificar cualquier impedimento lo antes posible para su solución.

Revisión de Sprint (Sprint Review): Al finalizar el Sprint, se revisa el trabajo completado y el no completado. El trabajo completado se presenta al Dueño de Producto o a otros interesados en el resultado del proyecto. Con este resultado, se revisa la Lista de Tareas de Producto, para modificar o re-priorizar las tareas en esa lista con la vista puesta en la planificación del siguiente Sprint.

Simplificación de la Metodología

Una vez se ha presentado la metodología de desarrollo de software Scrum, en la presente sección se introducirán las distintas simplificaciones que se realizan en la metodología para su aplicación al presente proyecto, teniendo en cuenta el contexto y la composición del equipo.

Equipo

Por el contexto en el que se encuadra el presente proyecto, debido a la cantidad de recursos disponibles, el equipo relacionado con el proyecto se simplifica de la siguiente manera:

Equipo de Desarrollo En el presente proyecto, el Equipo de Desarrollo está constituido, únicamente, por su autor. Por lo tanto, sus responsabilidades dentro del proyecto son:

- Colaborar en la confección de la Lista de Tareas de Producto, principalmente desde un punto de vista técnico; y estimando el esfuerzo necesario para cada una de ellas
- Elaborar la lista de Tareas de Sprint, estimando su esfuerzo
- Llevar a cabo las tareas del sprint, implementando las funcionalidades propuestas en ellas

Dueño de Producto y Maestro de Scrum Respecto a los papeles de Dueño de Producto y Maestro de Scrum, en este caso son realizados por una sola persona. Por lo tanto, sus tareas son las siguientes:

- Colaborar con los usuarios finales para comprender los requisitos del sistema
- Mantener la Lista de Tareas de Producto, añadiendo las tareas a realizar respecto a los requisitos
- Priorizar la Lista de Tareas de Producto
- Analizar los resultados obtenidos al finalizar el Sprint
- Organizar todos los eventos y reuniones de cada fase del desarrollo
- Facilitar el desarrollo, eliminando cualquier impedimento que pudiera presentarse

Equipo de Investigación Adicionalmente, existe un Equipo de Investigación, compuesto por dos investigadores que eran usuarios finales de la plataforma resultado de este proyecto. Por lo tanto, su responsabilidad en lo que atañe al presente proyecto es:

- Utilizar la plataforma en su investigación, para su aplicación en diferentes entornos de problema
- Colaborar con el Dueño de Producto en la elaboración y priorización de la Lista de Tareas de Producto

Documentos

Puesto que el equipo para el desarrollo es de pequeño tamaño, los documentos presentes en la metodología Scrum también son simplificados al máximo, sin dejar de ser útiles en el desarrollo del proyecto. En la presente sección se introducen los utilizados en el presente proyecto.

Lista de Tareas de Producto En este caso, la Lista de Tareas de Producto, mantenida por el Dueño de Producto, es una lista simple de tareas ordenadas por prioridad, e incluyendo una estimación inicial del esfuerzo necesario para su completado.

Esta lista es revisada y re-priorizada en la Revisión de Sprint por todos los integrantes del Equipo.

En el cuadro 2.1 se puede observar, como ejemplo, la Lista de Tareas de Producto inicial.

```
[ ] Algoritmo: DQN (2 semanas)
[ ] Almacenamiento del Estado del Experimento (1 semana)
[ ] Análisis: Valores de Estado (0.5 semanas)
[ ] Análisis: Rendimiento (0.25 semanas)
[ ] Algoritmo: Memoria de Repetición Priorizada (1 semana)
[ ] Algoritmo: Arquitectura en Duelo (0.5 semanas)
[ ] Algoritmo: A3C (3 semanas)
[ ] Análisis: Puntos de Referencia (0.25 semanas)
```

Cuadro 2.1: **Lista de Tareas de Producto:** Lista simple de tareas en orden de prioridad y con su estimación de esfuerzo. En este caso se muestra, como ejemplo, la Lista de Tareas Inicial de Producto, que incluye los requisitos iniciales que se deseaban en el resultado del proyecto. Esta lista es revisada y re-priorizada en la Revisión de Sprint, al finalizar cada Sprint.

Lista de Tareas de Sprint Partiendo de la Lista de Tareas de Producto presentada en la sección anterior, durante la Planificación de Sprint se seleccionan las siguientes tareas a desarrollar que entrarán durante el Sprint. Una vez seleccionadas son divididas, a su vez, en tareas de duración más reducida, para generar la Lista de Tareas de Sprint, estimando su duración.

Al igual que la Lista de Tareas de Producto, la Lista de Tareas De Sprint es una lista simple ordenada, en su caso, por orden de ejecución. Se puede observar un ejemplo de tal lista en el cuadro 2.2.

```
[ ] Q-Network (16 hr)
[ ] Agente Generador de Experiencia (8 hr)
[ ] Transformación Observación-Estado (8 hr)
[ ] Memoria de repetición (16 hr)
[ ] Entrenador (16 hr)
[ ] Rendimiento: Implementación en Tensorflow (16 hr)
```

Cuadro 2.2: **Lista de Tareas de Sprint:** Lista simple de tareas en orden de ejecución y con su estimación de esfuerzo, para las tareas a realizar en el Sprint actual. En este caso se muestra, como ejemplo, la Lista de Tareas Inicial de Sprint para implementar el algoritmo DQN. Esta lista es creada en la Planificación de Sprint, al comienzo de cada Sprint.

Flujo de Trabajo

La presente sección expone la implementación del flujo de trabajo basado en Scrum que se utiliza en el proyecto.

Sprint De nuevo, se adapta la metodología Scrum para el reducido tamaño del equipo. También se tiene en cuenta la importancia de que las características implementadas lleguen cuanto antes al equipo de usuarios finales para aplicar a la investigación en curso.

Por todo ello, pese a que la metodología Scrum opta por Sprints de duración fija, aunque flexible según avanza el proyecto; en este caso se opta por Sprints de duración variable, y donde la duración del Sprint viene marcada por la tarea o tareas seleccionadas de la Lista de Tareas de Producto, tomando como referencia que cada Sprint durase sobre 1 ó 2 semanas.

Las siguientes secciones exponen las actividades que se realizan durante el desarrollo del Sprint.

Planificación de Sprint

Como ya se expuso en el apartado correspondiente, en la reunión de Planificación de Sprint participan todos los miembros del equipo. En este caso, todos los miembros relacionados con el presente proyecto se reúnen para esta planificación, tanto el Equipo de Desarrollo, el Dueño del Producto y los integrantes del Equipo de Investigación.

En esta reunión se realizaban las acciones expuestas anteriormente: seleccionar de la Lista de Tareas de Producto la siguiente o siguientes tareas a realizar en el Sprint, dependiendo de la duración estimada, para crear el Sprint de 1 ó 2 Semanas, dividir en tareas más pequeñas y estimar su esfuerzo para crear la Lista de Tareas de Sprint.

Reunión Diaria

A la reunión diaria asisten tanto el Equipo de Desarrollo como el Dueño de Producto, durante el menor tiempo posible (alrededor de 15 minutos), donde se discuten el desarrollo del Sprint y el trabajo planificado de la jornada.

Durante esta reunión, se valora también la cancelación del Sprint, en caso de que sea necesario, por algún cambio en los requisitos que ha llevado al Dueño de Producto a re-priorizar la Lista de Tareas de Producto. En este caso, simplemente, se organizaría inmediatamente la Revisión de Sprint, para planificar el siguiente con las nuevas prioridades.

Revisión de Sprint

Al igual que en la Planificación de Sprint, en la Revisión de Sprint participan todos los miembros del equipo: el Equipo de Desarrollo, el Dueño del Producto y los integrantes del Equipo de Investigación.

El trabajo realizado durante el Sprint es presentado, de forma que las nuevas funcionalidades puedan ser utilizadas inmediatamente por el Equipo de Investigación.

Posteriormente, se revisa la Lista de Tareas de Producto y se añaden o re-priorizan las tareas necesarias para comenzar el Sprint siguiente.

Capítulo 3

Algoritmos

Este capítulo presenta diferentes conceptos, comunes a los algoritmos de Aprendizaje por Refuerzo, y expone los algoritmos que se implementan en la plataforma objeto de este proyecto.

Aprendizaje por Refuerzo

Esta sección expone brevemente los conceptos de Aprendizaje por Refuerzo que se utilizarán para la posterior presentación de los algoritmos.

Como ya se comentó en la introducción, en el Aprendizaje por Refuerzo (sección 1.1), al sistema o agente sólo se le informa sobre un objetivo que lograr. El agente debe entonces, autónomamente y mediante prueba y error, interaccionar con su entorno y encontrar la mejor manera de lograr el objetivo.

Como vemos en el diagrama 3.1, en un sistema de Aprendizaje por Refuerzo se pueden distinguir los siguiente participantes:

Entorno: El entorno del problema que se quiere resolver y que puede ser, en ocasiones, un modelo o incluso el problema real; en el que se pueden realizar acciones para interactuar con él.

Intérprete: Que puede observar el entorno e interpretar el estado en el que se encuentra y calcula una recompensa, como medida de lo cercano que se está al objetivo.

Agente: Que es el objeto del entrenamiento, recibiendo el estado y la recompensa y decidiendo las acciones a tomar en el entorno.

El comportamiento del sistema suele modelarse en pasos discretos; en los que, en cada momento t:

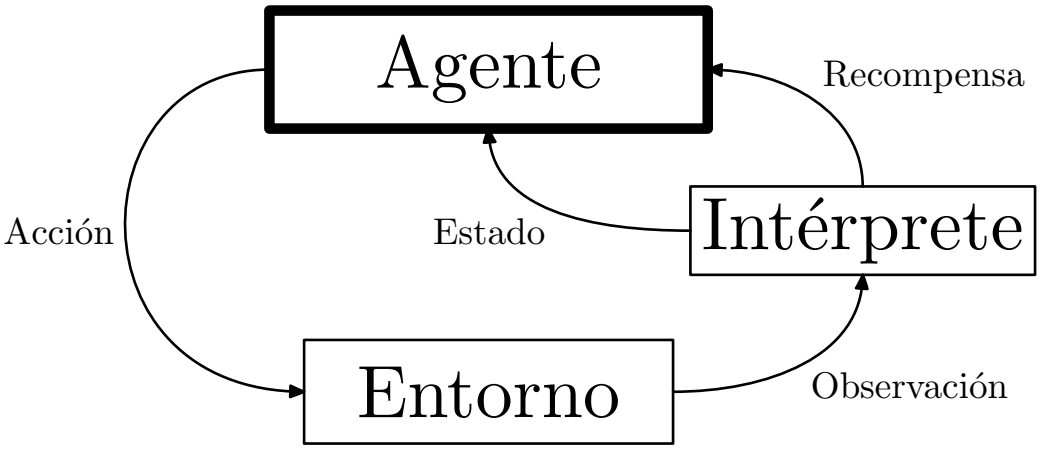


Figura 3.1: **Aprendizaje por Refuerzo:** Diagrama de los participantes en los algoritmos de aprendizaje por refuerzo.

1. El agente recibe un estado s_t, y una recompensa r_t
2. El agente escoge una acción a_t del conjunto de acciones disponibles y la envía al entorno
3. El entorno ejecuta la acción a_t y su estado pasa, entonces a s_{t+1}
4. El intérprete recibe, entonces una observación o_{t+1}, interpreta el nuevo estado s_{t+1} y calcula la recompensa r_{t+1} correspondiente a la transición (s_t, a_t, s_{t+1})

Otro término adicional es la política, que es la que determina la acción a realizar, dependiendo del estado en el que se encuentra el entorno. Se representa como un mapeado de estados a acciones, y que da la probabilidad de que el agente tome una acción determinada cuando el entorno se encuentra en un estado determinado. Finalmente, el agente debe aprender cuál es la política para obtener la mayor recompensa posible.

Adicionalmente, todo el comportamiento del sistema puede ser estocástico, lo que aumenta la dificultad del problema. Por ejemplo, si el sistema está en un estado determinado y realiza una acción, el estado siguiente resultante no tiene que ser siempre el mismo, sino seguir algún tipo de distribución aleatoria desconocida. El agente tiene entonces que aprender cuáles son las acciones que le pueden maximizar la recompensa esperada.

Además, el comportamiento del sistema suele dividirse en Episodios en los que el entorno comienza en un estado inicial y termina con un estado final determinado. El intérprete determina también si un episodio está en proceso o ha terminado.

Para entender mejor este proceso, un ejemplo clásico es un sistema que tenga que aprender a andar en bicicleta. El objetivo del sistema será avanzar con la bicicleta sin caer al suelo. Pensemos que, en los primeros pasos del entrenamiento, el agente realiza unas acciones y la bicicleta está excesivamente inclinada hacia la izquierda 45^o. Entonces el agente puede decidir si girar el manillar a izquierda o derecha. Inicialmente, lo mueve a la izquierda, que hace que la bicicleta caiga, y el agente recibe un refuerzo negativo en forma de recompensa. En algún momento posterior, vuelve a encontrarse, de nuevo, inclinado hacia la izquierda 45^o; como comprobó que girar hacia la izquierda era negativo, en este caso gira el manillar hacia la derecha. De nuevo, la bicicleta vuelve a caer, recibiendo un nuevo refuerzo negativo. Con todo esto, el sistema ya ha comprobado que girar a izquierda o derecha cuando el estado es «inclinado 45^o a la izquierda» es negativo; pero también sabe que haber llegado a ese estado es negativo. Si, en un momento posterior, la bicicleta está inclinada 40^o a la izquierda, y el sistema decide girar el manillar a la derecha, la bicicleta terminará estando inclinada 45^o a la izquierda y acabará cayendo y recibiendo el refuerzo negativo, por lo que el agente sabrá que no debe girar el manillar a la derecha estando inclinado 40^o a la izquierda. Realizando este proceso infinidad de veces, el agente de Aprendizaje por Refuerzo podrá, finalmente, realizar la acción de manejar la bicicleta sin que caiga al suelo. [12]

Basado en valor

Esta sección presenta el algoritmo DQN, como representativo de la familia de algoritmos basados en valor, y que es implementado en el presente proyecto.

Para comenzar, se define el valor de un estado como la suma de las recompensas que se recibirán estando en ese estado y siguiendo una política hasta un estado terminal. Así definido, el valor da una medida de lo «bueno» que es estar en un determinado estado. Por lo tanto, se define la función del valor de un estado V respecto al estado s, siguiendo una política π, como el valor esperado de las recompensas descontadas:

$$V^{\pi}(s) = E[R|s,\pi] = E[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s] \tag{3.1}$$

Y γ es un valor positivo y menor o igual que 1, conocido como Tasa de Descuento («Discount Rate»), y que implementa el concepto de mayor valor de las recompensas immediatas frente a las futuras.

Una vez definidos estos conceptos, podemos decir que el problema del Aprendizaje por Refuerzo será aprender la política óptima que maximice esta suma de recompensas cuando el sistema se encuentre en un estado arbitrario.

Finalmente, los algoritmos basados en valor mantienen una estimación de estos valores para decidir las acciones a tomar, estimando cuál de las acciones lleva a un nuevo estado con la estimación de esos valores más alta.

DQN

De todos los algoritmos basados en valor, DQN es el más extendido actualmente en el aprendizaje por refuerzo. [16]

Para este algoritmo, inicialmente, se definen valores adicionales (similares a los anteriores, pero dependientes de la acción) denominados valores Q («Q Values»), dependientes del estado s y la acción a, como:

$$Q^{\pi}(s,a) = E[R|s,a,\pi] \tag{3.2}$$

El valor óptimo de Q obedece la llamada ecuación de Bellman, que encuentra este valor óptimo de Q, $Q^*(s_t,a_t)$ teniendo en cuenta que la estrategia óptima es seleccionar la siguiente acción a_{t+1} que maximiza el valor esperado de $r + \gamma Q^*(s_{t+1},a_{t+1})$:

$$Q^*(s_t,a_t) = E[r_t + \gamma max_{a_{t+1}} Q^*(s_{t+1},a_{t+1})|s_t,a_t] \tag{3.3}$$

Las técnicas que tratan de calcular o estimar estos valores Q para decidir la política óptima a aplicar en cada estado se denominan Aprendizaje-Q («Q-Learning»). Estas técnicas se basan en la identidad anterior para estimar estos valores Q de forma iterativa, en el paso i, como:

$$Q_{i+1}(s_t,a_t) \leftarrow Q_i(s_t,a_t) + \alpha \cdot (r_t + \gamma \cdot max_{a_{t+1}} Q^*(s_{t+1},a_{t+1}) - Q(s_t,a_t)) \tag{3.4}$$

Siendo α un valor positivo y menor o igual que 1, conocido como Tasa de Aprendizaje («Learning Rate»), y que modula la cantidad de cambio que se realizará en cada paso del algoritmo.

Lo que diferencia a los algoritmos dentro del Aprendizaje-Q es la técnica que utilizan para calcular o estimar estos valores Q. En el caso del algoritmo DQN («Deep Q-Network») se utiliza una red neuronal para estimar los valores, que es entrenada mediante la expresión anterior. Una vez se han estimado esos valores Q de los estados, se pueden utilizar para aproximarse a la política óptima, simplemente escogiendo la acción con el valor más alto de $Q(s_t,a_t)$.

Respecto a la exploración, en el algoritmo DQN se suele utilizar una técnica «Epsilon-greedy». En esta técnica, se define un término ϵ que es utilizado en el entrenamiento y con el que, cuando se decide la acción a tomar, se toma la acción indicada por la política del agente con una probabilidad $1 - \epsilon$ o una acción uniformemente aleatoria con una probabilidad ϵ.

Las siguientes secciones detallan características del algoritmo y opciones que mejoran el comportamiento en ciertos casos.

Repetición de experiencia

Uno de los problemas en el Aprendizaje por Refuerzo es cómo generar la suficiente cantidad de datos para poder realizar el entrenamiento iterativo mostrado en 3.4 el suficiente número de veces para que el sistema pueda aprender de esa experiencia, eliminando correlaciones que puedan alterar los valores deseados.

Para ello se implementa la Repetición de Experiencia («Experience Replay»), en el que la experiencia es almacenada en forma de transición (s_t, a_t, s_{t+1}), y que se extrae aleatoriamente para ser utilizada en el algoritmo. Estas transiciones se almacenan en la Memoria de Repetición («Replay Memory»), que consiguen entregar la experiencia de forma aleatoria y eficiente.

El tamaño de la Memoria de Repetición debe cumplir con dos requisitos al mismo tiempo:

- Ser lo suficientemente grande para que las transiciones extraídas aleatoriamente estén lo suficientemente separadas para que la correlación sea pequeña entre ellas
- Ser lo suficientemente pequeña para que transiciones antiguas, en los que la política del agente no era suficientemente buena, no se utilicen en el entrenamiento

Mejoras implementadas

Una vez el algoritmo DQN se generalizó, han ido surgiendo estudios posteriores que detallan mejoras para corregir ciertas características indeseables en él. En la siguiente lista se detallan algunas de esas mejoras, implementadas en la plataforma:

DQN Doble: Es conocido que el algoritmo DQN suele sobreestimar los valores Q, y eso puede llevar a que la política que se sigue no sea la óptima. Por lo tanto, y para mitigar este efecto, se implementa en el sistema

el DQN Doble [27], que utiliza una red neuronal adicional para calcular la acción a realizar en el entrenamiento y que va sincronizándose periódicamente con la ya existente.

Repetición de Experiencia Priorizada: En el algoritmo DQN, las transiciones a utilizar en el entrenamiento son escogidas aleatoriamente según son generadas por el agente. Sin embargo, se ha propuesto que si se priorizan las transiciones a utilizar para entrenar, extrayendo más a menudo las transiciones de las que se ha aprendido un resultado más útil [19], el entrenamiento puede ser más efectivo.

Basado en política

Una alternativa a los algoritmos basados en valor son los algoritmos basados en política. Estos últimos tratan de extraer la política óptima, no manteniendo una lista de valores de los que extraer esta política; sino, directamente, buscando y manteniendo esta política; estimando la distribución de las acciones que maximiza la esperanza de recompensa.

De ellos, el que se está demostrando últimamente como más efectivo es el «Asynchronous Advantage Actor-Critic» (A3C), que se presenta en la siguiente sección.

A3C

En el algoritmo A3C, se mantiene también una estimación del valor del estado $V(s)$, que es llamada la parte «Critic», pero que no se utiliza para calcular la política a utilizar sino, simplemente, para ser utilizada en la actualización de la política (la parte «Actor»), que es devuelta directamente por la red y es la que se utilizará para la decisión de las acciones a tomar. [15] En el corazón de este algoritmo, y al igual que en DQN, estas estimaciones son realizadas por una Red Neuronal, que es la que debe ser entrenada para optimizar la poítica y la estimación de los valores de estado.

Para este entrenamiento, el algoritmo define una función de pérdida que, al ser minimizada, permite que la Red Neuronal vaya estimando la política a utilizar. Como resumen, podemos decir que tal función de pérdida está compuesta de tres términos que, minimizados en su conjunto, permiten estimar tal política. Estos son:

Pérdida en política: O «Policy Loss», que describe la diferencia entre la política actual y la política óptima que se descubre durante el entrenamiento que da mejor resultado.

Pérdida en valor: O «Value Loss», que describe la diferencia entre el valor estimado actual del estado y el valor que se le calcula en ese momento.

Entropía: El término de la entropía en la pérdida añade exploración al entrenamiento. Este término permite que las probabilidades de escoger una determinada acción en un estado estén lo suficientemente repartidas para que se exploren alternativas que, seguramente, no se probarían de no existir este término.

Generación de experiencia asíncrona

En el algoritmo A3C, en lugar de utilizar la repetición de experiencia como en DQN (sección 3.2.2) para el uso eficiente de transiciones generadas por el entorno del que se pueda aprender, el algoritmo A3C instancia múltiples copias del entorno del que obtener esas transacciones asíncronamente. De esa forma, se consigue que la generación de transiciones para el entrenamiento sea eficiente, pero que esas transiciones no estén correladas, lo que haría decrecer la calidad del entrenamiento, al ser generadas por entornos independientes.

Capítulo 4

Herramientas y librerías

El presente capítulo presenta las diferentes herramientas y librerías utilizadas en el desarrollo de este proyecto. Se presentarán cada una de ellas, incluyendo, asimismo, las características más importantes aplicables al proyecto y sus ventajas ante otras alternativas.

Python

Para todo el desarrollo del proyecto, se ha utilizado el lenguaje de programación de propósito general Python [10] en su versión más actualizada (3.6).

El uso de Python como lenguaje de programación se ha ido generalizando en todos los sistemas aplicados a la Inteligencia Artificial. Toda una serie de características lo hacen óptimo para este tipo de aplicaciones:

Orientado a objetos y sintaxis familiar: Lo que permite a investigadores con menos experiencia en programación desarrollar sistemas de Inteligencia Artificial.

Legibilidad: Python fue diseñado desde un principio enfocado en la legibilidad del código, por lo que resulta apropiado para la implementación y mantenimiento de algoritmos con cierta complicación, como los utilizados en Inteligencia Artificial.

Estable y establecido: Surgido en 1991, cuenta con un historial de versiones estables y ampliamente utilizadas que avala su uso en producción.

Extensa documentación y Comunidad grande y activa: La alta cantidad de usuarios permite que sea sencillo resolver cualquier duda o problema participando en la comunidad surgida a su alrededor.

Interpretado y de Tipado dinámico: Lo que permite un prototipado rápido, y un ciclo ágil hasta la liberación de nuevas versiones de los sistemas.

Buen rendimiento: Pese a ser un lenguaje de alto nivel e interpretado, las implementaciones de Python más utilizadas suelen compilar a código intermedio, lo que permite que el rendimiento de las aplicaciones sea cercano al código compilado.

Multiplataforma: Existen implementaciones para todas las principales plataformas, lo que permite su despliegue y uso, sea cual sea el sistema en el que se quiere desplegar en producción.

Código abierto: Con todas las garantías que da el acceso libre al código fuente, en cuanto a mantenibilidad y garantía de futuro.

Librerías numéricas y de análisis: Muy estables y ampliamente utilizadas, tanto en la Inteligencia Artificial como en otros campos técnicos y científicos. Ejemplos de estas librerías (la mayoría de las cuales han sido utilizadas en este proyecto), son:

Numpy: Librería para cálculo vectorial, matricial y tensorial

Matplotlib: Para representación gráfica numérica

SciPy: Colección de herramientas y algoritmos matemáticos

Scikit-learn: Librería de diferentes algoritmos de Aprendizaje Automático

TensorFlow: Librería para Inteligencia Artificial, de cálculo tensorial mediante grafos de flujo de datos.

Alternativas

En el campo de la Inteligencia Artificial, la alternativa a Python más utilizada es R [25]. R es un lenguaje de programación orientado al análisis estadístico, pero que dispone de librerías dedicadas al Aprendizaje Automático ampliamente utilizadas. Sin embargo, y precisamente por esa especialización al análisis estadístico, implementar alguna de las características deseadas de la plataforma objeto del proyecto habría sido más complejo.

Otro lenguaje valorado para su uso en la Inteligencia Artificial es el C++ [11], como Lenguaje de propósito general orientado a objetos. Su uso es habitual, especialmente en robótica, y cuando se requiere intermediar con el

mundo real mediante sensores y actuadores, ya que los drivers de estos sistemas suelen requerir ser escritos en lenguajes de más bajo nivel. Sin embargo, como este no es el caso del proyecto que nos ocupa, y el C++ es, inicialmente, un lenguaje más complicado de utilizar, fue descartado para el desarrollo de esta plataforma.

TensorFlow

TensorFlow [1, 13] es una librería de código abierto para el Aprendizaje Automático, que permite el diseño y explotación de Redes Neuronales y aplicar los algoritmos de Aprendizaje Automático para el entrenamiento de estas redes. TensorFlow basa su uso en un sistema de grafos, definidos mediante operaciones matemáticas aplicadas a tensores de múltiples dimensiones, que permite implementar las Redes Neuronales; y que, al ser compilados y, opcionalmente, ejecutados en hardware específico, dotan de alto rendimiento a estas redes.

Su adopción tanto para investigación como para el despliegue en producción de estos sistemas se ha generalizado ampliamente en los últimos tiempos, gracias a la serie de características de las que dispone:

Amplia documentación: Los manuales y tutoriales disponibles para ella hacen que empezar sea sencillo, pese a que maneje conceptos y arquitecturas más o menos complejas.

Gran comunidad: La gran adopción de esta librería permite que encontrar respuesta a cualquier duda o problema sea sencillo.

Código abierto: Con el código disponible, si es necesario acceder a él para cualquier duda o problema. Sin embargo, cuenta con grandes corporaciones colaborando en su mantenimiento, lo que garantiza su continuidad.

Soporte de ejecución en CPU o GPU (Unidad de Procesamiento Gráfico): Permitiendo ejecutarse en diferentes arquitecturas hardware o de diferentes formas fácilmente, y permitiendo optimizar el rendimiento de cálculo en el hardware existente.

Soporte de características avanzadas: Como múltiples GPU o ejecución distribuida, que facilita futuras mejoras, si son necesarias.

Alternativas

Otra librería ampliamente utilizada en el campo del Aprendizaje Automático es Theano [6, 26]. Pese a que su lanzamiento fue anterior a la mencionada TensorFlow, y a que coincide con ella en muchas de sus características, su adopción, por el momento, es menor que en la anterior, luego es un poco más complicado obtener ayuda en caso de algún problema. Igualmente, al no contar con el apoyo de grandes corporaciones, puede tener más riesgo de carecer en un futuro de mantenimiento.

Otra alternativa sería la librería Scikit-learn, que dispone de gran cantidad de algoritmos desarrollados para el Aprendizaje Automático. Sin embargo, su falta de soporte para GPU hace que sea complicado alcanzar el rendimiento necesario para problemas de cierto tamaño.

Keras

Aunque, inicialmente, cualquier Red Neuronal necesaria en el desarrollo del presente proyecto pudiera ser diseñada e implementada únicamente utilizando TensorFlow, la interfaz de éste hace que definir tales redes puedas ser un proceso complicado y propenso a errores. Por lo tanto, es útil el uso de alguna herramienta que facilite el uso de esta librería.

Keras [28] es una interfaz para TensorFlow de más alto nivel y más sencillo uso. Permite definir Redes Neuronales de forma mucho más sencilla que en TensorFlow, ya que cuenta con las arquitecturas de Redes Neuronales más utilizadas en la actualidad. Sin embargo, también facilita el acceso a la capa de bajo nivel, si es necesario, para realizar retoques a nivel de TensorFlow que permitan implementar características más avanzadas o mejorar el rendimiento.

Por lo tanto, Keras permite manejar Redes Neuronales de forma mucho más amigable, pero es también extensible, permitiendo añadir nueva funcionalidad mediante código TensorFlow. Adicionalmente a TensorFlow, también soporta Theano como capa de bajo nivel, por lo que sería posible cambiar de librería si fuese interesante en un futuro.

Por último y, al igual que las anteriores, cuenta con una licencia de código abierto, que permite el acceso al código fuente y a su modificación.

Alternativas

Inicialmente, TensorFlow incluye dos alternativas similares, que pueden ser utilizadas como interfaz de más alto nivel: tf.layers y tf.nn. Como des-

ventaja de utilizar estas alternativas, se puede nombrar que su interfaz es un poco más compleja si no se está familiarizado con TensorFlow, lo que hace que sea más complicada de utilizar. Y, como principal ventaja, al estar integradas en TensorFlow se puede acceder inmediatamente tras haberlo instalado, sin pasos adicionales. Esta última ventaja está disminuyendo su importancia, puesto que Keras está siendo integrada, igualmente, como extensión en TensorFlow; lo que haría que acceder a ella fuera igual de sencillo.

OpenAI Gym

OpenAI Gym [8, 17] ofrece un conjunto de herramientas para el desarrollo y prueba de algoritmos de Aprendizaje Automático. Básicamente, consta de una serie de entornos de problema que pueden utilizarse fácilmente para la investigación y comprobación de estos algoritmos. Estos entornos están definidos siguiendo una interfaz que estandariza la definición de los entornos, y permite cambiar uno por otro con facilidad; y hace más reproducibles los resultados de cualquier experimento. Dicha interfaz es compatible con cualquier algoritmo de Aprendizaje por Refuerzo Profundo y cualquier librería de computación de las mencionadas anteriormente (TensorFlow, Theano...).

La lista de entornos que ofrece actualmente es muy completa y de muy variada dificultad y continúa creciendo. La siguiente lista ofrece ejemplos de tipos de entornos disponibles:

Sistemas de control sencillos: Con motores de física simples, y pocas acciones a realizar.

Sistemas de control 2D y 3D: Donde el motor de física y las acciones son más complejas; y que incluye problemas como controlar que un robot ande o salte, por ejemplo.

Juegos de Atari: Juegos clásicos 2D de la consola Atari 2600, que han sido tradicionalmente usados para la investigación de los algoritmos de Aprendizaje por Refuerzo Profundo.

Doom: Como ejemplo de Juego 3D con un grado de dificultad superior a los de Atari.

Juegos de Mesa: Como el Go, donde la resolución mediante Inteligencia Artificial es compleja por la cantidad de combinaciones posibles para cada jugada.

Igualmente, aunque no es un fin en el presente proyecto, OpenAI Gym permite compartir y reproducir fácilmente los resultados de los algoritmos, para su comparación, mediante su página e interfaz web.

Para el desarrollo del proyecto se han utilizado dos entornos para el desarrollo y prueba de los algoritmos y el sistema implementado (apéndice A):

CartPole: Como entorno más sencillo de control; en el que se controla un carro que puede moverse unidimensionalmente, que cuenta con un poste vertical que puede girar libremente. El objetivo del entorno es controlar el carro, de forma que el poste se mantenga vertical y evitar que se caiga, el mayor tiempo posible.

Pong: Como entorno más complejo y que incluye información visual, puesto que se basa en la imagen en pantalla. Es un juego de la videoconsola Atari 2600, en el que el sistema debe controlar uno de los jugadores y conseguir la puntuación más alta posible.

Alternativas

No existen, actualmente, muchas alternativas que ofrezcan la variedad de entornos de OpenAI Gym. Se puede mencionar DeepMind Lab [3, 9], que ha sido lanzada en estas últimas fechas pero que, por su novedad, no ha sido tan probada como la anterior. Asimismo, DeepMind Lab está especializada en entornos 3D, luego ofrece menos selección que OpenAI Gym.

Matplotlib

Para el análisis gráfico en el sistema, se ha utilizado la librería Matplotlib [2, 24]. Matplotlib es una librería de representación gráfica 2D para la publicación de gráficos con calidad de publicación en multitud de formatos y multiplataforma. Cuenta con una interfaz Python sencilla para la generación de los gráficos, y un sistema de configuración potente, si resulta necesario personalizar el resultado.

Alternativas

Existen otras librerías de representación gráfica con interfaces en Python, de código abierto y con un aspecto más moderno y visualización interactiva, como Bokeh o Plotly; pero, en este caso, se decidió por Matplotlib por ser más estable y tener funcionalidad más que suficiente para la representación gráfica en el sistema.

Otra alternativa sería realizar el análisis en entornos dedicados de análisis matemático o estadístico como MATLAB o R, que cuentan con gran funcionalidad para ese tipo de análisis; pero la integración con el sistema se habría complicado en exceso para la funcionalidad requerida para el presente proyecto, por lo que fueron descartados.

Capítulo 5

Implementación

El presente capítulo expone las partes más relevantes de la implementación, centrándose, ante todo, en aquellas partes que inciden directamente en los objetivos del proyecto.

Algoritmos

En el presente proyecto fueron implementados dos de los algoritmos más utilizados actualmente en el aprendizaje por refuerzo. Las siguientes secciones presentan las características de la implementación de cada uno de ellos: DQN y A3C.

DQN

Como ya se comentó para el algoritmo DQN, su base es la utilización de Redes Neuronales para la estimación de los Valores Q, denominadas Redes Q («Q-Networks»). En este caso, para su implementación se utilizó la librería Keras (sección 4.3), que facilita enormemente la creación de redes neuronales sin entrar a su composición. En el cuadro 5.1, se puede ver un ejemplo de implementación de una red neuronal, utilizada en el presente proyecto.

La segunda parte del algoritmo DQN es el método iterativo del Aprendizaje Q (expresado en 3.4). En el cuadro 5.2 se puede ver la parte central de la implementación de tal método, el cálculo del error entre el valor Q actual estimado, y el nuevo valor Q que se ha calculado. Este error es el que se trata de minimizar iterativamente, para ir llegando a mejores estimaciones de los valores Q. De nuevo, Keras implementa distintos algoritmos de optimización que se utilizan en este proyecto para ir minimizando ese error.

```
create_cnn_fc_layers(
    cnn_layers=[
        k.layers.Conv2D(filters=32, kernel_size=8, strides=4,
            name='CNN1'),
        k.layers.Activation(activation=k.activations.relu,
            name='RELU_CNN1'),

        k.layers.Conv2D(filters=64, kernel_size=4, strides=2,
            name='CNN2'),
        k.layers.Activation(activation=k.activations.relu,
            name='RELU_CNN2'),

        k.layers.Conv2D(filters=64, kernel_size=3, strides=1,
            name='CNN3'),
        k.layers.Activation(activation=k.activations.relu,
            name='RELU_CNN3')],
    fc_layers=[
        k.layers.Dense(units=256, name='Dense'),
        k.layers.Activation(activation=k.activations.relu,
            name='RELU_Dense')])
```

Cuadro 5.1: **Creación de Red Neuronal para DQN:** Se muestra la creación de una red neuronal que será utilizada en el algoritmo DQN. En este caso, se trata de la creación de una red neuronal compuesta de 3 capas de Redes Neuronales Convolucionales (CNN), y una capa completamente conectada («Fully-connected»).

Repetición de experiencia

Como ya se mostró en la sección 3.2.2, la repetición de experiencia requería la implementación de una Memoria de Repetición, para el almacenamiento de las transiciones generadas por el agente sobre el entorno, que luego utilizar en el entrenamiento. En este caso, se utilizó un vector circular para su implementación, ya que permite un acceso aleatorio a sus elementos con comportamiento $O(1)$. Al mismo tiempo, permite añadir y eliminar de sus extremos con el mismo comportamiento $O(1)$, con lo que es eficiente expirar las transiciones más antiguas, para que no sean utilizadas en el entrenamiento.

DQN Doble Como se puede ver en el cuadro 5.3, se implementa también en el presente proyecto el algoritmo DQN Doble, creando una Red Neuronal adicional para el cálculo de los valores objetivo a optimizar.

```
q_values = q_network.predict(state)
q_value = q_values[action]

next_q_values = q_network.predict(next_state)
next_q_value = max(next_q_values)

target_q_value = reward + (discount_factor * next_q_value)

td_error = q_value - target_q_value
```

Cuadro 5.2: **Cálculo del error en el valor Q:** Se muestra el cálculo del error utilizado en el algoritmo DQN. Este error será el que se tratará de minimizar iterativamente mediante la modificación de los parámetros de la Red Neuronal.

Repetición de experiencia priorizada La base de la repetición de experiencia priorizada es la implementación de una memoria de repetición distinta, en la que se puedan seleccionar las transiciones almacenadas aleatoriamente pero siguiendo una distribución en base a una prioridad almacenada junto a esas transiciones. Para conseguir realizar eso de forma eficiente, se implementa la memoria de repetición priorizada en forma de árbol, almacenando la suma de las prioridades de los elementos en cada uno de los subárboles. De esa forma, se consigue que extraer n transiciones siguiendo la distribución deseada tenga un comportamiento $O(n \log n)$.

A3C

Al igual que DQN, A3C está basado en la utilización de Redes Neuronales para la estimación de la política a utilizar y el valor del estado. Podemos ver en el cuadro 5.4, un ejemplo de implementación de una red neuronal para tal algoritmo, utilizada en el presente proyecto.

La otra característica del algoritmo A3C es la función de pérdida que, siendo minimizada, permite la mejora de la política y de la estimación del valor del estado. Se puede ver en el cuadro 5.5 el código para su cálculo.

Generación de experiencia asíncrona

Finalmente, la última característica del algoritmo A3C es la generación de experiencia asíncrona. Como se puede observar en el cuadro 5.6, la implementación de múltiples hilos (threads) permite este tipo de generación, mediante el uso de múltiples entornos independientes generando transiciones

```
q_values = training_q_network.predict(state)
q_value = q_values[action]

next_q_values = target_q_network.predict(next_state)
next_q_value = max(next_q_values)

target_q_value = reward + (discount_factor * next_q_value)

td_error = q_value - target_q_value
```

Cuadro 5.3: **Cálculo del error en el valor Q en el Aprendizaje-Q doble:** Se puede ver cómo, comparado con el mismo cálculo en el algoritmo DQN tradicional (cuadro 5.2), se utiliza una red adicional para el cálculo de los valores objetivo.

concurrentemente. Ello permite que las transiciones utilizadas para entrenar no estén correladas entre ellas, mejorando así la calidad del entrenamiento.

Otras características

Ejecución de experimentos

Para facilitar la ejecución de experimentos, se crea la clase *Experiment* que es responsable de la ejecución completa. Simplemente recibiendo los hiperparámetros en su inicialización, inicializa el experimento, lo ejecuta y se encarga de guardar sus resultados:

```
with Experiment(**hyper_params) as experiment:
    experiment.run()
```

De esta forma, ejecutar los experimentos es tan sencillo como crear un diccionario con los hiperparámetros seleccionados y llamar al método de la clase *Experiment* desde un fichero Python.

Modificación y almacenado de los hiperparámetros

Como se indicó en el apartado anterior, la generación y modificación de los hiperparámetros consiste, únicamente, en la creación de un diccionario en un fichero Python incluyendo tales hiperparámetros. Por lo tanto, ese fichero puede ser modificado fácilmente e incluso mantenido en un sistema de control de versiones, si fuese necesario.

```
create_fc_layers(
    fc_layers=[
        k.layers.Dense(units=16, name='Dense1'),
        k.layers.Activation(activation=k.activations.relu,
            name='RELU_Dense1')],
    policy=[
        k.layers.Dense(units=output_dimension,
            name='PolicyOutput'),
        k.layers.Activation(activation=k.activations.softmax,
            name='PolicySoftmax')],
    value=[
        k.layers.Dense(units=1, name='ValueOutput')])
```

Cuadro 5.4: **Creación de Red Neuronal para A3C:** Se muestra la creación de una red neuronal que será utilizada en el algoritmo A3C. En este caso, deben resaltarse, a diferencia de la Red Neuronal para DQN (5.4), las dos entradas diferentes, la de la política (Policy) y la del valor de estado (Value).

Asimismo, los hiperparámetros utilizados son almacenados junto con los resultados del análisis del experimento en un fichero en formato JSON, de forma que se pueden comprobar fácilmente los hiperparámetros utilizados en la ejecución de cualquier experimento. En el apéndice A se pueden ver múltiples ejemplos de tales ficheros.

Análisis del desarrollo y resultado del experimento

Durante la ejecución del experimento, múltiples indicadores son calculados y almacenados para su posterior análisis y visualización o, incluso, para su visualización mientras el experimento está siendo ejecutado.

El objetivo de la plataforma objeto de este proyecto es analizar y almacenar el mayor número de indicadores relevantes, así que, actualmente, se han añadido:

- Valores Q
- Políticas
- Valores de estado
- Tasa de aprendizaje
- Recompensa

```
log_prob = k.backend.log(policy)

next_value = a3c_network.predict(next_state)
target = reward + (discount_factor * next_value)
advantage = target - value

policy_loss = -log_prob * advantage
value_loss = 0.5 * k.backend.square(advantage)
entropy = policy * k.backend.log(policy)

loss = policy_loss + (k_v * value_loss) + (k_h * entropy)
```

Cuadro 5.5: **Cálculo de la pérdida en A3C:** Donde se observan los tres términos de la pérdida que serán minimizados: la pérdida en política (policy_loss), la pérdida en valor (value_loss) y la entropía (entropy).

- Velocidad

La recompensa mencionada en la lista anterior es la recompensa recibida durante el proceso de entrenamiento. Durante este proceso el agente se comporta incluyendo una parte de exploración en la que las acciones se escogen aleatoriamente, luego no recoge exactamente el desempeño del agente siguiendo la política que ha aprendido. Por lo tanto, se han incluido puntos de referencia en los que se calcula la recompensa pero con el agente siguiendo la política sin ningún componente aleatorio, para poder medir y comparar el estado de aprendizaje en el que se encuentra el agente.

En el apéndice A se pueden ver múltiples ejemplos de la visualización de tales indicadores.

```
def create_transition_supplier_thread(transition_supplier):
    def add_transition():
        while not stop_flag:
            transition = transition_supplier.transition(step)
            queue.put(transition)

            # Yield to another thread
            time.sleep(0)

        queue.put(None)

    return threading.Thread(target=add_transition)
```

Cuadro 5.6: **Creación del hilo para la generación de transiciones:** La generación de experiencia asíncrona se consigue mediante la creación de múltiples hilos (threads) para la generación de transiciones, cada uno conteniendo una instancia diferente del entorno del problema, consiguiendo que la información generada sea independiente.

Capítulo 6

Despliegue

El presente capítulo presenta las diferentes plataformas escogidas para el despliegue del resultado del proyecto y las características que motivan su elección.

En este caso, se utilizan dos plataformas diferentes para su despliegue y ejecución: la principal, mediante servidores físicos, especialmente configurados para la ejecución de algoritmos de Aprendizaje Automático; pero también su despliegue en la nube, utilizando servidores virtuales también específicos para estas aplicaciones.

Despliegue local

Para el despliegue de la plataforma en el proyecto, se adquieren dos servidores físicos, donde se despliega la plataforma resultado de este proyecto para su ejecución y uso para la experimentación.

En el cuadro 6.1 pueden verse las características de dichos servidores.

En el caso del presente proyecto, dos son las características que más priman en la selección del servidor a adquirir:

Potencia de procesador: Puesto que los algoritmos de Aprendizaje por Refuerzo Profundo son, sobre todo, limitados por la capacidad de CPU disponible; al ser, habitualmente, los datos utilizados en el aprendizaje, generados por los entornos programáticamente. Por lo tanto, se escoge el procesador con la capacidad de cálculo y número de núcleos más altos posible.

Potencia gráfica: Estos algoritmos de Aprendizaje por Refuerzo Profundo son altamente paralelizables e implementables para su ejecución en Unidades de Procesamiento Gráfico (GPU), aumentando, de esta forma,

Dell XPS Desktop 8920	
Procesador	Intel Core i7-7700
Cache	8 MB
Frecuencia de reloj	Hasta 4.2 GHz
Memoria	16 GB 2400 MHz memoria DDR4
Disco duro 1	256 GB M.2 PCIe x4 SSD
Disco duro 2	2 TB 7200 rpm
Tarjeta gráfica	NVIDIA GeForce GTX 1060 6 GB
Sistema operativo	Ubuntu Linux 16.04 LTS

Cuadro 6.1: **Características de los servidores físicos:** Los servidores son especialmente seleccionados para contar con la tarjeta gráfica más potente posible que admitiese la ejecución de algoritmos de Aprendizaje Automático mediante el cálculo en paralelo de Redes Neuronales.

el rendimiento de tales algoritmos considerablemente. Por lo tanto, se escoge la tarjeta gráfica más potente posible, soportada por las librerías de Aprendizaje Automático.

Tarjeta gráfica

Respecto a la selección de la tarjeta gráfica, debemos reseñar que, inicialmente, las Unidades de Procesamiento Gráfico (GPU) eran dedicadas, exclusivamente, a la manipulación y creación de imágenes para su visualización a través de la pantalla o para el renderizado y edición de imágenes o vídeos. Sin embargo, está cada vez más extendido que estas unidades tengan la capacidad para ejecutar otro tipo de cálculos, aprovechando la potencia de la que se ha conseguido dotar a las modernas aceleradoras gráficas. De esta forma, se consigue incrementar el rendimiento de procesos que, como los algoritmos de Aprendizaje Automático, pueden descomponerse en cálculos vectoriales y paralelizarse, para su ejecución en la alta cantidad de núcleos disponibles en este tipo de unidades.

El cuadro 6.2 muestra las características de la tarjeta gráfica, debiendo resaltar el elevado número de núcleos disponibles para la realización de cálculo en paralelo.

Para poder aprovechar la potencia de la tarjeta gráfica, las librerías de NVIDIA Compute Unified Device Architecture (CUDA), y su extensión CUDA Deep Neural Network (cuDNN) son desplegados en el servidor.

En el caso de CUDA, es una plataforma de computación en paralelo, para aumentar el rendimiento de cálculo aprovechando, precisamente, la potencia de cálculo de las GPU instaladas; mediante la definición de una interfaz para

NVIDIA GeForce GTX 1060 6 GB	
Núcleos	1280
Frecuencia de reloj	1506 MHz
Memoria	6 GB GDDR5
Frecuencia de memoria	8 Gb/s
Ancho de banda de memoria	192 GB/s

Cuadro 6.2: **Características de la tarjeta gráfica:** La tarjeta gráfica utilizada es la más potente posible a instalar en los servidores mencionados, contando con un alto número de núcleos, para la realización de cálculo en paralelo, que puede ser aprovechado por las librerías de Aprendizaje Automático.

el acceso a esa infraestructura. Adicionalmente, cuDNN es una librería que incluye primitivas para el cálculo de Redes Neuronales aceleradas mediante GPU.

Por lo tanto, estas dos librerías son utilizadas por TensorFlow para el acceso a la potencia de cálculo de la unidad gráfica para el cálculo de los algoritmos.

Despliegue en la nube

Como alternativa al despliegue local en servidores físicos, se prueba el despliegue y ejecución de experimentos en la nube, para el caso en el que se necesite potencia extra para ello. Para realizar tal despliegue, se seleccionaron instancias de servidores virtuales de Amazon Web Services (AWS) EC2, que permiten, utilizando su interfaz web, iniciar y acceder a servidores virtuales en escaso tiempo, para escalar cualquier necesidad que haya de computación. Específicamente, se seleccionan instancias P2 diseñadas para la ejecución de algoritmos de Aprendizaje Automático, ya que cuentan con GPU dedicadas muy potentes para tal fin. En el cuadro 6.3, se mencionan las características de los servidores utilizados en el presente proyecto.

Por lo demás, una vez las instancias de servidores virtuales han sido iniciadas, el despliegue es exactamente igual que el despliegue local mencionado en la sección 6.1, ya que las librerías de acceso a la computación en GPU soportan las tarjetas gráficas de ambas formas de despliegue.

	AWS EC2: Instancia P2 xlarge
Procesador	High Frequency Intel Xeon E5-2686v4 (Broadwell)
CPU virtuales	4
Frecuencia de reloj	2.3 GHz
Memoria	61 GiB
Tarjeta gráfica	NVIDIA K80 GPU 2496 núcleos 12GiB de memoria
GPU	1
Sistema operativo	Ubuntu Linux 16.04 LTS

Cuadro 6.3: **Características de los servidores virtuales:** Los servidores virtuales seleccionados están específicamente diseñados para la ejecución de algoritmos de Aprendizaje Automático, con lo que cuentan con tarjetas gráficas especialmente creadas para esta clase de cálculos.

Alternativas

Como alternativa en el despliegue en nube, cabe mencionar que Microsoft Azure cuenta con instancias de servidores virtuales equivalentes (Instancias NC6), que disponen de características muy similares a las seleccionadas e, incluso, montan la misma tarjeta gráfica. Sin embargo, se prefiere el uso de AWS puesto que cuenta con más experiencia en el despliegue del sistema operativo seleccionado para el proyecto.

Capítulo 7

Conclusiones

Este capítulo refleja las principales conclusiones extraídas del desarrollo del presente proyecto, detallando la adecuación a los objetivos iniciales detectados en su definición.

Como ya se ha comentado en el desarrollo de la presente memoria, el objetivo de este proyecto es el desarrollo de una plataforma de experimentación de Aprendizaje por Refuerzo Profundo que cumpliese con las características detalladas en la sección 1.3, y que todavía no estaba disponible por la novedad del campo de aplicación.

Por lo tanto, todo el diseño y la implementación del proyecto han ido enfocados al cumplimiento de tales objetivos y, como tal, ha sido un éxito; pues cada una de las características que han ido lanzándose en el desarrollo de la plataforma, han ido siendo utilizadas por los investigadores para el análisis de los algoritmos en distintos entornos de problema.

Como conclusión final, las siguientes secciones detallan la consecución de los distintos objetivos de este proyecto.

Ejecución de experimentos

Mediante el uso de Python (sección 4.1) como lenguaje de programación de la plataforma, y lenguaje seleccionado para la ejecución de los experimentos, se ha conseguido que esta ejecución sea sencilla para los investigadores, sin la necesidad de que sean programadores experimentados.

Adicionalmente, y gracias a la disponibilidad de este lenguaje de programación, los experimentos pueden ser ejecutados en múltiples sistemas y arquitecturas, tanto en servidores locales, como en remoto y en servidores virtuales en la nube (sección 6).

Múltiples entornos de problema

Como se detalló en la sección 4.4, la selección de OpenAI Gym, tanto en los entornos disponibles como en el uso de su interfaz de entorno de problema, permite que cualquier tipo de entorno pueda ser integrado en la plataforma, de forma que los algoritmos de Aprendizaje por Refuerzo puedan aplicarse a él. Simplemente, en el desarrollo de nuevos entornos a medida se debe cumplir la definición de la interfaz definida, y puede ser inmediatamente utilizado.

Múltiples algoritmos

En el curso del presente proyecto, se han implementado dos de las familias más importantes de algoritmos (sección 3), que están siendo utilizados, últimamente, para resolver múltiples problemas con gran éxito. De esta forma, simplemente mediante parámetros de configuración, se pueden seleccionar una u otra familia de algoritmos (basado en valor o basado en política), y sus variantes, para su experimentación en los entornos de problemas.

Igualmente, implementando varios algoritmos, se ha demostrado que, pese a la complejidad de algunos de ellos, es posible añadir nuevos algoritmos si, llegado el caso, fuese necesario.

Modificación y almacenado de los hiperparámetros

Se vio en la sección 5.2.2, que la configuración de los hiperparámetros es realizable mediante la modificación de simples ficheros de configuración de texto y que, por lo tanto, puede ser efectuado fácilmente por los investigadores usuarios de la plataforma. Y, por su formato, puede ser almacenado y mantenido en, por ejemplo, sistemas de control de versiones, como Git.

Además, los parámetros usados son almacenados con los resultados de la ejecución de los experimentos, con lo que pueden ser analizados a posteriori, o utilizados en nuevos experimentos.

Análisis del desarrollo y resultado del experimento

Finalmente, como se comentó en la sección 5.2.3, se implementó el análisis de múltiples indicadores del resultado del experimento, con lo que es posible el análisis de su desarrollo, tanto a posteriori como mientras los experimentos están siendo ejecutados. Esto añade muchas posibilidades al análisis de los distintos algoritmos, a la selección de los hiperparámetros óptimos para el entrenamiento y al desempeño de los algoritmos al problema seleccionado para su solución.

Capítulo 8

Desarrollo futuro

El presente capítulo recoge múltiples líneas que han surgido como posibilidades de desarrollo futuro que amplíen la utilidad de la plataforma, tanto para la mejora de sus actuales características como para su aplicación con éxito a nuevos problemas.

Algoritmos adicionales

En el transcurso de este proyecto se han implementado los algoritmos principales que se están utilizando actualmente en el Aprendizaje por Refuerzo Profundo, pero han surgido nuevos algoritmos o técnicas de entrenamiento que pueden ser útiles para mejorar el aprendizaje para ciertos problemas. Citando algunos que resultan interesantes:

ACTKR: Según las últimas investigaciones, el algoritmo ACTKR [29] puede mejorar el aprendizaje de los algoritmos basados en política A3C, luego sería interesante su implementación para la experimentación.

Estrategias de evolución («Evolution Strategies»): Como alternativa a los métodos de entrenamiento basados en gradientes, se presentan en [18] métodos alternativos, que podrían mejorar los tiempos y la efectividad de los entrenamientos.

Técnicas adicionales

El objetivo del presente proyecto era la aplicación de Aprendizaje por Refuerzo, pero, en algunos entornos de problema, dada su complejidad o la longitud de los episodios, es difícil que, usando únicamente estas técnicas,

el sistema pueda ser entrenado con éxito. Esto sucede, mayoritariamente, en aquellos entornos en los que es difícil que el agente, en el proceso de exploración, llegue a recibir alguna recompensa, o que la recompensa esté muy retrasada en el tiempo.

Para tales entornos, se pueden añadir técnicas adicionales, como las nombradas en las secciones siguientes, para facilitar la resolución de tales entornos.

Búsqueda en Árbol («Tree search»)

Para problemas con gran cantidad de combinaciones de estados posibles, y en los que el agente es difícil que reciba recompensas simplemente explorando autónomamente, pueden combinarse los sistemas de Aprendizaje por Refuerzo con sistemas de Búsqueda en Árbol [21]. Estos sistemas de Búsqueda en Árbol de soluciones, en entornos con gran cantidad de estados diferentes, es muy complicado que puedan encontrar soluciones viables en tiempos razonables. Sin embargo, apoyados por sistemas de Aprendizaje por Refuerzo, pueden encontrar tales soluciones.

Aprendizaje supervisado

Si uno de los problemas de los entornos a abordar es que el sistema no llega a recompensas explorando autónomamente, se puede, al menos en una fase inicial de entrenamiento, utilizar el Aprendizaje Supervisado, donde el sistema empezaría el entrenamiento revisando soluciones ya aportadas, de forma que la exploración le sea más fácil. De nuevo, esta técnica es utilizada en [21].

Optimización automática de hiperparámetros

Uno de los objetivos del sistema es que pudiese utilizarse para la búsqueda de los valores de los hiperparámetros de forma manual, mediante la investigación y la experimentación. Sin embargo, pueden añadirse al sistema sistemas automáticos de optimización de hiperparámetros, que hiciesen esta búsqueda más sistemática y eliminando el sesgo humano.

Para ello, se sopesa la inclusión de distintos métodos de búsqueda de hiperparámetros:

En cuadrícula: En el que los valores de los hiperparámetros son escogidos equidistantemente en forma de cuadrícula.

Aleatoria: Aunque la búsqueda en cuadrícula es sistemática, hay veces en las que puede no llegar a ninguna de las soluciones óptimas, por lo que se propone que la búsqueda aleatoria [5] pueda dar mejores resultados.

Bayesiana: Como alternativa a la búsqueda aleatoria, también puede implementarse la búsqueda bayesiana [22], en la que la historia de pasados experimentos es tenida en cuenta para nuevos valores de hiperparámetros a experimentar.

Acciones continuas

Hasta este momento, el proyecto se ha centrado en entornos de problema en los que las acciones, pueden ser más o menos numerosas, pero son discretas. Pese a esta limitación, el sistema puede utilizarse en gran cantidad de entornos, o incluso aplicarse en entornos que presentan acciones continuas pero que pueden discretizarse de alguna manera: por ejemplo, si el entorno espera una acción numérica, puede transformarse, en su lugar, en las acciones discretas «aumentar» y «disminuir» el valor de la acción.

Sin embargo, sería interesante adaptar el sistema a esta clase de acciones continuas, pues podría facilitar la resolución de ciertos entornos de problema.

Ejecución distribuida

Por último, la ejecución de los experimentos es iniciada manualmente en el sistema que se quiere que lo ejecute. Por la cantidad de procesado necesario y el tiempo que toman los experimentos en ejecutarse, es interesante que la plataforma pueda decidir el sistema en el que ejecutar los experimentos, dependiendo de la carga de cada uno de los servidores disponibles; o que, incluso, pudiese paralelizarse la ejecución de un experimento en concreto.

Apéndice A

Ejemplos

El presente apéndice presenta ejemplos de la ejecución de la plataforma para distintos entornos, y el análisis de su resultado.

CartPole

Esta sección incluye ejemplos de entrenamiento utilizando el entorno *CartPole*. Dicho entorno es un problema clásico de Aprendizaje por Refuerzo, en el que un poste está unido a un carro que se mueve sin fricción por un raíl (figura A.1). El poste comienza vertical y el objetivo del agente es mantener el poste sin que caiga hacia los lados el mayor tiempo posible, únicamente moviendo el carro a izquierda y derecha.

Por lo tanto, el entorno sólo tiene dos acciones posibles: moverse a izquierda o derecha. La observación (que será directamente igual al estado del sistema) contiene cuatro componentes: la posición del carro, su velocidad, el ángulo del poste y su velocidad angular.

La recompensa recibida será 1 cada paso en el que el poste esté vertical hasta que sobrepase una inclinación. Los episodios tienen un límite de 200 pasos, luego la recompensa máxima será 200.

DQN

Como primer paso, se utilizó el algoritmo DQN para entrenar al agente en el control de CartPole. Las siguientes imágenes corresponden al análisis de algunos de los indicadores del experimento una vez concluido este.

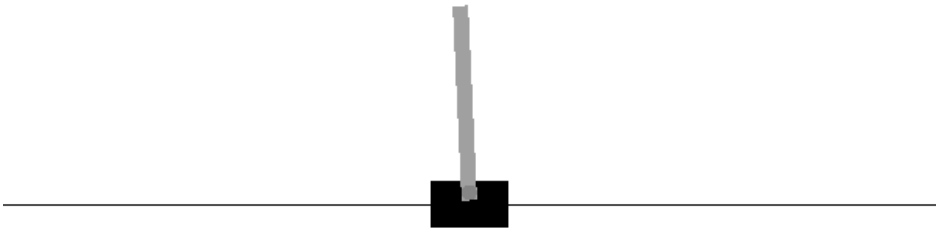
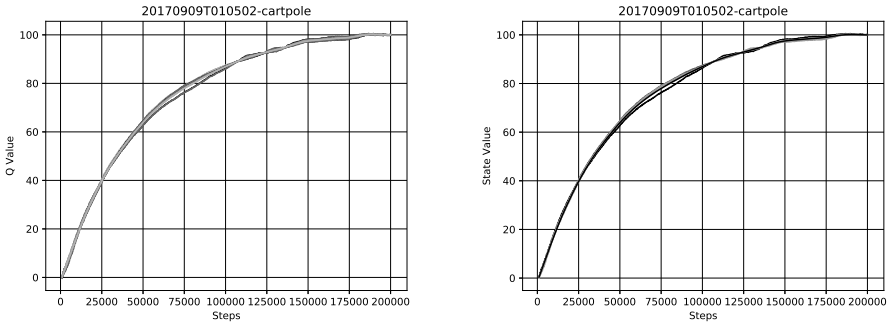


Figura A.1: CartPole

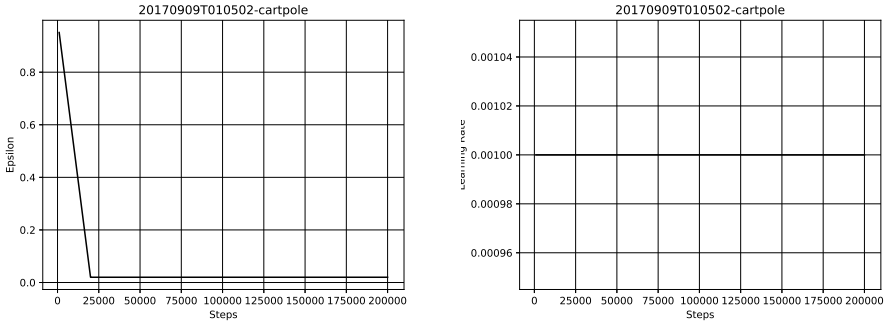


(a) Valores Q (b) Valores de estado

Figura A.2: **CartPole: DQN: Valores Q y de estado:** Se puede observar como estos valores Q y los valores de estado estimados por la Red Neuronal van creciendo y convergiendo hacia un valor estable, lo que usualmente indica que el agente está estimando los valores correctamente y que su política es correcta, es decir, que el agente está aprendiendo.

```
{
"analysis_step_fraction": 0.001,
"batch_size": 32,
"checkpoint_reward_episodes": 100, "checkpoint_seed": 0,
"checkpoint_step_fraction": 0.05,
"concurrent_experience_suppliers": 16,
"discount_factor": 0.99,
"entropy_loss_factor": 0.01,
"epsilon_decay_rate": null, "epsilon_step_final_fraction": 0.1,
"final_epsilon": 0.02,
"gradient_clipnorm": 10,
"gradient_decay": 0.99,
"initial_epsilon": 1,
"label": null,
"learning_rate": 0.001,
"learning_rate_decay": null, "learning_rate_step_final_fraction": null,
"load_network_model": false, "load_network_weights": true,
"load_path": null,
"load_replay_memory": true,
"loss_log_offset": 1e-10,
"max_episode_steps": null,
"max_steps": 200000,
"name": "cartpole",
"render": false,
"replay_memory_beta_step_final_fraction": 1.0, "replay_memory_final_beta": 1.0,
"replay_memory_importance_sampling": true, "replay_memory_initial_beta": 0.4,
"replay_memory_initial_size": 1000, "replay_memory_max_episode_steps": null,
"replay_memory_max_size": 50000, "replay_memory_priority_exponent": 0.6,
"replay_memory_priority_offset": 1e-06,
"results_path": null,
"save_base_path": "../results", "save_networks": true,
"save_replay_memory": true,
"state_group_size": null,
"target_update_frequency": 500,
"test_episodes": 10,
"training_frequency": 1,
"value_loss_factor": 1.0
}
```


Cuadro A.1: **CartPole: DQN: Fichero de hiperparámetros:** Hiperparámetros utilizados durante el entrenamiento.



(a) Epsilon (ϵ) (b) Tasa de aprendizaje (α)

Figura A.3: **CartPole: DQN: Hiperparámetros variables:** ϵ, indicador de la exploración del agente, decrece linealmente, pero se ha seleccionado una tasa de aprendizaje constante.

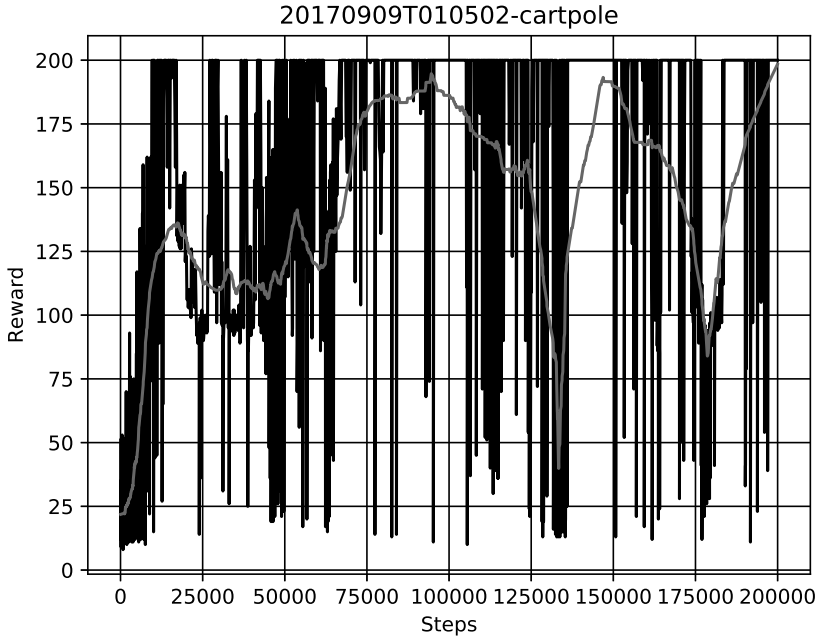
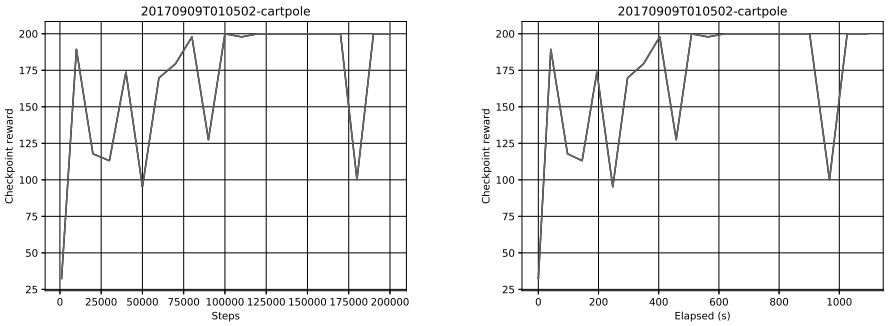


Figura A.4: **CartPole: DQN: Recompensa por episodio:** Como indicación de que el entrenamiento ha sido efectivo se puede observar que, alrededor del paso 10000, el agente había conseguido alcanzar la puntuación máxima de 200.



(a) Recompensa media de referencia (b) Recompensa media de referencia por tiempo

Figura A.5: **CartPole: DQN: Recompensa media de referencia:** La recompensa media de referencia es una mejor medida del éxito del entrenamiento, ya que no tiene en cuenta la exploración al realizar su cálculo.

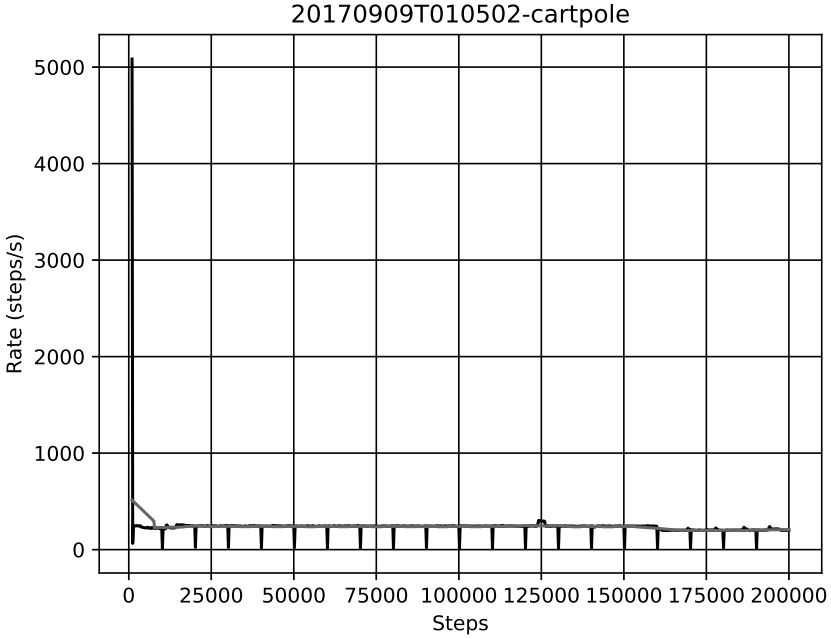
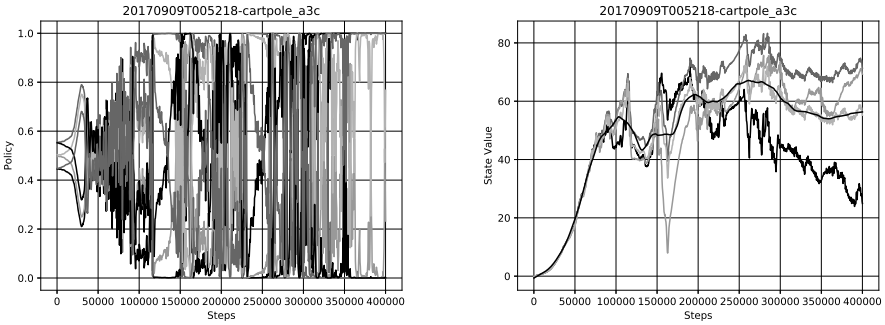


Figura A.6: **CartPole: DQN: Velocidad:** Expresa la velocidad a la que se ejecuta el algoritmo, medido en pasos del entorno por segundo.

A3C

En la presente sección, podemos observar algunos de los indicadores al ejecutar el entrenamiento, esta vez usando el algoritmo A3C para el entorno CartPole.



(a) Políticas (b) Valores de estado

Figura A.7: **CartPole: A3C: Política y valores de estado:** Valores de salida de la Red Neuronal en el algoritmo A3C.

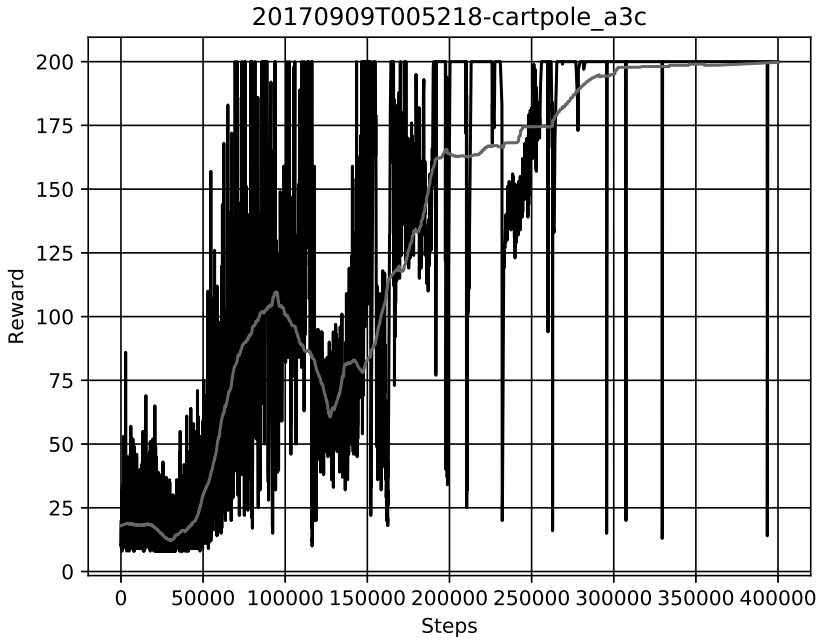
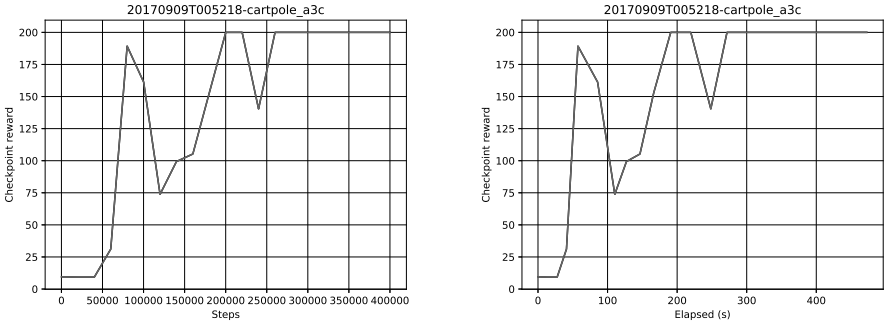


Figura A.8: **CartPole: A3C: Recompensa por episodio:** Puede observarse que el entrenamiento ha sido efectivo pero, en comparación con DQN, el agente necesita explorar mucho más el entorno (sobre 70000 pasos), para alcanzar la puntuación máxima de 200.



(a) Recompensa media de referencia (b) Recompensa media de referencia por tiempo

Figura A.9: **CartPole: A3C: Recompensa media de referencia:** Pese a que el algoritmo A3C tarda muchos más pasos en llegar a la puntuación máxima, requiriendo más exploración del entorno, podemos ver que es mucho más eficiente en tiempo, puesto que llega a esa recompensa máxima en la mitad de tiempo que DQN.

```
{
"analysis_step_fraction": 0.001,
"batch_size": 128,
"checkpoint_reward_episodes": 100, "checkpoint_seed": 0,
"checkpoint_step_fraction": 0.05,
"concurrent_experience_suppliers": 8,
"discount_factor": 0.99,
"entropy_loss_factor": 0.01,
"epsilon_decay_rate": null, "epsilon_step_final_fraction": 0.1,
"final_epsilon": 0.02,
"gradient_clipnorm": 10,
"gradient_decay": 0.99,
"initial_epsilon": 1,
"label": null,
"learning_rate": 0.005,
"learning_rate_decay": null, "learning_rate_step_final_fraction": null,
"load_network_model": false, "load_network_weights": true,
"load_path": null,
"load_replay_memory": true,
"loss_log_offset": 1e-10,
"max_episode_steps": null,
"max_steps": 400000,
"name": "cartpole_a3c",
"render": false,
"replay_memory_beta_step_final_fraction": 1.0, "replay_memory_final_beta": 1.0,
"replay_memory_importance_sampling": true, "replay_memory_initial_beta": 0.4,
"replay_memory_initial_size": 1000, "replay_memory_max_episode_steps": null,
"replay_memory_max_size": 50000, "replay_memory_priority_exponent": 0.6,
"replay_memory_priority_offset": 1e-06,
"results_path": null,
"save_base_path": "../results",
"save_networks": true,
"save_replay_memory": true,
"state_group_size": null,
"target_update_frequency": 500,
"test_episodes": 10,
"training_frequency": 1,
"value_loss_factor": 1.0
}
```


Cuadro A.2: CartPole: A3C: Fichero de hiperparámetros:

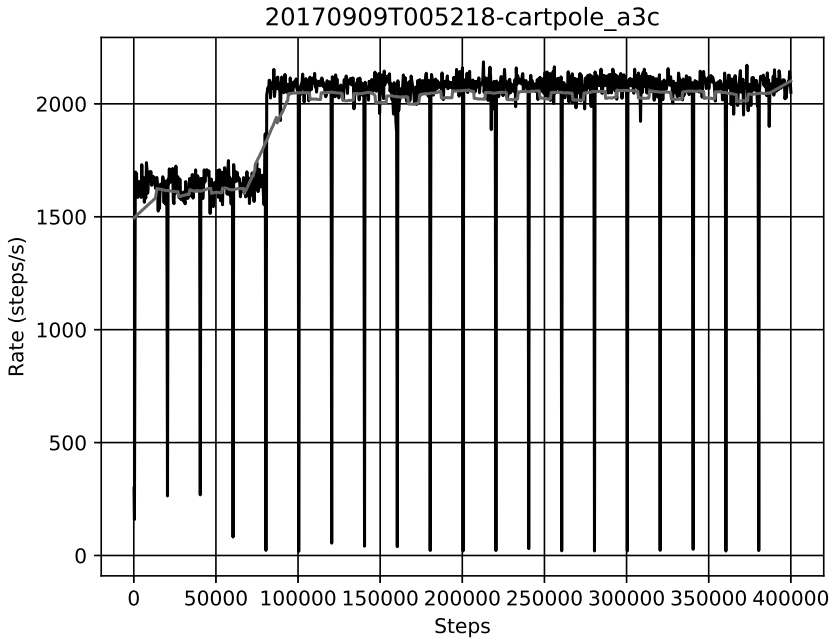


Figura A.10: CartPole: A3C: Velocidad

Pong

Esta sección incluye ejemplos de entrenamiento utilizando el entorno *Pong*, otro de los entornos que se han utilizado últimamente en la prueba de algoritmos de Aprendizaje por Refuerzo. En este caso, por ser uno de los entornos más simples en los que el agente recibe la imagen por pantalla del juego y debe controlarlo autónomamente.

Este entorno consta del Pong (figura A.11), el clásico juego de Atari, en el que el jugador maneja una barra vertical, frente a la máquina, que maneja una segunda barra. El objetivo es que una pelota sobrepase la barra contraria, haciéndola rebotar sobre la nuestra.

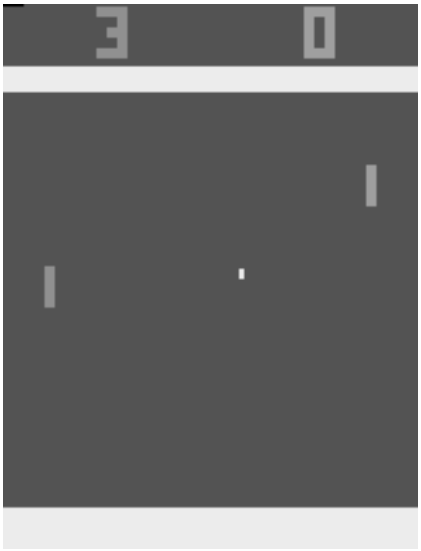
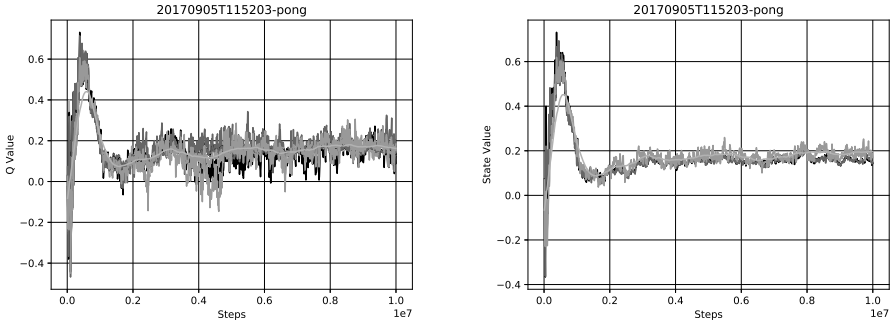


Figura A.11: Pong

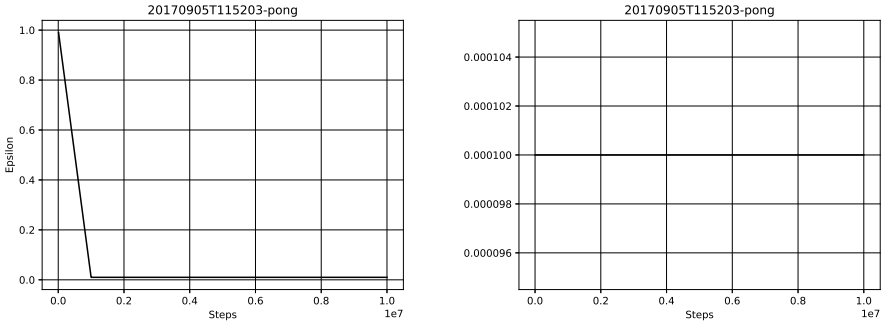
El entorno cuenta con tres acciones posibles: subir, bajar y mantenerse quieto. La observación del entorno será la imagen de la pantalla, codificada en tres canales conteniendo los píxeles RGB. Finalmente, la recompensa recibida será 1 cuando la pelota sobrepase la barra rival y -1 cuando la pelota sobrepase la barra controlada por el agente. Un episodio termina cuando un jugador sobrepasa los 21 puntos, con lo que la recompensa total de un episodio variará entre -21 y 21.

DQN



(a) Valores Q (b) Valores de estado

Figura A.12: **Pong: DQN: Valores Q y de estado:** Observamos, de nuevo, la estabilización de los valores, lo que suele indicar un aprendizaje efectivo.



(a) Epsilon (ϵ) (b) Tasa de aprendizaje (α)

Figura A.13: Pong: DQN: Hiperparámetros variables

```
{
"analysis_step_fraction": 0.001,
"batch_size": 32,
"checkpoint_reward_episodes": 10, "checkpoint_seed": 0,
"checkpoint_step_fraction": 0.05,
"concurrent_experience_suppliers": 8,
"discount_factor": 0.99,
"entropy_loss_factor": 0.01,
"epsilon_decay_rate": null, "epsilon_step_final_fraction": 0.1,
"final_epsilon": 0.01,
"gradient_clipnorm": 10,
"gradient_decay": 0.99,
"initial_epsilon": 1,
"label": null,
"learning_rate": 0.0001,
"learning_rate_decay": 0.0, "learning_rate_step_final_fraction": null,
"load_network_model": false, "load_network_weights": true,
"load_path": null,
"load_replay_memory": false,
"loss_log_offset": 1e-10,
"max_episode_steps": null,
"max_steps": 10000000,
"name": "pong",
"render": false,
"replay_memory_beta_step_final_fraction": 1.0, "replay_memory_final_beta": 1.0,
"replay_memory_importance_sampling": true, "replay_memory_initial_beta": 0.4,
"replay_memory_initial_size": null, "replay_memory_max_episode_steps": null,
"replay_memory_max_size": 10000, "replay_memory_priority_exponent": 0.6,
"replay_memory_priority_offset": 1e-06,
"results_path": null,
"save_base_path": "../results", "save_networks": true,
"save_replay_memory": false,
"state_group_size": 4,
"target_update_frequency": 1000,
"test_episodes": 10,
"training_frequency": 4,
"value_loss_factor": 1.0
}
```


Cuadro A.3: Pong: DQN: Fichero de hiperparámetros:

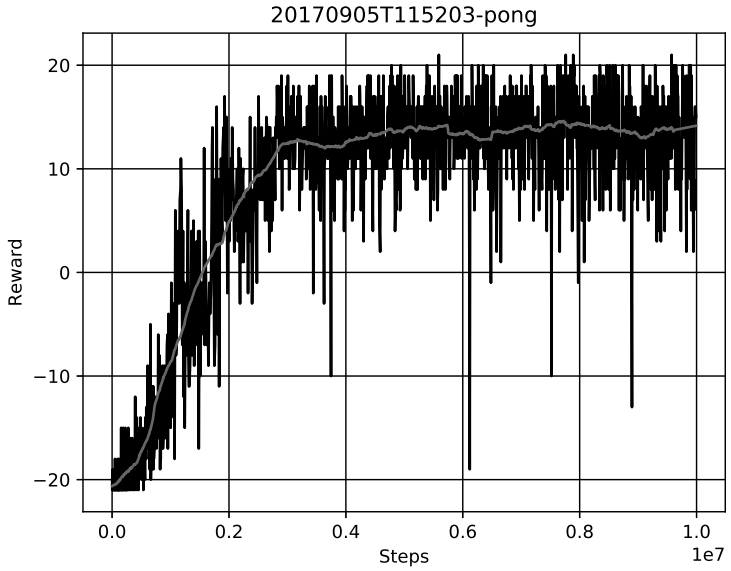


Figura A.14: Pong: DQN: Recompensa por episodio

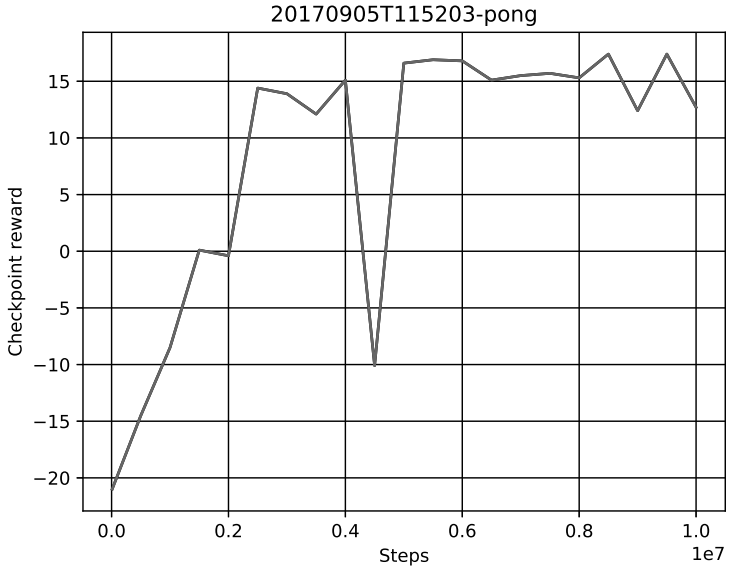


Figura A.15: **Pong: DQN: Recompensa media de referencia:** Se puede ver como el sistema ha aprendido a jugar con bastante efectividad puesto que, en promedio, gana las partidas con una puntuación de alrededor de 15 puntos.

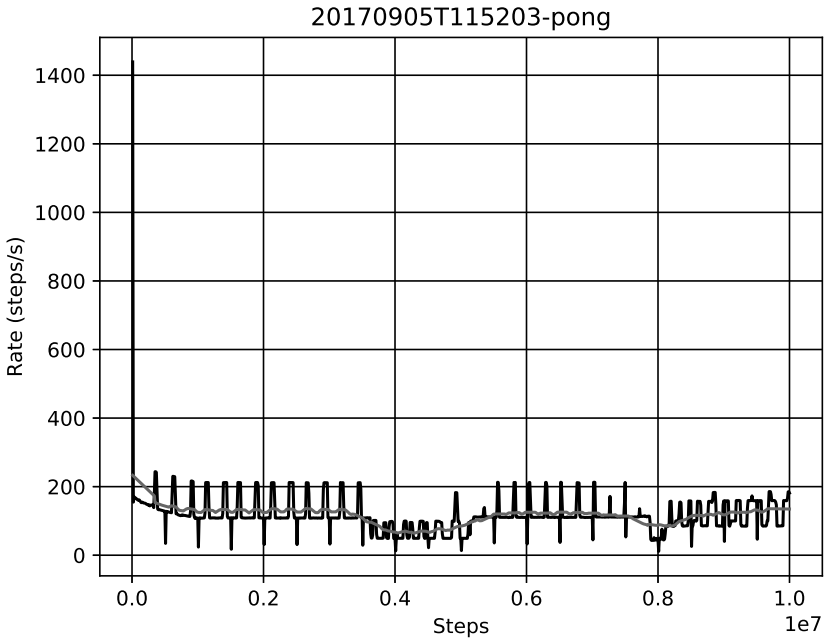
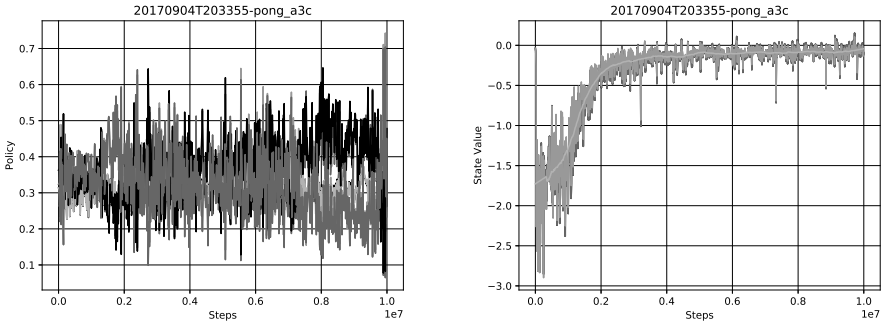


Figura A.16: Pong: DQN: Velocidad

A3C



(a) Políticas (b) Valores de estado

Figura A.17: Pong: A3C: Política y valores de estado

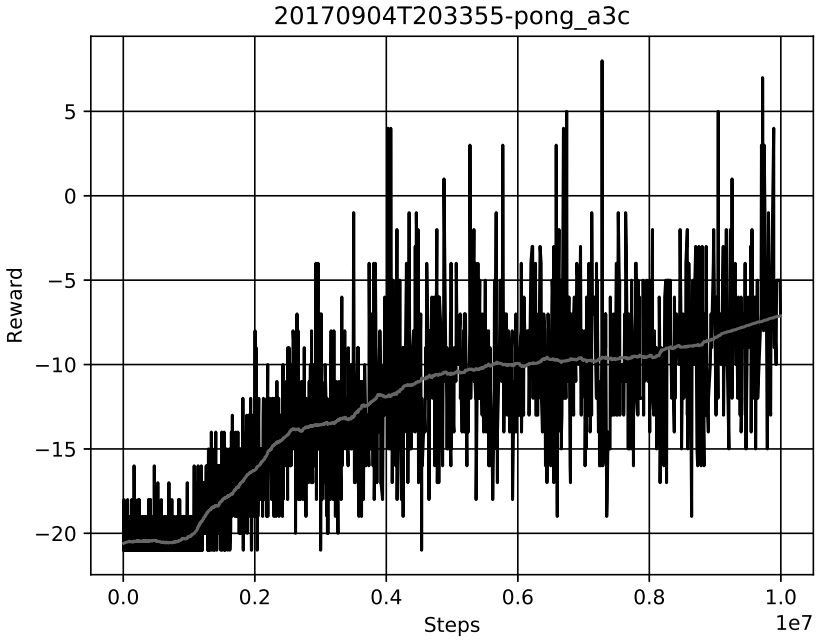


Figura A.18: **Pong: A3C: Recompensa por episodio:** Se puede observar que las recompensas, en este caso, son mucho más bajas que las calculadas en los puntos de referencia (como veremos en la siguiente gráfica). Ello es debido a que el algoritmo A3C incluye la exploración durante todo el entrenamiento como la entropía en la pérdida, y esa exploración hace que se seleccionen, con una probabilidad alta, acciones que no son las que se creen óptimas. Lo que reduce la recompensa total obtenida en este proceso de entrenamiento. Puesto que la recompensa media de referencia se calcula sin exploración, es mucho más alta y refleja mejor el estado de entrenamiento del agente.

```
{
"analysis_step_fraction": 0.001,
"batch_size": 32,
"checkpoint_reward_episodes": 10, "checkpoint_seed": 0,
"checkpoint_step_fraction": 0.05,
"concurrent_experience_suppliers": 8,
"discount_factor": 0.99,
"entropy_loss_factor": 0.01,
"epsilon_decay_rate": null, "epsilon_step_final_fraction": 0.1,
"final_epsilon": 0.01,
"gradient_clipnorm": 10,
"gradient_decay": 0.99,
"initial_epsilon": 1,
"label": null,
"learning_rate": 0.0001,
"learning_rate_decay": 0.0, "learning_rate_step_final_fraction": null,
"load_network_model": false, "load_network_weights": true,
"load_path": null,
"load_replay_memory": false,
"loss_log_offset": 1e-10,
"max_episode_steps": null,
"max_steps": 10000000,
"name": "pong_a3c",
"render": false,
"replay_memory_beta_step_final_fraction": 1.0, "replay_memory_final_beta": 1.0,
"replay_memory_importance_sampling": true, "replay_memory_initial_beta": 0.4,
"replay_memory_initial_size": null, "replay_memory_max_episode_steps": null,
"replay_memory_max_size": 10000, "replay_memory_priority_exponent": 0.6,
"replay_memory_priority_offset": 1e-06,
"results_path": null,
"save_base_path": "../results", "save_networks": true,
"save_replay_memory": false,
"state_group_size": 4,
"target_update_frequency": 1000,
"test_episodes": 10,
"training_frequency": 4,
"value_loss_factor": 1.0
}
```


Cuadro A.4: Pong: A3C: Fichero de hiperparámetros:

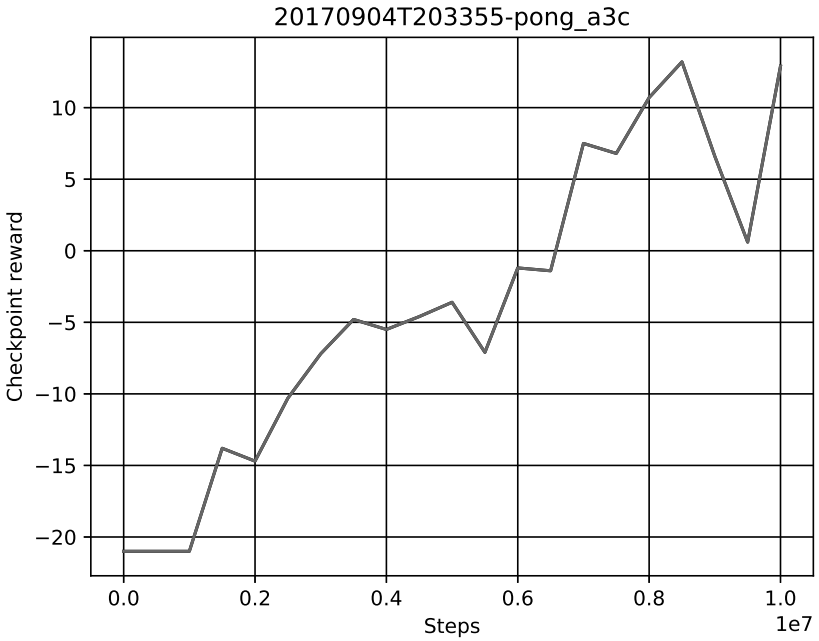


Figura A.19: **Pong: A3C: Recompensa media de referencia:** Se observa que esta recompensa es sobre 5 puntos peor que la obtenida por DQN. No se debe inferir por esto que el algoritmo A3C sea inferior al DQN, sino que, probablemente, no se han optimizado suficientemente los hiperparámetros en este experimento y no se está encontrando una solución óptima.

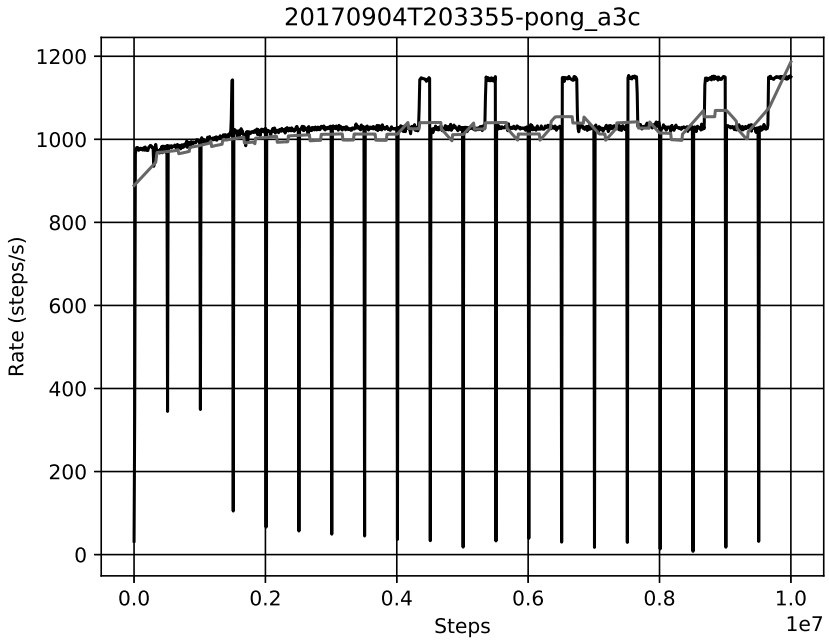


Figura A.20: Pong: A3C: Velocidad

Referencias

[1] Martín Abadi y col. "Tensorflow: Large-scale machine learning on heterogeneous distributed systems". En: *arXiv preprint arXiv:1603.04467* (2016). URL: https://arxiv.org/abs/1603.04467.

[2] Paul Barrett y col. "matplotlib–A Portable Python Plotting Package". En: *Astronomical Data Analysis Software and Systems XIV*. Vol. 347. 2005, pág. 91. URL: http://adsabs.harvard.edu/full/2005ASPC..347...91B.

[3] Charles Beattie y col. "Deepmind lab". En: *arXiv preprint arXiv:1612.03801* (2016). URL: https://arxiv.org/abs/1612.03801.

[4] Kent Beck y col. *Manifesto for Agile Software Development*. 2001. URL: http://agilemanifesto.org/ (visitado 01-09-2017).

[5] James Bergstra y Yoshua Bengio. "Random search for hyper-parameter optimization". En: *Journal of Machine Learning Research* 13.Feb (2012), págs. 281-305. URL: http://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf.

[6] James Bergstra y col. "Theano: A CPU and GPU math compiler in Python". En: *Proc. 9th Python in Science Conf.* 2010, págs. 1-7. URL: http://www-etud.iro.umontreal.ca/~wardefar/publications/theano_scipy2010.pdf.

[7] Mariusz Bojarski y col. "End to end learning for self-driving cars". En: *arXiv preprint arXiv:1604.07316* (2016). URL: https://arxiv.org/pdf/1604.07316.

[8] Greg Brockman y col. "OpenAI gym". En: *arXiv preprint arXiv:1606.01540* (2016). URL: https://arxiv.org/abs/1606.01540.

[9] Google DeepMind. *DeepMind Lab*. 2017. URL: https://github.com/deepmind/lab (visitado 01-09-2017).

[10] Python Software Foundation. *Python*. 2017. URL: https://www.python.org/ (visitado 01-09-2017).

[11] Standard C++ Foundation. *Standard C++*. 2017. URL: https://isocpp.org/ (visitado 01-09-2017).

[12] Mance E Harmon y Stephanie S Harmon. "Reinforcement learning: A tutorial". En: *WL/AAFC, WPAFB Ohio* 45433 (1996). URL: http://kiosk.nada.kth.se/kurser/kth/2D1432/2006/rltutorial.pdf.

[13] Google Inc. *TensorFlow*. 2017. URL: https://www.tensorflow.org/ (visitado 01-09-2017).

[14] Ian Lenz, Honglak Lee y Ashutosh Saxena. "Deep learning for detecting robotic grasps". En: *The International Journal of Robotics Research* 34.4-5 (2015), págs. 705-724. URL: https://arxiv.org/pdf/1301.3592.

[15] Volodymyr Mnih y col. "Asynchronous methods for deep reinforcement learning". En: *International Conference on Machine Learning*. 2016, págs. 1928-1937. URL: http://proceedings.mlr.press/v48/mniha16.pdf.

[16] Volodymyr Mnih y col. "Human-level control through deep reinforcement learning". En: *Nature* 518.7540 (2015), págs. 529-533. URL: http://dx.doi.org/10.1038/nature14236.

[17] OpenAI. *OpenAI Gym*. 2017. URL: https://gym.openai.com/ (visitado 01-09-2017).

[18] Tim Salimans y col. "Evolution strategies as a scalable alternative to reinforcement learning". En: *arXiv preprint arXiv:1703.03864* (2017). URL: https://arxiv.org/abs/1703.03864.

[19] Tom Schaul y col. "Prioritized experience replay". En: *arXiv preprint arXiv:1511.05952* (2015). URL: https://arxiv.org/abs/1511.05952.

[20] Ken Schwaber y Jeff Sutherland. "The scrum guide". En: *Scrum Alliance* 21 (2011). URL: http://www.scrumguides.org/docs/scrumguide/v2016/2016-Scrum-Guide-US.pdf.

[21] David Silver y col. "Mastering the game of Go with deep neural networks and tree search". En: *Nature* 529.7587 (2016), págs. 484-489.

[22] Jasper Snoek, Hugo Larochelle y Ryan P Adams. "Practical bayesian optimization of machine learning algorithms". En: *Advances in neural information processing systems*. 2012, págs. 2951-2959. URL: http://papers.nips.cc/paper/4522-practical-bayesian-optimization-of-machine-learning-algorithms.pdf.

[23] Richard S Sutton y Andrew G Barto. *Reinforcement learning: An introduction*. Vol. 1. 1. MIT press Cambridge, 1998.

[24] Matplotlib Development Team. *matplotlib*. 2017. URL: `http://matplotlib.org/` (visitado 01-09-2017).

[25] R Core Team. *The R Project for Statistical Computing*. 2017. URL: `https://www.r-project.org/` (visitado 01-09-2017).

[26] Theano Development Team. *Theano*. 2017. URL: `http://www.deeplearning.net/software/theano/` (visitado 01-09-2017).

[27] Hado Van Hasselt, Arthur Guez y David Silver. "Deep Reinforcement Learning with Double Q-Learning." En: *AAAI*. 2016, págs. 2094-2100. URL: `http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/download/12389/11847`.

[28] Various. *Keras: The Python Deep Learning library*. 2017. URL: `https://keras.io/` (visitado 01-09-2017).

[29] Yuhuai Wu y col. "Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation". En: *arXiv preprint arXiv:1708.05144* (2017). URL: `https://arxiv.org/abs/1708.05144`.