CAMPUS TERRA

APRENDIZAJE FEDERADO EN MÚLTIPLES ROBOTS CON ARQUITECTURAS HETEROGÉNEAS Y DESTILACIÓN DEL CONOCIMIENTO

TRABAJO FIN DE GRADO
PARA LA TITULACIÓN DEL
GRADO EN ROBÓTICA

AUTOR:

MARTÍN ROMERO ROMERO

TUTORES:

DR. FRANCISCO JAVIER GARCÍA POLO
DPTO ELECTRÓNICA E COMPUTACIÓN - USC

DR. ROBERTO IGLESIAS RODRÍGUEZ
DPTO ELECTRÓNICA E COMPUTACIÓN - USC

CONVOCATORIA:
JULIO DE 2023

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

ESCOLA POLITÉCNICA SUPERIOR DE ENXEÑARÍA

UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

TRABAJO FIN DE GRADO

Aprendizaje federado en múltiples robots con arquitectura heterogénea y destilación del conocimiento.

AUTOR:

MARTÍN ROMERO ROMERO

TUTORES:

Dr. FRANCISCO JAVIER GARCÍA POLO
Dpto. de Electrónica e Computación – USC

Dr. ROBERTO IGLESIAS RODRÍGUEZ
Dpto. de Electrónica e Computación - USC

Este trabajo se presenta para cumplir con los requirimentos normativos exigidos para lograr el Grado en Robótica en la Escola Politécnica Superior de Enxeñaría.

Fecha de la Convocatoria: Julio de 2023

ESCOLA POLITÉCNICA SUPERIOR
DE ENXEÑARÍA

Resumo

APRENDIZAXE FEDERADA EN MÚLTIPLES ROBOTS CON ARQUITECTURAS HETEROXÉNEAS E DESTILACIÓN DO COÑECEMENTO

por
Martín Romero Romero

Palabras Chave: Aprendizaxe federada, aprendizaxe por reforzo, destilación do coñecemento, comité, privacidade.

A aprendizaxe federada é un novidoso paradigma do aprendizaxe automático que permite o adestramento de modelos entre varios dispositivos, de maneira distribuída, ao mesmo tempo que se garante a privacidade dos datos. Ao largo deste traballo exploraranse mecanismos de aprendizaxe federado e por reforzo para un sistema multi-robot. O obxectivo é que todos os robots aprendan a realizar unha mesma tarefa colaborando entre eles e compartindo o seu coñecemento local para así xerar un modelo global (máis robusto) do problema. O desafío reside en que cada axente posuirá unha arquitectura ou modelo de aprendizaxe local distinto ao dos seus compañeiros. Exploraranse estratexias de aprendizaxe baseadas en comités e destilación do coñecemento, que permitan facer fronte a esta heteroxeneidade. Por último, estas estratexias serán avaliadas en tempo real nun Turtlebot.

Resumen

APRENDIZAJE FEDERADO CON MÚLTIPLES ROBOTS CON ARQUITECTURAS HETEROGÉNEAS Y DESTILACIÓN DEL CONOCIMIENTO

por
Martín Romero Romero

Palabras Clave: Aprendizaje federado, aprendizaje por refuerzo, destilación del conocimiento, comité, privacidad.

El aprendizaje federado es un novedoso paradigma de aprendizaje automático que permite el entrenamiento de modelos entre varios dispositivos, de manera distribuida, al mismo tiempo que se garantiza la privacidad de los datos. A lo largo de este trabajo se explorarán mecanismos de aprendizaje federado y por refuerzo para un sistema multi-robot. El objetivo es que todos los robots aprendan a realizar una misma tarea colaborando entre ellos y compartiendo su conocimiento local para así generar un modelo global (más robusto) del problema. El desafío reside en que cada agente poseerá una arquitectura o modelo de aprendizaje local distinto al de sus compañeros. Se explorarán estrategias de aprendizaje basadas en comités y destilación del conocimiento, que permitan hacer frente a esta heterogeneidad. Por último, estas estrategias serán evaluadas en tiempo real en un Turtlebot.

Abstract

FEDERATED LEARNING IN MULTIPLE ROBOTS WITH HETEROGENEOUS ARCHITECTURES AND KNOWLEDGE DISTILLATION

by
Martín Romero Romero

Keywords: Federated learning, reinforcement learning, policy distillation, committee, privacy.

Federated learning is a novel machine learning paradigm that enables model training across multiple devices, in a distributed fashion, while ensuring data privacy. Throughout this work, federated and reinforcement learning mechanisms for a multi-robot system will be explored. The goal is for all the robots to learn to perform the same task by collaborating with each other and sharing their local knowledge in order to generate a (more robust) global model of the problem. The challenge is that each agent will have a local learning architecture or model that is different from its peers. Learning strategies based on committees and knowledge distillation will be explored, which allow to face this heterogeneity. Finally, these strategies will be evaluated in real time in a Turtlebot.

Índice General

Índice de figuras

Índice de Tablas

Lista de Abreviaciones

ε	Épsilon

1 INTRODUCCIÓN

La robótica es un campo que se enfoca en el diseño, construcción, programación y operación de robots. Un robot es una máquina capaz de funcionar de forma autónoma, siguiendo comandos preprogramados o tomar decisiones basadas en información de sensores [1]. En la robótica se combina el conocimiento de la mecánica, la electrónica, la computación y la informática para crear sistemas que pueden realizar una gran variedad de tareas. La robótica se ha expandido significativamente en los últimos años y se está convirtiendo en una herramienta clave en la mayoría de los ámbitos de la sociedad.

Actualmente, existen muchas tareas en las que el uso de los robots es fundamental y cada vez más común, como el uso en la industria y la vida cotidiana. Los robots domésticos son un gran ejemplo. Estos robots nos ayudan en tareas del hogar como limpiar (Figura 2 (b)), cocinar o cuidar a niños y ancianos [2] (Figura 1 (b)). También hay robots de entretenimiento que se han popularizado en los últimos años, como es el caso de [3] . En la industria, los robots se utilizan principalmente para tareas de automatización, tareas repetitivas y peligrosas, consiguiendo mejorar la eficiencia y seguridad en los procesos industriales. Los robots industriales pueden realizar tareas como soldar, pintar, montar y embalar [4] (Figura 2 (a)). La robótica es también fundamental en el ámbito de la logística [5] (Figura 1 (a)). Además, la robótica está revolucionando la medicina, permitiendo realizar cirugías menos invasivas y más precisas [6].

(a) (b)

Figura 1: (a) Robot logístico [7]. (b) Robot social [8]

(a) (b)

Figura 2: (a) Robot soldador [9]. (b) Robot aspirador [10]

En este contexto de evolución constante, sería deseable que las empresas pudieran compartir los comportamientos de sus robots de forma que se crearan comportamientos unificados globales mejorados que después fuesen a su vez compartidos entre ellas. Por ejemplo, una empresa especializada en crear comportamientos de navegación dentro de edificios podría compartirlos con los de otra empresa especializada en comportamientos de navegación en terrenos abruptos. La unificación de ambos comportamientos daría lugar a un comportamiento de navegación más robusto y global del que se podrían beneficiar ambas compañías. Además, sería deseable mantener la privacidad de los datos de las empresas involucradas en este proceso de compartición. Por lo tanto, este proceso se podría llevar a cabo mediante lo que se conoce en

aprendizaje automático como aprendizaje federado [11].

El aprendizaje federado tiene como objetivo entrenar algoritmos de aprendizaje automático (e.g. redes de neuronas profundas) en diversos dispositivos para construir modelos generales y robustos sin el intercambio de muestras de datos explícitos. Esto implica entrenar un modelo local con un conjunto de datos locales y compartir los parámetros (como los pesos de una red neuronal profunda) para generar un modelo global. De esta forma, el entrenamiento del algoritmo puede lograr sin comprometer la privacidad y seguridad de los datos almacenados en cada dispositivo, los objetivos fijados Todas estas definiciones se pueden extrapolar fácilmente a la robótica, donde las distintas plataformas robóticas serían nodos locales, sobre los que se entrenan modelos/comportamientos/políticas locales, a partir de los cuales se pueden construir modelos/comportamientos/políticas globales asegurando la privacidad de la información obtenida.

1.1 Motivación

Por lo tanto, a pesar de que el aprendizaje federado ofrece un marco de trabajo para la compartición y creación de comportamientos globales muy beneficiosos para la robótica, también introduce algunas complicaciones que se pretenden estudiar en este trabajo: la heterogeneidad en la representación de los comportamientos, y, derivado parcialmente de lo anterior, la generación de comportamientos globales que pudieran ser demasiado complejos y costosos.

Por un lado, parece lógico que las empresas representen sus comportamientos utilizando sus propias estrategias. Por ejemplo, el comportamiento de navegación en edificios podría ser aprendido mediante la utilización del algoritmo Q-Learning[1] dentro del aprendizaje por refuerzo, y el de navegación en terrenos abruptos mediante la utilización de una DQN[2]. El aprendizaje federado se centra en su mayor parte en la creación de modelos globales a partir de modelos homogéneos y poca atención ha recibido la creación de modelos globales a partir de modelos heterogéneos. En este trabajo, se investiga el uso de conjuntos de modelos (*ensembles*[3]) como una forma de combinar modelos heterogéneos en un modelo global.

Por otro lado, si el número de modelos compartidos en el *ensemble* es muy elevado, la compartición de este *ensemble* entre todas las empresas involucradas puede ser muy costoso computacionalmente, tanto a la hora de descargarlo como a la hora de utilizarlo. Sería deseable por tanto, "destilar" el conocimiento de este *ensemble* en un modelo más pequeño y manejable que idealmente se comportará de la misma manera que el *ensemble* global. Por lo tanto, en este trabajo se estudiarán y emplearán técnicas de destilación para aligerar estos *ensembles* de modelos.

[1] Q-Learning es un algoritmo de aprendizaje por refuerzo que utiliza una tabla de valores de acción para determinar la mejor acción a tomar y un estado determinado.
[2] DQN (Deep Q-Network) es un algoritmo de aprendizaje por refuerzo que utiliza redes neuronales profundas para estimar la función Q en lugar de una tabla de valores de acción.
[3] Nos referiremos como *ensembles*, al conjunto de modelos que utilizaremos para combinar su conocimiento en un modelo global.

1.2 Objetivos

Por todo ello, el objetivo general de este trabajo es destilar el conocimiento de un conjunto o *ensemble* de modelos/comportamientos representados de manera heterogénea. Para demostrar estas ideas y su funcionalidad, en este trabajo nos centraremos en un dominio de navegación sencillo. Este objetivo general se puede dividir en los siguientes objetivos particulares:

1. **Estudio del estado del arte en aprendizaje federado.** Este estudio se llevará a cabo prestando especial importancia a:
   i. La literatura existente que trata de generar modelos globales a partir de modelos heterogéneos.
   ii. Las técnicas de destilación existentes.
2. **Selección del dominio de experimentación.** En este caso, el dominio será un entorno Open AI Gym construido a propósito para este proyecto.
3. **Estudio y diseño de la estrategia de generación de comportamientos globales a partir de comportamientos representados de manera heterogénea.** El aprendizaje de comportamientos en este proyecto se llevará a cabo mediante aprendizaje por refuerzo y las políticas aprendidas se representarán de manera heterogénea de manera tabular (e.g., empleando el algoritmo Q-Learning), o utilizando redes de neuronas (e.g., utilizando DQN).
   A partir de estas políticas heterogéneas se creará un *ensemble* que permitirá la creación de este comportamiento global perseguido en aprendizaje federado.
4. **Estudio y diseño de la estrategia de destilación.** Se estudiarán y diseñarán las estrategias de destilación que permitan obtener un modelo reducido con un rendimiento similar al del modelo global, en este caso, al del *ensemble*.
5. **Experimentación en el dominio propuesto.** Se aplicarán los modelos obtenidos sobre el dominio de simulación creado.

1.3 Estructura del trabajo

El siguiente documento se estructura en los siguientes capítulos:

1. **Introducción.** Que se corresponde con este capítulo del trabajo.
2. **Estado del arte**. En el que se ofrecerá el marco teórico en el que se encuentra este trabajo.
3. **Del ensemble de políticas heterogéneas a una política destilada**. Donde se describirá la estrategia desarrollada para el aprendizaje del conjunto de comportamientos/políticas heterogéneas.
4. **Dominio de navegación Open AI**. Donde se describe el entorno creado para llevar a cabo la experimentación del trabajo.
5. **Resultados**. Se presentarán y explicarán los resultados obtenidos en la experimentación del trabajo.
6. **Conclusiones**. Se desarrollarán las conclusiones obtenidas a partir de los resultados obtenidos en el capítulo anterior.

2. ESTADO DEL ARTE

El aprendizaje federado, el aprendizaje por refuerzo y la destilación del conocimiento son tres técnicas de aprendizaje automático que se han utilizado de forma independiente en una variedad de aplicaciones. Para resolver problemas de aprendizaje en entornos distribuidos que también requieren privacidad, la combinación de estas técnicas crea nuevas y emocionantes posibilidades. En este capítulo, se presenta el estado del arte de estas técnicas en el contexto de la construcción de modelos de aprendizaje automático colaborativos, capaces de aprovechar la información local en diferentes dispositivos (en nuestro caso, robots), para mejorar la eficiencia del aprendizaje y mantener la privacidad de los datos. Por todo ello, en primer lugar, se presentará brevemente el aprendizaje federado (Sección 2.1.), posteriormente se ofrecerá una panorámica del aprendizaje por refuerzo (Sección 2.2.), y luego se describirá la combinación de ambos (Sección 2.3.). Finalmente, se introducirán algunas nociones básicas a la destilación del conocimiento (Sección 2.4.) y la librería Open AI Gym que se empleará durante la fase de experimentación de este trabajo (Sección 2.5.). Con ello se construye el marco teórico en el que se encuentra este trabajo.

2.1. Aprendizaje federado

En esta sección, se pretende introducir brevemente el aprendizaje federado. Para ello, en primer lugar se ofrecerá una breve introducción al mismo (Sección 2.1.1.), y después se describirá en qué consiste el aprendizaje federado con modelos heterogéneos (Sección 2.1.2.).

2.1.1. Breve introducción al aprendizaje federado

El aprendizaje federado es una técnica de aprendizaje automático que se utiliza para entrenar modelos sin necesidad de datos centralizados [11]. En su lugar, se utilizan dispositivos distribuidos por toda la red para realizar el proceso de entrenamiento. Esta técnica ha ganado popularidad en los últimos años debido a su capacidad para permitir el aprendizaje de modelos en entornos muy comunes, como dispositivos móviles y sensores de Internet de las cosas (IoT). Por ejemplo, la imagen a continuación muestra un ejemplo de una aplicación de aprendizaje combinado en el cuidado de la salud donde la protección de datos es primordial. En la Figura 3 se muestra un esquema de uso de aprendizaje federado. En este se ve como los datos locales no son compartidos, si no que se comparten los modelos de conocimiento ya entrenados, manteniendo la privacidad de los datos.

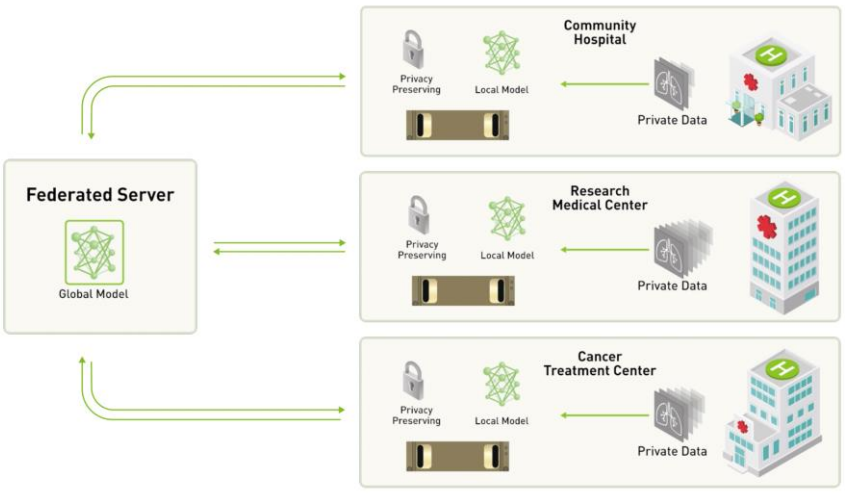


Figura 3: Aprendizaje Federado [11]

El aprendizaje federado funciona mediante la distribución de la carga de trabajo entre múltiples dispositivos,

lo que permite que los modelos se entrenen en datos locales en lugar de enviarlos a un servidor centralizado. Esto significa que los datos permanecen en los dispositivos y no se transmiten a una ubicación central. Esto es particularmente útil en situaciones en las que los datos son sensibles o privados, como en el caso de información médica o financiera.

El proceso de entrenamiento federado consta de diversas fases. En primer lugar, se seleccionan los dispositivos que se utilizarán para el entrenamiento. A continuación, se entrena un modelo inicial utilizando un subconjunto de los datos disponibles en los dispositivos seleccionados. Por último, los modelos se envían a los dispositivos para su optimización, y se utilizan técnicas de agregación para obtener los modelos actualizados y producir un modelo final.

El problema de la heterogeneidad de los dispositivos, la comunicación deficiente y la privacidad en los datos son solo alguno de los diversos desafíos a los que se enfrenta el aprendizaje federado. Para hacer frente a estos desafíos, se han desarrollado varias técnicas, por ejemplo, el uso de técnicas de aprendizaje transferido para adaptar los modelos a diferentes dispositivos, algoritmos de compresión para reducir la cantidad de datos transmitidos, y técnicas de encriptación para garantizar la privacidad de los datos.

A pesar de estas dificultades, el aprendizaje federado se ha ganado popularidad en una variedad de aplicaciones, incluido el procesamiento de imágenes, el reconocimiento de voz y la detección de anomalías en el IoT. Este método es especialmente útil para aplicaciones en las que se requiere entrenar modelos precisos con una gran cantidad de datos, pero en las que la centralización de los datos no es posible debido a la privacidad o la capacidad de comunicación limitada.

En resumen, el aprendizaje federado es una técnica de aprendizaje automático altamente descentralizada que permite el entrenamiento de modelos en dispositivos distribuidos sin la necesidad de datos centralizados. Aunque plantea retos únicos, ha demostrado ser útil en diversos contextos, y se espera que su uso aumente en el futuro.

2.1.2. Aprendizaje federado con modelos heterogéneos

El aprendizaje federado asume en la mayor parte de los casos que los clientes emplean los mismos modelos de aprendizaje [12]. Por ejemplo, si consideramos redes de neuronas, entonces todos los clientes emplearían exactamente la misma arquitectura de red (misma cantidad de capas ocultas y misma cantidad de neuronas por capa). Sin embargo, a la heterogeneidad de dispositivos también se le podría sumar la de los modelos: los clientes podrían utilizar modelos locales diferentes entre sí (por ejemplo, con distintas capas ocultas y/o neuronas). Esto dificulta el proceso de generación del modelo global: si para los modelos homogéneos se emplean técnicas de promedio de pesos para la generación del modelo global [13], la combinación de modelos heterogéneos en un modelo global ha sido menos estudiada [14]. En este último caso, lo habitual es construir un *ensemble* de modelos [15], técnica que también se explorará en este trabajo, pero en el contexto del aprendizaje por refuerzo.

Por otro lado, el aprendizaje federado permite el entrenamiento de modelos en dispositivos distribuidos, incluso cuando estos dispositivos constan de diferentes arquitecturas hardware y software. Esta técnica es especialmente útil en entornos descentralizados, como el Internet de las cosas (IoT), donde la variabilidad de los dispositivos y la escaseza de recursos computacionales y de almacenamiento pueden hacer que el aprendizaje centralizado no sea factible. También es especialmente útil en entornos robóticos, cuando consideramos que cada uno de esos dispositivos pueden ser plataformas robóticas heterogéneas.

En este contexto, es necesario que los modelos puedan ajustarse a los diferentes dispositivos de forma óptima. En otras palabras, los modelos deben ser capaces de aprender de manera eficaz a pesar de las diferencias en las características de los dispositivos.

La forma para abordar la heterogeneidad de los dispositivos varía según las características específicas de cada dispositivo y del conjunto de datos. Por lo tanto, el aprendizaje federado con modelos y/o dispositivos heterogéneos se divide en las siguientes fases:

1. **Selección de dispositivos**: se seleccionan los dispositivos que se utilizarán para el entrenamiento (teniendo en cuenta la propia heterogeneidad entre los dispositivos y los datos).

2. **Preprocesamiento de los datos**: se realiza un preprocesamiento de los datos para adaptarlos a las características de cada dispositivo. En este preprocesamiento se puede incluir la normalización de los datos, reducción de la dimensionalidad, eliminación de características irrelevantes y selección de las características más relevantes.
3. **Entrenamiento de los modelos**: se entrena un modelo inicial utilizando un subconjunto de datos disponibles en los dispositivos. Es fundamental que el modelo sea lo suficientemente general como para adaptarse a las diferentes características de los dispositivos.
4. **Adaptación de los modelos a los dispositivos**: los modelos se adaptan a las características de los dispositivos utilizando técnicas de aprendizaje transferido. Estas técnicas permiten que los modelos aprendan de los datos de un dispositivo y lo apliquen a otro dispositivo.
5. **Agregación de los modelos**: los modelos obtenidos se integran utilizando técnicas de agregación, como la media ponderada o la mediana, o incluso *ensembles* como los que se exploran en este trabajo. Esto permite obtener un modelo final que refleje las características de los dispositivos utilizados en el entrenamiento.

El aprendizaje federado con modelos y dispositivos heterogéneos presenta varios desafíos, como la variabilidad en las características de los dispositivos y la necesidad de adaptar los modelos a los mismos. Para abordar estos desafíos, se han desarrollado diversas técnicas, como la selección de dispositivos basada en el rendimiento del modelo, la utilización de algoritmos de compresión para reducir la cantidad de datos transmitidos y la utilización de técnicas de encriptación para garantizar la privacidad de los datos.

El aprendizaje federado heterogéneo se ha utilizado en una variedad de aplicaciones, como la detección de objetos en el IoT, la detección de fraude en transacciones financieras y el reconocimiento de voz en dispositivos móviles [16]. Esta técnica se ha vuelto cada vez más popular debido a su capacidad para entrenar modelos en entornos altamente descentralizados, permitiendo una mejor adaptación a los diferentes dispositivos.

2.2. Aprendizaje por refuerzo

En esta sección se va a describir de forma breve en qué consiste el aprendizaje por refuerzo. Para ello, en la sección 2.2.1. se introducirán los conceptos principales del aprendizaje por refuerzo, y en las secciones 2.2.2. y 2.2.3., se explicarán dos de los algoritmos más usados en aprendizaje por refuerzo, como son Q-Learning y DQN.

2.2.1. Breve introducción al aprendizaje por refuerzo

El aprendizaje por refuerzo es una técnica de aprendizaje automático cuyo concepto se basa en una recompensa y un castigo para guiar el comportamiento de un agente o un sistema por entorno dinámico [17]. Se utiliza en una variedad de aplicaciones, como la robótica, juegos, optimización y control de procesos, y se ha demostrado que es una manera efectiva de resolver problemas complejos en los que no se dispone de un conjunto de datos etiquetados.

El aprendizaje por refuerzo se basa en la idea de que un agente o un sistema aprende a través de la interacción con un entorno. El agente toma una acción en el entorno y recibe una recompensa o un castigo en función del resultado de la acción. El objetivo del agente es maximizar la recompensa total que recibe a lo largo del tiempo.

Figura 4: Esquematización del aprendizaje por refuerzo [18]

El proceso de aprendizaje por refuerzo implica tres componentes principales: el agente, el entorno y la función de recompensa como se puede observar en la Figura 4. El agente es el encargado de tomar decisiones en el entorno, y su objetivo es maximizar la recompensa total que recibe. El entorno es el mundo en el que el agente opera, y puede ser cualquier entorno, desde un juego hasta un proceso de fabricación. La función de recompensa es una función que se utiliza para evaluar el resultado de una acción del agente y proporcionar la recompensa o el castigo según proceda.

El proceso de aprendizaje por refuerzo se lleva a cabo en un ciclo continuo de observación, acción y recompensa. El agente observa el estado actual del entorno, decide qué acción tomar en función de su política de decisión y luego toma esa acción en el entorno. El entorno responde a la acción del agente y proporciona una recompensa o un castigo en función del resultado de la acción. El agente utiliza la recompensa para actualizar su política de decisión y mejorar su capacidad para maximizar la recompensa total.

El aprendizaje por refuerzo se ha utilizado en una variedad de aplicaciones, como la robótica, el control de procesos y la optimización. En la robótica, se utiliza para enseñar a los robots a realizar tareas complejas, como caminar y manipular objetos [19]. En el control de procesos, se utiliza para optimizar el rendimiento de los procesos [20], como la producción de energía y la fabricación de productos químicos. En la optimización, se utiliza para encontrar la mejor solución a un problema, como la asignación de recursos o la planificación de rutas.

El aprendizaje por refuerzo ha demostrado ser una técnica poderosa para la resolución de problemas complejos en los que no se dispone de un conjunto de datos etiquetados. Sin embargo, también presenta retos únicos en términos de estabilidad y seguridad, ya que los agentes pueden aprender comportamientos no deseados si se les proporciona la retroalimentación incorrecta.

En resumen, el aprendizaje por refuerzo es una técnica de aprendizaje automático que utiliza la recompensa y el castigo para guiar el comportamiento de un agente o un sistema en un entorno dinámico. Se utiliza en una variedad de aplicaciones y ha demostrado ser efectiva para resolver problemas complejos en los que no se dispone de un conjunto de datos.

2.2.2. Q-Learning

Q-Learning es una técnica de aprendizaje por refuerzo que se utiliza para aprender una política de decisión óptima en un entorno desconocido. El fundamento en el que se basa es la idea de que un agente puede aprender a tomar decisiones óptimas en un entorno incierto mediante el uso de una función de valor llamada Q-valor.

El Q-valor es una función que evalúa la calidad de una acción tomada en un estado determinado. Es decir, para cada par de estado-acción, se asigna un valor que indica cuánto se espera que el agente reciba de recompensa a largo plazo al tomar esa acción en ese estado. Por lo tanto, el objetivo del agente es maximizar el Q-valor total a lo largo del tiempo.

El proceso de aprendizaje de Q-Learning implica que el agente explore el entorno y actualice su función de valor Q-valor en función de la recompensa recibida. En cada iteración, el agente observa el estado actual del entorno y decide qué acción tomar en función de su política de decisión actual. Luego, toma esa acción y

recibe una recompensa. Utilizando esta recompensa, el agente actualiza su función de valor Q-valor mediante la aplicación de una regla de actualización.

La regla de actualización de Q-Learning es una ecuación que actualiza el Q-valor para un par de estado-acción en función de la recompensa recibida y el Q-valor máximo de todos los posibles pares de estado-acción en el siguiente estado. La ecuación es la siguiente:

$$Q(s,a) = Q(s,a) + \alpha\left[r + \gamma\left(\max\left(Q(s',a')\right)\right) - Q(s,a)\right]$$

Donde:

- $Q(s,a)$ es el Q-valor para el par de estado-acción actual

- s es el estado actual del entorno

- a es la acción tomada en el estado actual

- α es la tasa de aprendizaje, que determina la rapidez con la que se actualiza el Q-valor

- r es la recompensa recibida por el agente por tomar la acción a en el estado s

- γ es el factor de descuento, que determina la importancia de las recompensas futuras en comparación con las recompensas inmediatas

- $max(Q(s',a'))$ es el Q-valor máximo para todas las posibles acciones a en el siguiente estado s'

El proceso de aprendizaje continúa hasta que el agente converge a una política de decisión óptima que maximiza el Q-valor total. La política óptima es aquella que, en cada estado, toma la acción que tiene el Q-valor máximo.

Q-Learning es una técnica de aprendizaje por refuerzo ampliamente utilizada debido a su simplicidad y capacidad para aprender políticas óptimas en entornos desconocidos. En la Figura 5 se resume visualmente el funcionamiento de este algoritmo:

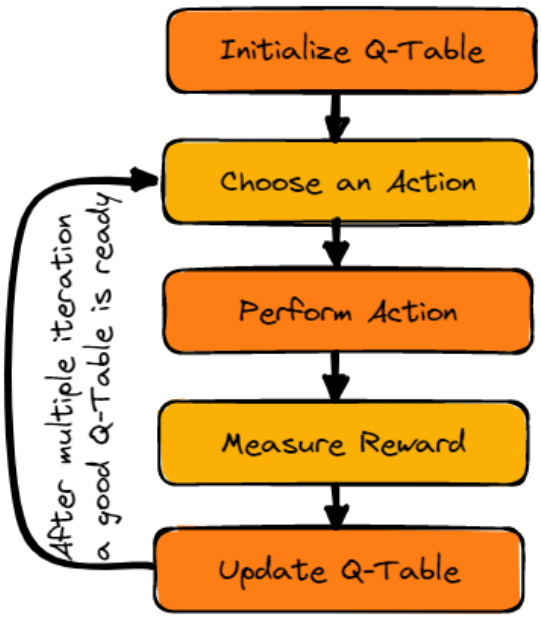


Figura 5: Explicación Q-Learning [21]

Sin embargo, también presenta desafíos, como el problema de la exploración frente a la explotación.

El problema de la exploración frente a la explotación es un desafío común en el aprendizaje por refuerzo. La dificultad de este problema recae en la dificultad de encontrar el equilibrio adecuado entre la elección de

acciones que se conocen que funcionan bien (explotación) y la búsqueda de nuevas acciones para descubrir mejores opciones (exploración).

La explotación se refiere a la elección de la acción que tiene el mayor valor Q en un estado dado, según la función de valor Q-valor. Esta estrategia puede ser efectiva en la explotación de las mejores acciones para maximizar la recompensa a corto plazo. Sin embargo, la explotación puede llevar a una política subóptima si el agente se estanca en una acción subóptima y no explora otras opciones.

Por otro lado, la exploración implica tomar acciones aleatorias o probar opciones que no se han elegido con frecuencia en el pasado, incluso si no tienen un alto valor Q. La exploración puede ayudar a descubrir mejores opciones y mejorar la política de decisión a largo plazo. Sin embargo, demasiada exploración puede disminuir el rendimiento y la eficiencia del agente.

Por lo tanto, el desafío es encontrar un equilibrio adecuado entre la exploración y la explotación para maximizar la recompensa a largo plazo y mejorar la política de decisión del agente. Hay varias estrategias de exploración frente a la explotación utilizadas en Q-Learning, como la exploración ε-greedy, la exploración por valores aleatorios, la exploración basada en incertidumbre y la exploración por búsqueda heurística. Cada estrategia tiene sus ventajas y desventajas, y la elección depende del entorno y la naturaleza del problema.

La exploración ε -greedy es una estrategia comúnmente utilizada en el aprendizaje por refuerzo para resolver el problema de la exploración frente a la explotación. Esta estrategia se utiliza con frecuencia en el contexto del algoritmo Q-Learning y se basa en la idea de que el agente debe elegir una acción de manera aleatoria en base a un porcentaje ε del tiempo y elegir la acción con el valor Q más alto en el estado actual el resto del tiempo.

La idea es que el agente elija una acción aleatoria con una probabilidad ε, que normalmente es un valor pequeño (por ejemplo, 0,1). Esta acción aleatoria se elige de manera uniforme al azar de entre todas las acciones disponibles para el agente en ese momento. El resto del tiempo, el agente elige la acción con el valor Q más alto para maximizar la recompensa a corto plazo.

La exploración ε-greedy asegura que el agente explore nuevas acciones con una probabilidad razonablemente alta (ε), mientras sigue aprovechando las acciones conocidas que producen altas recompensas en el pasado (greedy). A medida que el agente continúa aprendiendo, el valor de ε puede reducirse gradualmente para reducir la cantidad de exploración y aumentar la explotación.

Es importante destacar que la elección del valor de ε puede tener un impacto significativo en el rendimiento del agente. Si ε es demasiado bajo, el agente puede explotar demasiado las acciones conocidas y perder oportunidades de descubrir mejores acciones. Por otro lado, si ε es demasiado alto, el agente puede explorar demasiado y perder eficiencia. Por lo tanto, se recomienda ajustar el valor de ε según la naturaleza del problema y el entorno de aprendizaje.

En resumen, Q-Learning es una técnica de aprendizaje por refuerzo que utiliza una función de valor llamada Q-valor para aprender una política de decisión óptima en un entorno.

2.2.3. DQN

Deep Q-Network (DQN) es un algoritmo de aprendizaje por refuerzo profundo que ha demostrado ser efectivo en una amplia gama de tareas de aprendizaje por refuerzo, incluyendo juegos de Atari y otros juegos de arcade [22]. Fue desarrollado por el equipo de Google DeepMind en 2013 y ha sido un punto de referencia para la investigación en aprendizaje por refuerzo profundo desde entonces.

DQN se basa en la idea de utilizar una red neuronal profunda para aproximar la función Q-valor en lugar de una tabla de búsqueda, como en el caso del algoritmo Q-Learning convencional. La función Q-valor es una función que asigna un valor numérico a cada acción en un estado dado, lo que indica su capacidad para maximizar la recompensa a largo plazo. El objetivo del algoritmo es encontrar la política de decisión óptima que maximiza la recompensa total a largo plazo.

DQN utiliza una técnica llamada experiencia de repetición para mejorar la estabilidad y la eficiencia del aprendizaje. En lugar de actualizar la función Q-valor después de cada acción, el algoritmo almacena las

transiciones de experiencia en un búfer de repetición y las utiliza para actualizar la función Q-valor de forma más eficiente en lotes.

El algoritmo también utiliza una red neuronal de doble objetivo para evitar la sobreestimación de la función Q-valor y mejorar la estabilidad del aprendizaje. La red neuronal principal se actualiza en cada iteración del algoritmo, mientras que la red de destino se actualiza con menos frecuencia para evitar la propagación de errores en cascada y mejorar la estabilidad del aprendizaje.

La implementación de DQN también utiliza una técnica de exploración ε-greedy para equilibrar la exploración y la explotación en la elección de acciones. Esto permite que el agente explore nuevas acciones con una probabilidad razonablemente alta (ε), mientras que sigue aprovechando las acciones conocidas que producen altas recompensas en el pasado (greedy). A continuación, en la Figura 6 se resume el comportamiento de una DQN:

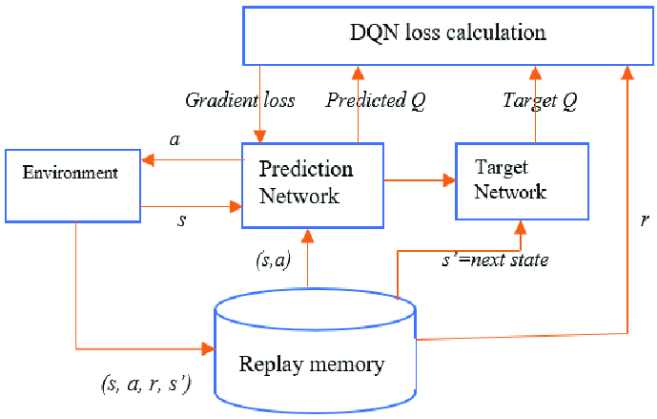


Figura 6: Explicación DQN [23]

2.3. Aprendizaje por refuerzo federado

Una vez explicado el aprendizaje federado (Sección 2.1.) y el aprendizaje por refuerzo (Sección 2.2.), en esta sección se presenta una técnica que combinan ambos conceptos, que es el aprendizaje por refuerzo federado.

El aprendizaje por refuerzo federado es un enfoque novedoso en el campo del aprendizaje automático que combina el aprendizaje por refuerzo y el aprendizaje federado para abordar problemas de colaboración y privacidad en sistemas distribuidos [24]. Esta técnica trata de entrenar modelos de aprendizaje por refuerzo utilizando múltiples agentes distribuidos, cada uno con su propia experiencia local y sin compartir directamente los datos brutos entre ellos.

Generalmente, en aprendizaje por refuerzo, los datos están centralizados en un servidor, pero hay veces en las que los datos de entrenamiento están distribuidos en diferentes ubicaciones y no pueden ser compartidos debido a problemas de privacidad, seguridad o costo computacional.

En este contexto, entra en juego el aprendizaje federado, que permite entrenar modelos en dispositivos o ubicaciones distribuidas sin la necesidad de compartir los datos reales. En lugar de enviar los datos brutos a un servidor centralizado, los agentes envían actualizaciones de los gradientes del modelo de aprendizaje por refuerzo al servidor. Estas actualizaciones son agregadas de forma federada en el servidor, utilizando técnicas como promedios ponderados o agregación segura, para obtener un nuevo modelo global. Luego, el modelo global actualizado se envía de regreso a los agentes para su siguiente iteración de entrenamiento.

El aprendizaje por refuerzo federado presenta varias ventajas, como la mejora de la eficiencia y la escalabilidad del entrenamiento de modelos de aprendizaje por refuerzo, pero también varios desafíos, como la heterogeneidad de los entornos y de los agentes de aprendizaje (que es la motivación de este trabajo).

2.4. Destilación del conocimiento

Una vez introducido el aprendizaje federado, el aprendizaje por refuerzo y el aprendizaje por refuerzo federado, esta sección pretende explicar una técnica diferente, que se ha utilizado con éxito también en el contexto del aprendizaje federado: la destilación del conocimiento. Como se comentaba en la sección 2.1.2., una posible alternativa para generar un modelo global a partir de clientes con modelos locales heterogéneos es la utilización de *ensembles.* Sin embargo, cuando existen muchos modelos locales dentro del *ensemble,* este se puede hacer muy pesado y costoso de transferir nuevamente a los clientes individuales. Un posible remedio a este problema es aligerar el *ensemble* replicando su comportamiento en un modelo más sencillo. Esto se conoce como destilación de conocimiento.

La destilación del conocimiento es una técnica de aprendizaje automático que tiene como objetivo transferir el conocimiento de un modelo a otro, generalmente para reducir la complejidad de un modelo existente sin perder su capacidad de predicción [25]. Esta técnica es particularmente útil en situaciones en las que se necesita reducir el tamaño o la complejidad de un modelo para poder ejecutarlo en dispositivos con recursos limitados, como teléfonos móviles, dispositivos IoT o robots.

Durante el proceso de destilación del conocimiento se busca que un modelo más grande y complejo que contiene datos útiles, puede ser utilizado para entrenar un modelo más pequeño. Este proceso de destilación consiste en entrenar un modelo más grande y complejo (llamado "maestro" o "profesor"), para que aprenda de un conjunto de datos dado. A continuación, se utiliza este modelo maestro para entrenar un modelo más pequeño y simple (llamado “alumno").

En el proceso de destilación el alumno no solo aprende de los datos, sino que también aprende del modelo maestro, permitiendo que el modelo alumno adquiera el conocimiento y la información valiosa del modelo maestro. De esta forma se puede reducir la complejidad del modelo alumno, sin perder su capacidad de predicción.

La destilación del conocimiento se lleva a cabo en varias fases. En primer lugar, se entrena el modelo maestro utilizando un conjunto de datos de entrenamiento. Después, se utiliza este modelo maestro para generar un conjunto de datos “suavizado”, que contiene las salidas del modelo maestro en lugar de los datos reales. Este conjunto de datos suave conforma los datos utilizados para entrenar el modelo alumno.

El conjunto de datos suave se utiliza en lugar de las etiquetas verdaderas para permitir que el modelo alumno pueda aprender de la información adicional proporcionada por el modelo maestro. El objetivo que se busca es que el modelo alumno sea capaz de aprender a imitar la distribución de las salidas del modelo maestro y no simplemente ajustarse a las etiquetas verdaderas del conjunto de datos original.

La destilación del conocimiento tiene una gran cantidad de aplicaciones, como el reconocimiento de imágenes, el procesamiento de lenguaje natural y la clasificación de datos. Se ha demostrado que la destilación del conocimiento puede mejorar significativamente la capacidad de predicción de un modelo alumno y reducir su complejidad, lo que lo hace adecuado para su uso en dispositivos con recursos limitados.

Además, la destilación del conocimiento puede utilizarse para transferir el conocimiento entre modelos de diferentes arquitecturas, permitiendo que un modelo más pequeño y simple aprenda de un modelo más grande y complejo, aunque estos dos modelos sean diferentes.

En resumen, la destilación del conocimiento es una técnica de aprendizaje automático que permite transferir el conocimiento de un modelo maestro a un modelo alumno más pequeño y simple. Consigue mejorar su capacidad de predicción y reducir su complejidad. La técnica se utiliza en una variedad de aplicaciones y puede ser utilizada para transferir el conocimiento entre modelos de diferentes arquitecturas.

2.5. Open AI Gym

Una vez vistos los principales algoritmos de aprendizaje por refuerzo que se utilizarán durante el trabajo, esta sección introduce la biblioteca Open AI Gym (Figura 7). OpenAI Gym es una biblioteca de código abierto diseñada para facilitar la investigación y el desarrollo de algoritmos de aprendizaje por refuerzo. Fue lanzada por OpenAI en 2016 y se ha convertido en una herramienta fundamental para la comunidad de inteligencia artificial. En este estado del arte, exploraremos las características clave de OpenAI Gym, sus contribuciones al campo del aprendizaje por refuerzo y los avances recientes en su desarrollo.

Figura 7: Logotitpo Open AI [26]

OpenAI Gym [26] proporciona un entorno de simulación que permite a los investigadores y desarrolladores probar y comparar algoritmos de aprendizaje por refuerzo de manera estandarizada. Algunas de las características más destacadas de OpenAI Gym incluyen:

1. Entornos: Gym ofrece una amplia variedad de entornos personalizados o predefinidos, como juegos Atari, problemas clásicos de control continuo y tareas de robótica. El diseño de estos entornos sirve para representar problemas del mundo real y proporcionar desafíos para los algoritmos de aprendizaje por refuerzo.
2. Interfaz unificada: Gym define una interfaz unificada para interactuar con los entornos, facilitando de esta forma el desarrollo y la comparación de algoritmos. En la interfaz se incluyen métodos para la toma de acciones, la observación de estados y la obtención de recompensas, permitiendo a los agentes interactuar con los entornos de manera consistente.
3. Evaluación y comparación: se proporcionan métricas y herramientas para evaluar y comparar el rendimiento de diferentes algoritmos de aprendizaje. Esto permite medir el desempeño de sus agentes y la realización de análisis comparativos para identificar enfoques más efectivos.
4. Comunidad activa: OpenAI Gym cuenta con una comunidad activa de usuarios que contribuyen con nuevos entornos, algoritmos y herramientas. Esto ha llevado a una biblioteca de extensiones y complementos que abarcan una amplia gama de aplicaciones de aprendizaje por refuerzo.

Desde su lanzamiento inicial, OpenAI Gym ha experimentado avances significativos que han ampliado su utilidad y aplicabilidad. A continuación, se presentan algunos de los avances más destacados:

1. Mujoco: OpenAI Gym ha integrado un motor de física llamado MuJoCo, que le permite simular tareas de control continuo con alta precisión. Esto ha permitido el desarrollo de algoritmos de aprendizaje por refuerzo que pueden aprender a controlar brazos robóticos y realizar tareas de manipulación complejas.
2. Baselines: OpenAI ha integrado una biblioteca llamada "Baselines" que proporciona implementaciones de referencia de algoritmos de aprendizaje por refuerzo populares. Estas implementaciones son un punto de partida para los investigadores y ayudan a establecer baselines para el rendimiento en diferentes entornos.
3. Aprendizaje profundo: Gym ha sido utilizado como plataforma para desarrollar algoritmos de aprendizaje por refuerzo profundo. El uso de redes neuronales profundas ha permitido avances significativos en el rendimiento y la generalización de los agentes de aprendizaje, para que sean capaces de aprender directamente de datos sensoriales brutos, como píxeles de imágenes, lo que ha

llevado a avances significativos en la eficiencia y generalización de los agentes. Esto permite que los agentes de aprendizaje aprendan a jugar juegos de Atari a niveles de habilidad sobrehumanos y lograr resultados destacados en otros entornos desafiantes.

4. Transferencia de aprendizaje: OpenAI Gym ha facilitado la investigación y el desarrollo de técnicas de transferencia de aprendizaje relacionadas con el aprendizaje por refuerzo. La transferencia del aprendizaje permite aprovechar el conocimiento y la experiencia adquiridos en un entorno se usen para acelerar el aprendizaje en un entorno relacionado. Esto es bastante útil cuando los agentes tienen que resolver múltiples tareas o están en entornos similares.
5. Meta-aprendizaje: Gym también se utilizó para investigar el meta-aprendizaje en relación con el aprendizaje por refuerzo. El meta-aprendizaje implica que los agentes de aprendizaje aprendan a aprender de manera más efectiva y rápida al adquirir conocimientos y estrategias generales que se puedan ser aplicadas a nuevas tareas o entornos. Mediante el uso de Gym, los investigadores han desarrollado algoritmos de meta-aprendizaje que pueden adaptarse rápidamente a nuevas tareas de aprendizaje por refuerzo sin requerir grandes cantidades de datos de entrenamiento.
6. Interacción con el mundo real: Además de los entornos de simulación, OpenAI Gym ha empezado a explorar la interacción directa de los agentes de aprendizaje con el mundo real. Esto implica el desarrollar entornos físicos reales en los cuales, los agentes puedan interactuar y aprender a través de la retroalimentación sensorial y motora real. Esta extensión de Gym ha permitido explorar y abordar desafíos adicionales, como la transferencia de aprendizaje desde simulaciones a entornos reales, lidiar con la incertidumbre y el ruido del mundo real, y el desarrollo de técnicas de aprendizaje por refuerzo que sean seguras y confiables en entornos físicos.

En resumen, OpenAI Gym ha sido una herramienta clave en el campo del aprendizaje por refuerzo, proporcionando un entorno estandarizado y una interfaz común para la investigación y el desarrollo de algoritmos. Los recientes avances en OpenAI Gym van desde el uso de aprendizaje por refuerzo profundo a la transferencia de aprendizaje, el meta-aprendizaje y la interacción con el mundo real. Estos avances han supuesto mejoras significativas en el rendimiento y capacidad de generalización de los agentes de aprendizaje, así como a la aplicación de técnicas de aprendizaje por refuerzo en una variedad de dominios y problemas del mundo real.

3. DEL *ENSEMBLE* DE POLÍTICAS HETEROGÉNEAS A UNA POLÍTICA DESTILADA

En este capítulo, se describe la estrategia desarrollada para el aprendizaje del conjunto de comportamientos/políticas heterogéneas mediante aprendizaje por refuerzo y su posterior destilación. Por lo tanto, conviene destacar que:

1. En este trabajo los modelos utilizados en los clientes no serán los típicos supervisados que se encuentran en la mayor parte de la literatura del aprendizaje federado, sino que serán políticas de comportamiento propias del aprendizaje por refuerzo.
2. Además, se asume que estas políticas de comportamiento no están representadas de manera homogénea, sino que existe disparidad a la hora de representarlas.
3. La disparidad en la representación complica la generación del modelo global. En este trabajo, se explorarán técnicas de *ensemble* y destilación para la construcción de este modelo global.

Para ello, en primer lugar, se presentará el diseño del sistema (Sección 3.1.), que servirá para representar de manera gráfica el flujo de trabajo de la propuesta. Posteriormente, se explicarán cada uno de los componentes de este esquema global: en primer lugar, el aprendizaje de los modelos individuales (Sección 3.2.); en segundo lugar, la construcción del *ensemble* (Sección 3.3.); y por último, la destilación (Sección 3.4.).

3.1. Diseño del sistema

Para el diseño del sistema, se han utilizado una serie de modelos, con políticas de comportamiento individuales, concretamente varias DQN y Q-Learning, que formarán el *ensemble*. Este *ensemble* será el encargado de generar las entradas y salidas que se utilizarán para la construcción de la política destilada. Para cada entrada, el conjunto de modelos que forman parte del *ensemble* aportarán sus predicciones. Estas predicciones serán sometidas a un preprocesado antes de incluirse en el buffer de memoria que se utilizará para destilar el modelo final. Este preprocesado consiste en una normalización de las predicciones entre 0 y 1, y su posterior suma, obteniendo de esta forma para cada estado, la predicción correspondiente.

Una vez construido un buffer de memoria lo suficientemente grande, con los estados y sus correspondientes predicciones provenientes de los modelos que componen el *ensemble*, se puede iniciar la destilación de las políticas. En la Figura 8 se resume el proceso de creación del *ensemble*, el buffer de memoria y la destilación del modelo final.

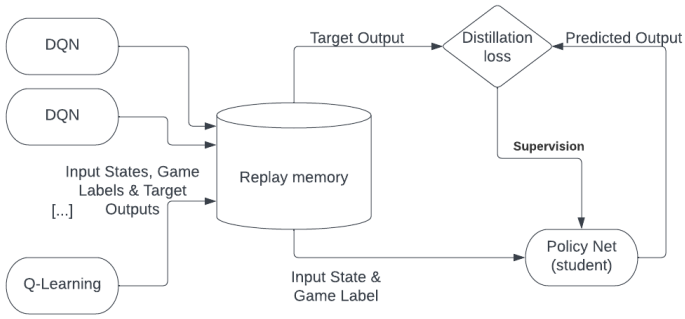


Figura 8: Diseño del sistema [25]

3.2. Construcción de las políticas de comportamiento

Esta sección se centra en la explicación de la generación de las políticas de comportamientos individuales dentro del esquema global presentado en la sección 3.1.. Para poder construir el *ensemble* de entrenamiento, primero necesitamos tener una serie de políticas de comportamiento individuales. Para generar estas políticas se han utilizado dos algoritmos de aprendizaje por refuerzo, Q-Learning y DQN. Con estos dos algoritmos (explicados respectivamente en las secciones 2.2.2. y 2.2.3. del estado del arte) se han conseguido varios comportamientos que son capaces de llevar a cabo la tarea asignada.

Para el entrenamiento del algoritmo Q-Learning, la representación de la política de comportamiento se realiza a través de una tabla Q. Esta tabla Q se obtiene tras una serie de episodios en los que el agente trata de resolver el laberinto propuesto. En cada acción de cada episodio, el agente tomará una acción y recibirá un refuerzo que podrá ser positivo o negativo. De esta forma el algoritmo obtiene la tabla Q, en la que para cada estado (situación en la que se encuentra el agente) elige la acción con mejor valoración, que será la que lleve al agente a estar más cerca de resolver el laberinto.

Durante el entrenamiento de Q-Learning, hay que tener en cuenta una serie de factores que pueden hacer que el agente sea capaz o no de resolver el laberinto de forma óptima. Primeramente, el número de episodios. Los episodios son el número de veces que el agente trata de resolver el laberinto. Si el número de episodios es muy bajo puede que no sean suficientes para que el agente aprenda a resolver el laberinto, pero si el número de episodios es muy grande, el agente puede sobreaprender las acciones que tiene que tomar para resolver el laberinto, de forma que, si se cambiase ligeramente el laberinto, el agente no sería capaz de resolverlo (el famoso "overfiting"). Por otro lado, la estrategia de exploración-explotación. En este caso se ha utilizado la estrategia ε-greedy (explicada en la sección 2.2.2.).

En el caso de la DQN, funciona de forma parecida a Q-Learning, pero usando redes neuronales como aproximador de la función de valor-acción. En este caso el agente vuelve a ejecutar una serie de episodios en los que trata de resolver el laberinto, pero con el refuerzo que recibe, en vez de actualizar una tabla, actualiza los pesos de la red neuronal. Para el entrenamiento de la DQN también hay que tener en cuenta una serie de factores. El número de episodios vuelve a ser fundamental, teniendo que encontrar una cantidad de episodios que permitan a la DQN aprender a resolver el laberinto, pero sin que se produzca el "overfiting". Otro factor decisivo es la arquitectura de red que se escoja (número de capas ocultas y número de neuronas por capa). Si el número de capas ocultas y neuronas es muy bajo es posible que la DQN no sea capaz de aprender a resolver el laberinto, pero si hay un gran número de capas y neuronas, hará que la DQN tarde más en aprender, y que el resultado obtenido no sea del todo óptimo.

3.3. Construcción del *ensemble*

Una vez construidas las políticas individuales, el siguiente paso es construir el *ensemble*. Para construir el *ensemble* necesitaremos varias políticas individuales, que pasarán por un proceso de preprocesado antes de poder añadirse al *ensemble*.

En este trabajo, el preprocesado consiste en calcular la predicción que hace cada modelo a un estado dado, normalizar entre 0 y 1 las predicciones y sumar todas las predicciones de los modelos para el mismo estado. Con esto conseguimos un conjunto de datos (*ensemble*) en el que para cada estado tenemos una predicción basada en el conocimiento de todos los modelos entrenados anteriormente. Con este *ensemble* ya formado se puede pasar al siguiente paso, que sería la destilación del modelo.

3.4. Destilación

Una vez construidas las políticas individuales y el *ensemble*, el último paso es la destilación del conocimiento. Como se describió en la sección 2.4. , la destilación del conocimiento consiste en entrenar una red neuronal (que llamaremos "estudiante") utilizando las predicciones de otra red neuronal (o en este caso a partir de varias redes neuronales, que llamaremos "profesor") para que, en vez de aprender a replicar datos, intente replicar las salidas de las redes neuronales. Este proceso hace que la red destilada pueda ser más pequeña que la red original, e incluso llegar a obtener mejores resultados que la red original.

Con los *ensembles* ya construidos destilamos una red que nos ha permitido obtener los mismos resultados que las redes originales, pero con un tamaño bastante más pequeño, poniendo de manifiesto la gran utilidad de la destilación.

Durante el proceso de destilación, la red neuronal estudiante trata de replicar el comportamiento de la red "profesor" tratando de minimizar un error. Este error, se calcula comparando las salidas de la red estudiante con las de la red profesor, y se pueden utilizar diversos métodos, en el trabajo de [25] utilizan tres formas distintas, pero en este trabajo utilizaremos solo una de ellas (la que mayor éxito ha resultado obtener). Esta ecuación de cálculo del error (L) viene determinado por la siguiente expresión:

$$L_{KL}(D^T, \theta_S) = \sum_{i=1}^{|D|} softmax\left(\frac{q_i^T}{\tau}\right) \ln \frac{softmax(\frac{q_i^T}{\tau})}{softmax(q_i^S)}$$

En esta ecuación el error se calcula usando la divergencia de Kullback-Leibler (KL) y una temperatura τ. q_i^T son las predicciones del profesor, y q_i^S son las predicciones de la destilación, a las cuales se les aplica una función softmax que suaviza las probabilidades, para así poder comparar las salidas de la red estudiante con las salidas de la red profesor.

4. DOMINIO DE NAVEGACIÓN OPEN AI

En este capítulo, se describe el entorno creado para llevar a cabo la experimentación del trabajo, en el que se ha utilizado la librería Open AI Gym, explicada en la sección 2.4 del estado del arte.

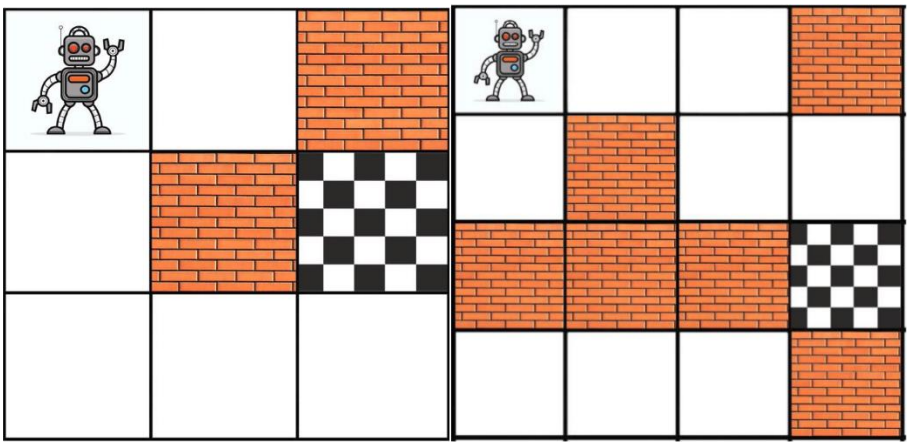
Esta librería nos permite crear nuestro propio entorno de simulación, y que adaptaremos a nuestro caso específico. En concreto, hemos creado un laberinto donde podemos especificar sus dimensiones de altura y anchura (e.g., 3x3 o 4x4). Podemos definir la posición inicial del agente (robot), la posición objetivo del agente y la posición de los obstáculos. Para la detección de los obstáculos, el agente simula un lidar que le permite detectar si en las celdas contiguas a la celda en la que se encuentra hay un obstáculo o no.

Se han definido también el espacio de estados, el espacio de acciones y una función de refuerzo. El espacio de estados se define como un vector de 5 elementos. Los tres primeros elementos son de tipo booleano, e indican si existe un obstáculo en la celda situada a la derecha, en frente o a la izquierda del robot respectivamente. El siguiente elemento es la posición relativa del objetivo con respecto al agente, es decir, se indica si la celda objetivo está delante o detrás del agente y si está a la derecha o a la izquierda del agente, pudiendo combinar estos dos parámetros (e.g., delante a la derecha). Por último, se define también como parámetro del espacio de estados la distancia de Manhattan desde el agente al objetivo.

El espacio de acciones cuenta con tres posibles opciones: avanzar hacia delante, girar 90º hacia la derecha y girar 90º hacia la izquierda.

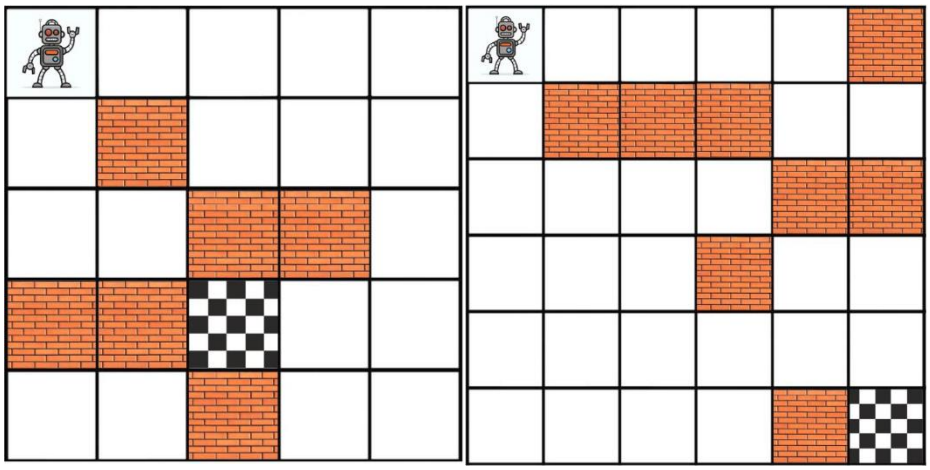
Finalmente, la función de refuerzo consiste en penalizar con un resultado de –1, las acciones que al ejecutar el agente no llega directamente al objetivo, y premiar con +10, las acciones que al ejecutarse hacen que el agente llegue al objetivo.

Con todo esto podemos aplicar distintas técnicas de aprendizaje por refuerzo que permitan al robot obtener el conocimiento necesario para resolver el circuito. Para las experimentaciones se han diseñado distintas configuraciones de laberintos. Concretamente se han diseñado 4 configuraciones distintas, una configuración de tamaño 3x3, una configuración de 4x4, una configuración de 5x5 y una configuración de 6x6. En las Figuras 9 y 10 , se representan los laberintos diseñados, donde la figura del robot se corresponde con la posición inicial, la celda de cuadros con la posición objetivo, y las casillas naranja con las casillas que contienen muros y que no se pueden acceder a ellas.



(a) (b)

Figura 9: Laberintos: (a) 3x3. (b) 4x4



(a) (b)

Figura 10: Laberintos: (a) 5x5. (b) 6x6

5. RESULTADOS

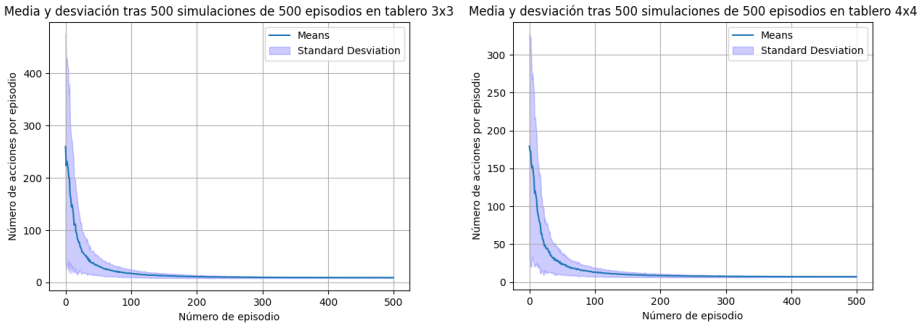
Una vez descritos todos los pasos de la propuesta diseñada en el Capítulo 3, en este capítulo se presentan los resultados de los entrenamientos para conseguir políticas individuales (Sección 5.1.), la destilación de políticas individuales (Sección 5.2.1.) y la construcción del *ensemble* y su posterior destilación (Sección 5.2.2.), que se han realizado sobre el entorno de simulación descrito en el Capítulo 4. Por último, se incluye una última sección de experimentación en robot reales (Sección 5.3.).

5.1. Entrenamiento de políticas individuales

En primer lugar vamos a presentar los resultados que hemos obtenido de los entrenamientos para lograr las políticas individuales. Para estas políticas se han utilizado dos algoritmos, Q-Learning y DQN.

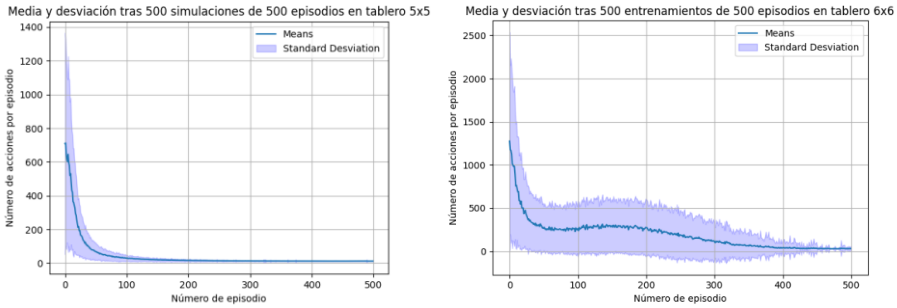
5.1.1. Experimentación con Q-Learning

Para este entrenamiento, se ha utilizado el algoritmo Q-learning (explicado en la Sección 2.2.2.). En las Figuras 11-12 se representa la media y la desviación estándar tras realizar 500 entrenamientos con Q-learning en las distintas configuraciones de laberinto presentadas en el Capítulo 4. En el eje X se representa el número de episodio de la simulación y en el eje Y el número de acciones por episodio. En las cuatro gráficas vemos un comportamiento parecido, en los primeros episodios necesita muchas acciones para lograr el objetivo, y según van pasando los episodios el número de pasos decae hasta llegar al mínimo de pasos requeridos para resolver el laberinto. Esta evolución nos demuestra que con el algoritmo Q-learning hay un claro aprendizaje.



(a) (b)

Figura 11: Evolución de entrenamiento con Q-Learning para: (a) Laberinto 3x3. (b) Laberinto 4x4



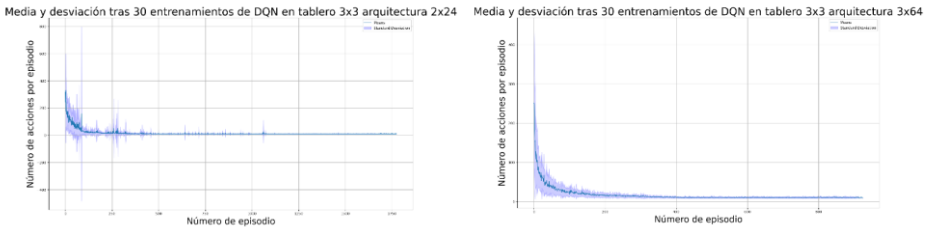
(a) (b)

Figura 12: Evolución de entrenamiento con Q-Learning para: (a) Laberinto 5x5. (b) Laberinto 6x6

5.1.2. Experimentación con DQN

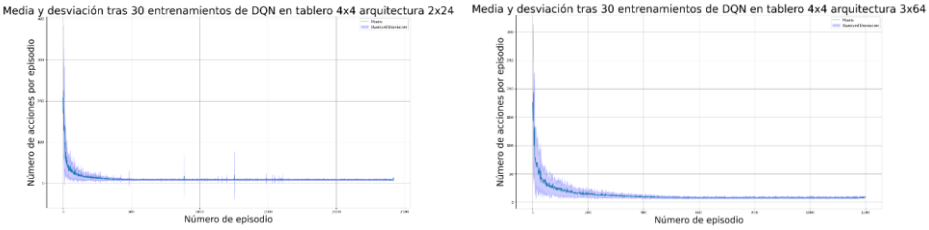
Para el entrenamiento con DQN se ha repetido el mismo procedimiento explicado en la sección anterior. Para los laberintos 3x3 y 4x4 se han usado dos arquitecturas de red distintas. La primera ha sido una red con dos capas ocultas de 24 neuronas cada capa, y en la segunda arquitectura, la red ha constado de 3 capas ocultas con 64 neuronas por capa. En este caso se han hecho 30 entrenamientos con DQN, y al representar el número de acciones por episodio vemos también que el modelo de DQN muestra un claro aprendizaje. Para el laberinto 5x5 y 6x6 se ha tenido que aumentar el número de neuronas por capa de la DQN para que consiguiese converger, concretamente para 5x5 y 6x6 se han utilizado una arquitectura de 3 capas ocultas, con 96, 192 y 96 neuronas respectivamente.

En la Figura 13-14-15 se representa la media y la desviación estándar tras realizar 30 entrenamientos con DQN en las distintas configuraciones de laberinto y con las distintas arquitecturas de red.



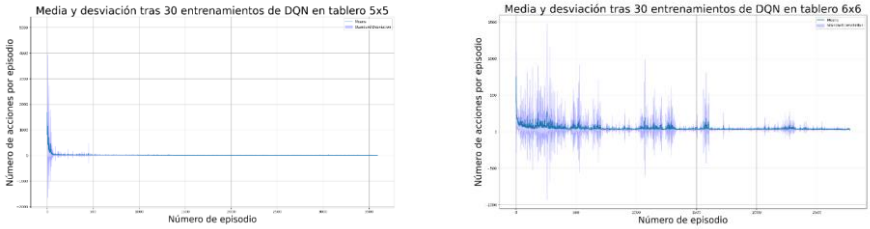
(a) (b)

Figura 13: Media y desviación en tablero 3x3: (a) arquitectura 2x24. (b) arquitectura 3x64



(a) (b)

Figura 14: Media y desviación en tablero 4x4: (a) arquitectura 2x24. (b) arquitectura 3x64



(a) (b)

Figura 15: Media y desviación en: (a) tablero 5x5. (b) tablero 6x6

5.2. Destilación

En esta sección se presentarán los resultados obtenidos tanto de la destilación de las políticas individuales, cuyos resultados se han presentado en la anterior sección, como la destilación del *ensemble*.

5.2.1. Destilación de políticas individuales

Antes de hacer una destilación del *ensemble*, se ha probado a destilar las políticas individuales para ver la capacidad de compresión que esta técnica puede aportar a los modelos. Para los modelos entrenados en los laberintos 3x3, 4x4 y 5x5, la arquitectura de la red destilada ha sido de una red con dos capas ocultas de 12 neuronas por capa (en 3x3 y 4x4 partíamos de 3 capas de 64 neuronas cada una y en 5x5 de tres capas de 96, 192, 96 neuronas). Para el modelo de laberinto 6x6 se ha utilizado una red con una arquitectura de dos capas ocultas y 24 neuronas por capa, debido a que la arquitectura utilizada para las otras destilaciones no ha sido capaz de obtener resultados satisfactorios.

Una vez destilados los comportamientos individuales, se han probado en el mismo entorno que los modelos originales para analizar los resultados de la destilación. Estos resultados se recogen en la Tabla 1, donde se presenta una tabla donde se una comparación de la puntuación de las redes originales (DQN) y las redes destiladas (Dist-KL). En particular, en esta tabla se muestra la columna *score* que describe la puntuación final obtenida (recordemos que cada movimiento que no sea llegar a la meta resta 1 punto y al llegar a la meta se suman 10 puntos) tanto para la DQN de partida como la destilada. En las redes destiladas se aporta también el porcentaje de neuronas %DQN que contiene la red destilada con respecto a la original. Así, por ejemplo, para el laberinto 3x3 obtenemos el mismo score tanto con el "profesor" como con el "estudiante", pero sin embargo el "estudiante" tiene un 12,5% de neuronas con respecto al "profesor".

Tabla 1: Tabla comparativa entre red original y destilada

Laberinto	DQN	Dist-KL	
	Score	Score	%DQN
3x3	2	2	12,5
4x4	4	4	12,5
5x5	1	1	6,25
6x6	-17	-7	12,5

Esta información que se muestra en la Tabla 1, se presenta en la Figura 16 en forma de gráfica, donde se representan diferentes bloques de figuras. En los bloques con la etiqueta *Profesor*, se muestra el porcentaje de puntuación obtenida (barra azul) y el porcentaje de tamaño de la red empleada durante el proceso de aprendizaje (barra roja) durante el aprendizaje del profesor para los diferentes tamaños de laberinto. En este caso, es la red original de partida, luego todas tienen un 100%. En los bloques con la etiqueta Dist-KL se muestra la misma información para el proceso de destilación: el porcentaje de puntuación obtenida (barra azul), pero en este caso se muestra el porcentaje de compresión de la red con respecto a la red original empleada durante la destilación (barra naranja) Podemos leer, por ejemplo, que mediante una red con un 6,25% de neuronas con respecto a la red original, obtenemos el mismo comportamiento en el laberinto 5x5.

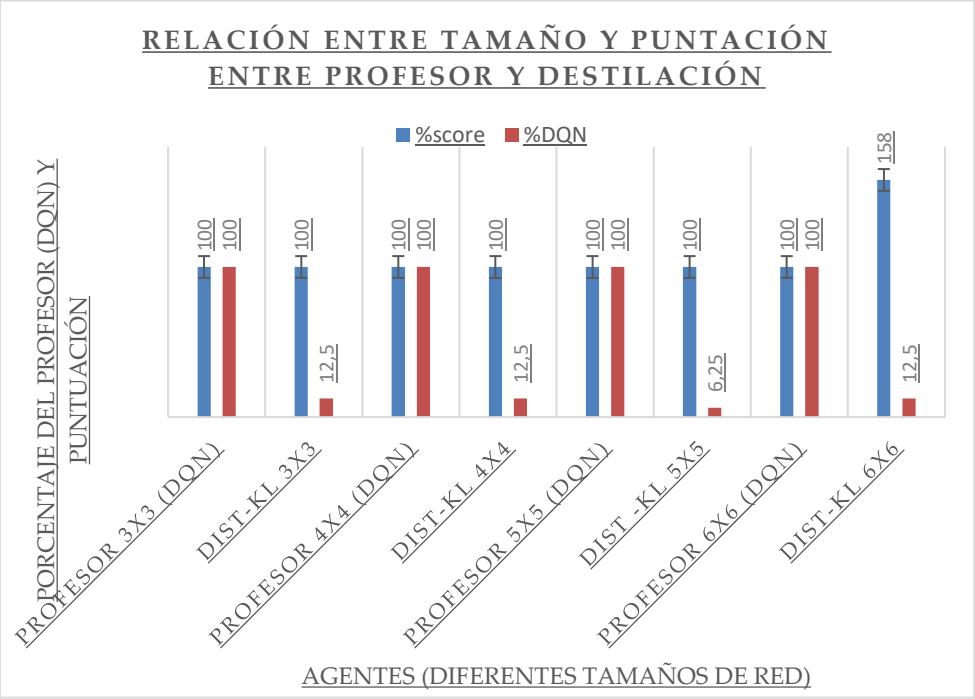


Figura 16: Gráfica de relación de tamaño y puntuación entre profesor y alumno

Tras ver los datos de las destilaciones, vemos que los modelos destilados consiguen igualar las puntuaciones de los modelos originales (en un caso incluso mejorarlo, como es el caso en el laberinto 6x6), pero con un tamaño de red mucho más bajo que el modelo original, demostrando la gran utilidad de la destilación que se mencionaba anteriormente.

5.2.2. Destilación del *ensemble*

Una vez explicado el entrenamiento de las políticas individuales y la destilación de estas políticas individuales, en esta sección se describirá la destilación del *ensemble*. En particular, se ha construido un *ensemble* para cada una de las configuraciones de laberinto propuestas, y cada uno está compuesto por dos políticas DQN (la DQN original y la destilada) y una política de Q-Learning. De esta manera, cada *ensemble* queda constituido con políticas representadas de manera heterogénea. Con estas tres políticas, generamos un dataset de información para cada laberinto (desde 3x3 hasta 6x6) que nos permitirá destilar los *ensembles* formados. El tamaño de la red destilada en cada caso es: para el laberinto 3x3, 4x4 y 5x5 una red de neuronas de 2 capas y 12 neuronas por capa, y para el laberinto 6x6 se emplea una red de neuronas de 2 capas y 24 neuronas por capa.

Una vez destilados los *ensembles* vamos a comparar los resultados de la política destilada con los resultados del *ensemble* global. De la misma manera que antes, en la Figura 17 se presenta esta comparación. La columna azul representa el porcentaje de puntuación del *ensemble*, que será la puntuación de partida para comparar la puntuación de la destilación (por eso será siempre del 100%). En la columna de color rojo se representa el porcentaje de puntación obtenida de la destilación en comparación con el *ensemble*.

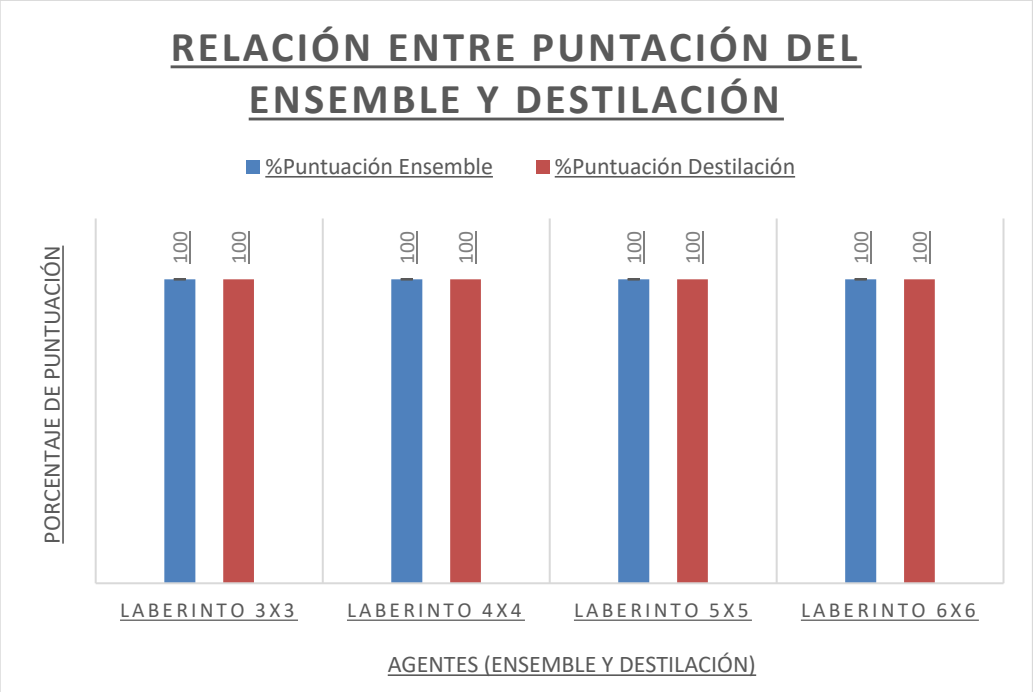


Figura 17: Relación entre puntuación del ensemble y destilación

Tras los resultados de la Figura 17, se puede ver que la destilación consigue la misma puntuación que el *ensemble*.

5.3. Experimentación en robot real

Tras todos los experimentos hechos en los apartados anteriores de este capítulo, en esta sección se presentan los resultados de entrenamiento con Q-Learning, DQN y la destilación de esta última, sobre un robot real, concretamente sobre el Turtlebot2. Para ello, el entorno de simulación que antes se ejecutaba con la librería Open AI Gym (Sección 2.5.), ahora ya no será necesario simularla ya que la representación de estados se recogerá de los sensores del Turtlebot, y las acciones serán también tomadas por este, utilizando ROS (Robot Operating System, librería con la que se controla el robot). Además, para reducir el tiempo de entrenamiento en el Turtlebot real, solo se ha utilizado el laberinto 3x3 (Figura 9a).

5.3.1. Experimentación con Q-Learning

Para la experimentación con Q-Learning, la ejecución del algoritmo se ha ejecutado íntegramente en el Turtlebot, y de la misma forma que en la Sección 5.1.1, se representa la media y la desviación estándar tras realizar 500 entrenamientos con Q-learning, donde el eje X representa el número de episodio de la simulación y el eje Y el número de acciones por episodio (Figura 18).

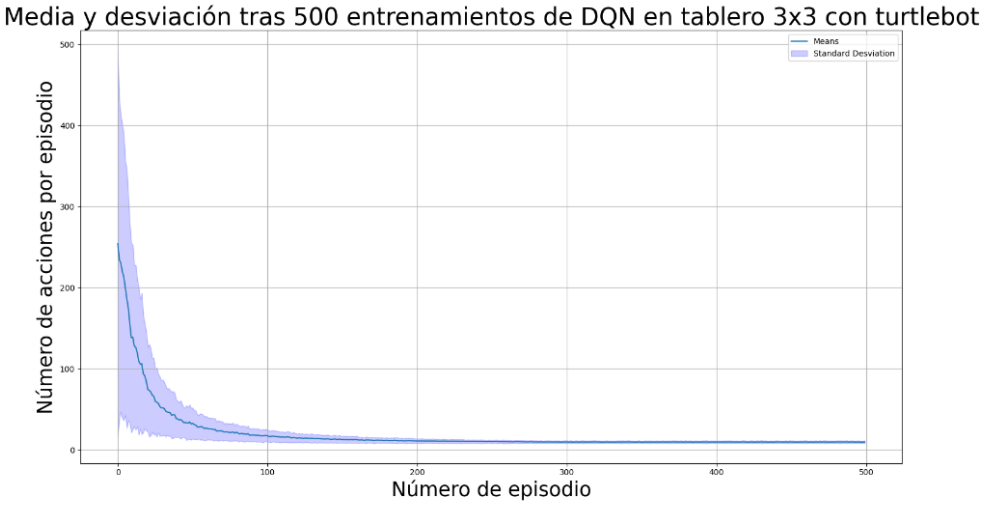


Figura 18: Evolución de entrenamiento con Q-Learning en Turtlebot2

Para ilustrar todo el entrenamiento llevado a cabo en el Turtlebot real, se han grabado una serie de vídeos a los que se puede acceder a través de este enlace: TurtleGym/Videos at master · RomeroRomeroMartin/TurtleGym · GitHub

En este enlace hay una serie de vídeos en los que se puede apreciar el aprendizaje llevado a cabo por el Turtlebot en las etapas iniciales y finales de entrenamiento, tanto con Q-Learning (Sección 5.3.1.) como con DQN (Sección 5.3.2.)

5.3.2. Experimentación con DQN

Para la experimentación con DQN, ha sido necesario adaptar un poco la ejecución debido a que el Turtlebot2 del que se dispone no soporta alguna de las librerías que se utilizaron para el entrenamiento de anterior de la DQN. Para afrontar este problema, el Turtlebot2 será el encargado de determinar el estado en el que se encuentra y ejecutar la acción. Sin embargo, las acciones que ejecutará el Turtlebot2 y los cálculos necesarios para en entrenamiento se llevarán a cabo un ordenador, que se comunicará con el Turtlebot a través de un servidor TCP.

Debido a esto, el tiempo de entrenamiento aumenta considerablemente respecto a la experimentación con la librería Open AI Gym, por lo que solo se ha hecho un entrenamiento de la DQN. Durante el entrenamiento, no se han ejecutado todos los episodios en el Turtlebot real, si no que cada ciertos episodios se testeaba de forma real como avanzaba el entrenamiento, para así poder reducir el tiempo de entrenamiento de la DQN.

Tras este entrenamiento, la DQN obtenida (con una arquitectura de dos capas oculta de 24 neuronas) también es capaz de resolver el laberinto, del mismo modo que lo hacía anteriormente.

5.3.3. Destilación de la DQN

Como ya hicimos en las simulaciones, una vez obtenido un comportamiento con DQN, procedemos a destilar la red obtenida. La red entrenada (profesor) era de una red de 2 capas ocultas de 24 neuronas, y la red destilada (estudiante) es una red de 2 capas ocultas con 12 neuronas cada una.

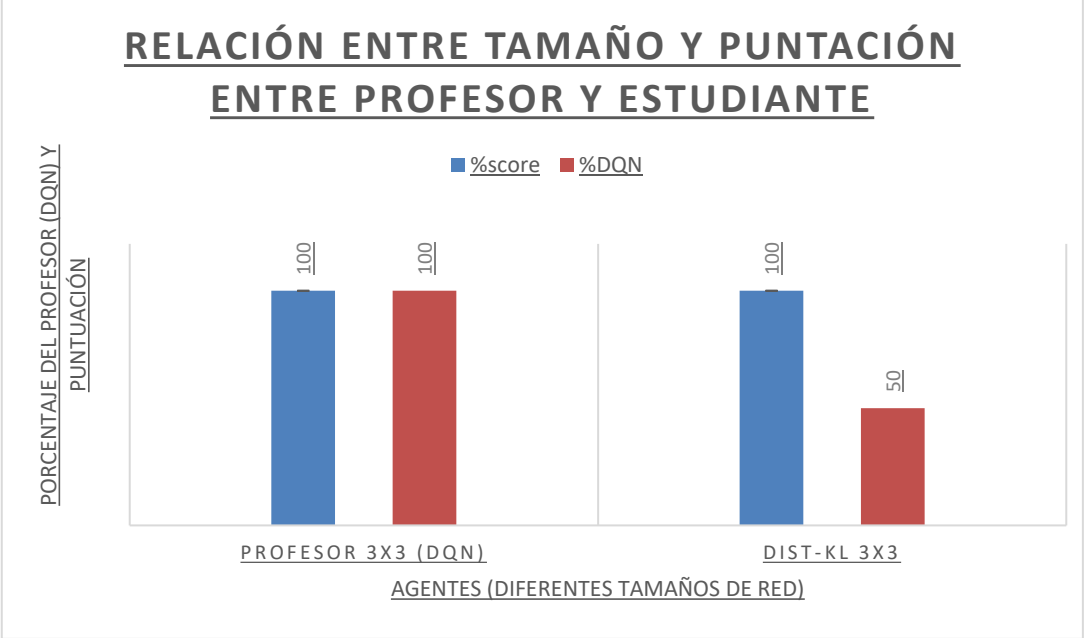


Figura 19: Gráfica de relación entre profesor y alumno en Turtlebot2

Una vez testeada esta nueva red y vistos los resultados de la Figura 19, la red destilada consigue igualar los resultados de la DQN original, pero con un 50% menos de neuronas en las capas ocultas.

6. CONCLUSIONES

En la introducción de este trabajo, se mencionaba la posibilidad de que empresas pudiesen crear comportamientos unificados globales mejorados a partir de los comportamientos de sus robots. Para hacer frente a esa posibilidad se ha desarrollado durante este trabajo el proceso necesario en el que, a partir de modelos de conocimiento heterogéneos, se pueda conseguir un comportamiento global unificado, tal y como se puede ver en el apartado de resultados (Capítulo 5).

En primer lugar, se han desarrollado políticas individuales y distintas entre sí, que resuelven una serie de problemas (laberintos), y a su vez se podrían equiparar a los conocimientos globales que tendría cada empresa.

Una vez desarrolladas estas políticas, se ha llevado a cabo la destilación, proceso que ha servido para reducir el tamaño de las políticas individuales y que no ha supuesto una pérdida en el rendimiento de los modelos.

Por último, con las políticas individuales y las destiladas, se ha conseguido mediante el uso de aprendizaje federado, un comportamiento global que aglutine el conocimiento de los modelos individuales, y que obtiene los mismos resultados que los comportamientos individuales. Demostrando de esta forma que es posible a partir de modelos heterogéneos conseguir un modelo global que aglutine todo ese conocimiento.

Tras comprobar que es posible obtener un comportamiento global, todo el desarrollo que se ha hecho, se ha aplicado a un robot real (concretamente un Turtlebot2) para así demostrar que todo lo desarrollado se puede aplicar a un robot real, como es el caso del Turtlebot.

Tras estos resultados, podemos concluir que el aprendizaje federado es una herramienta muy útil para obtener modelos de conocimiento globales, y que su aplicación puede ser de gran ayuda para obtener modelos más generales y manteniendo la privacidad de los datos, ya que en ningún momento los modelos comparten los datos locales con los que han sido entrenados, si no que solo comparten sus predicciones.

Por último, como líneas de investigación futuras, se podría destacar el desarrollo de una política global a partir de políticas heterogéneas y que hayan sido entrenadas en entornos distintos (distintos laberintos), ya que en este trabajo las políticas heterogéneas que más tarde son destiladas son siempre entrenadas en el mismo entorno, sin hacer una mezcla de estos entornos. Esto sería realmente útil para así conseguir políticas más globales y que puedan resolver distintos laberintos.

REFERENCIAS

[1] (Agosto 3,). *"Robótica"*. Available: https://concepto.de/robotica/.

[2] Calatrava Nicolás, F.M., Ortiz Zaragoza, F.J., Vera Repullo, J.A., Roca González, J., Jiménez Buendía, M., Martínez Mozos, O., " Sistema heterogéneo para la monitorización de la actividad diaria en el hogar y el bienestar de personas mayores," *XLII Jornadas De Automática: Libro De Actas,* pp. 632-639, 2021. . DOI: capítulo: Sistema heterogéneo para la monitorización de la actividad diaria en el hogar y el bienestar de personas mayores DOI libro: https://doi.org/10.17979/spudc.9788497498043.

[3] J. Ramos Pérez, "Diseño e implementación de un robot omnidireccional para fines de entretenimiento," 2020.

[4] S. Robla-Gómez *et al*, "Working Together: A Review on Safe Human-Robot Collaboration in Industrial Environments," *IEEE Access,* vol. 5, pp. 26754-26773, 2017.

[5] K. Azadeh, R. de Koster and D. Roy, "Robotized warehouse systems: Developments and research opportunities," 2017Available: http://hdl.handle.net/1765/99983.

[6] J. F. Ávila-Tomás, M. A. Mayer-Pujadas and V. J. Quesada-Varela, "La inteligencia artificial y sus aplicaciones en medicina II: importancia actual y aplicaciones prácticas," *Atención Primaria,* vol. 53, *(1),* pp. 81-88, 2021. Available: https://www.sciencedirect.com/science/article/pii/S0212656720301463. DOI: 10.1016/j.aprim.2020.04.014.

[7] (Agosto 17,). *Robot de almacén Kiva de Amazon con bastidor modelo 3d* . Available: https://free3d.com/es/modelo-3d/amazon-kiva-warehouse-robot-with-rack-5659.html.

[8] (Noviembre 13,). *El robot Pepper ficha por Lopesan*. Available: https://www.hosteltur.com/125105_robot-pepper-ficha-lopesan.html.

[9] (Junio 27,). *Robot Industrial Para Soldar*. Available: https://rivasrobotics.com/robot-industrial-para-soldar/.

[10] Vozpopuli, "Los mejores robots aspiradores para limpiar mejor que nunca " 2023. Available: https://www.vozpopuli.com/parati/electronica-tecnologia/mejor-robot-aspirador.html.

[11] (Enero 18,). *¿Qué Es el Aprendizaje Federado?*. Available: https://la.blogs.nvidia.com/2021/01/18/que-es-el-aprendizaje-federado/.

[12] C. Zhang *et al*, "A survey on federated learning," *Knowledge-Based Syst.,* vol. 216, pp. 106775, 2021. . DOI: 10.1016/j.knosys.2021.106775.

[13] A. Nilsson *et al*, *A Performance Evaluation of Federated Learning Algorithms.* 2018. DOI:

10.1145/3286490.3286559.

[14] K. Thonglek *et al*, *Federated Learning of Neural Network Models with Heterogeneous Structures.* 2020. DOI: 10.1109/ICMLA51294.2020.00120.

[15] T. Lin *et al*, "Ensemble Distillation for Robust Model Fusion in Federated Learning," *Advances in Neural Information Processing Systems,* vol. 33, 2020. Available: https://infoscience.epfl.ch/record/286861/files/NeurIPS-2020-ensemble-distillation-for-robust-model-fusion-in-federated-learning-Paper.pdf.

[16] D. Guliani, F. Beaufays and G. Motta, "Training speech recognition models with federated learning: A quality/cost framework," in - *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP),* 2021, . DOI: 10.1109/ICASSP39728.2021.9413397.

[17] R. S. Sutton and A. G. Barto, "Reinforcement Learning: An Introduction," *IEEE Transactions on Neural Networks,* vol. 9, *(5),* pp. 1054, 1998. . DOI: 10.1109/TNN.1998.712192.

[18] (Diciembre 24,). *Aprendizaje por refuerzo*. Available: https://aprendemachinelearning.com/aprendizaje-por-refuerzo/.

[19] K. Hwang, J. Lin and J. Li, "Biped Balance Control by Reinforcement Learning," *J. Inf. Sci. Eng.,* vol. 32, pp. 1041-1060, 2016.

[20] C. Sueldo *et al*, *Optimización Y Control Del Flujo De Materiales En Procesos De Producción Flexibles Utilizando Aprendizaje Profundo.* 2021.

[21] (Octubre). *An Introduction to Q-Learning: A Tutorial For Beginners*. Available: https://www.datacamp.com/tutorial/introduction-q-learning-beginner-tutorial.

[22] M. Roderick, J. MacGlashan and S. Tellex, "Implementing the Deep Q-Network," 2017.

[23] E. Arwa and K. Folly, "Reinforcement Learning Techniques for Optimal Power Control in Grid-Connected Microgrids: A Comprehensive Review," *IEEE Access,* vol. 8, pp. 1-16, 2020. . DOI: 10.1109/ACCESS.2020.3038735.

[24] J. Qi *et al*, "Federated reinforcement learning: techniques, applications, and open challenges," *Intelligence & Robotics,* vol. 1, 2021. . DOI: 10.20517/ir.2021.02.

[25] A. Rusu *et al*, "Policy Distillation," 2015.

[26] (). *OpenAI*. Available: https://openai.com.

ANEXO A

En este anexo se describirán los pasos necesarios para poder ejecutar el código proporcionado, a modo de "manual de usuario", para todas las personas que deseen replicar todo lo que se ha hecho en este trabajo o incluso continuar la línea de investigación.

En el código proporcionado se presentan dos carpetas: "1_CP_TurtleGym" y "2_CP_turtle_tfg".

La primera carpeta contiene todo el código necesario para la obtención de los modelos y resultados presentados hasta la sección 5.2. incluida, y también algunos programas necesarios para las pruebas realizadas en el robot real (Sección 5.3.).

Dentro de esta carpeta ("1_CP_TrutleGym") existen otras tres carpetas con información.

La primera de ellas es la carpeta "data", que contiene todos los archivos ".txt" de datos necesarios para obtener tanto las gráficas de evolución de Q-Learning y DQN, como los archivos ".txt" de datos necesarios para la destilación de las redes (tanto individuales como los *ensembles*).

Los archivos ".txt" para obtener las gráficas de evolución están nombrados de la siguiente manera: "XXXSimulacionesYYYZZZ.txt" o "XXXSimulacionesYYYZZZ_AAA.txt", donde:

- XXX son el número de simulaciones (30 o 500).
- YYY es el algoritmo utilizado ("Qlear" para Q-Learning, y "DQN" para DQN).
- ZZZ indica el laberinto sobre el que se entrenado el algoritmo (3x3, 4x4, 5x5 o 6x6).
- AAA indica el tamaño de la red, de la forma CxBB, donde C es el número de capas ocultas y BB es el número de neuronas por capa.

Estas gráficas pueden obtenerse ejecutando el programa Python "plotea.py", modificando los archivos de datos correspondientes en el programa.

Además, dentro de la carpeta "data", también están los datos necesarios para hacer la destilación. Para cada modelo destilado existen dos archivos, uno con las entradas y otro con las salidas. Estos archivos tienen una forma del tipo "XXXYYYZZZ.txt" o "XXXYYYZZZ_AAA.txt" donde:

- XXX indica si es un archivo de entrada con el nombre "Inputs" o un archivo de salida con el nombre "Outputs".
- YYY indica si es una destilación de un modelo individual con "Destilación" o si es una destilación de un ensemble con "Comité".
- ZZZ indica el laberinto en el que se ha hecho el entrenamiento algoritmo (3x3, 4x4, 5x5 o 6x6).
- AAA indica el tamaño de la red que se destila.

En caso de haber un archivo don un nombre que no siga este patrón, su función quedará explicada en el propio nombre del archivo.

La siguiente carpeta es la carpeta "models", en ella se guardan los modelos obtenidos de los entrenamientos. Dentro de esta carpeta están los modelos de Q-Learning nombrados de la siguiente forma: "QlearXXX.pkl", donde XXX es el laberinto en el que se ha entrenado. Por otro lado, están los modelos de DQN entrenados individualmente, la destilación de estos modelos y la destilación de los *ensembles.* Primeramente, se indica el laberinto en el que está entrenado, seguido de esto, se indica si son modelos individuales ("turtle_weights"), modelos destilados ("distilled_weights") o modelos destilados del *ensemble* ("comité_weights"). Por último, se indica el tamaño de la red usada, y en algunos casos de la destilación, el tamaño de la red original y el tamaño de la red destilada.

Para poder usar estos modelos, existen varios programas Python disponibles dentro de la carpeta inicial

"1_CP_Turtlegym". Para probar los modelos de Q-Learning existe el programa "testQlear.py", para probar los modelos de DQN individuales el programa "test.py", y para las destilaciones el programa "test_destilacion.py". En estos programas habrá que hacer los cambios necesarios para ajustar el laberinto y el tamaño de red al modelo que se quiera testear.

La última carpte es "turtle_robot_gym", que es la carpeta en la que se ha creado el entorno de simulación (los laberintos). Los entornos de simulación se encuentran dentro de la carpeta "envs" que se encuentra dentro de "turtle_robot_gym". Existen varias versiones de los laberintos en los que la principal diferencia entre ellos es la forma de codificar el espacio de estados. En este trabajo se ha utilizado la versión del entorno "turtle_robot_env_v1_2.py". Aunque para el entrenamiento de la DQN en el Turtlebot real se ha utilizado la versión "turtle_robot_env_v1_4.py", en la que se implementa parte de la comunicación con el Turtlebot real.

Por último, dentro de la carpeta "1_CP_Turtlegym" existen varios programas que todavía no se han mencionado:

- "main.py": con este programa se entrenan los modelos de Q-Learning.
- "mainDQN.py": con este programa se entrenan los modelos de DQN.
- "destilacion.py": con este programa se obtienen los modelos destilados.
- "create_comite_dataset.py": con este programa se crean los archivos (que se guardan en "data") necesarios para poder destilar los *ensembles*.
- "client_dqn.py": es un programa auxiliar que utiliza el programa "turtle_robot_env_v1_4.py" para crear una conexión TCP con el Turtlebot.
- "dqn_turtle.py" y "servidor.py" son programas que utilizaron para el entrenamiento en el Turtlebot real y se explicarán a continuación.

Para lanzar cualquiera de estos programas se puede lanzar de la siguiente forma en una terminal:

python3 nombre_programa.py

La segunda carpeta que nos encontramos es "2_CP_turtle_tfg". Esta carpeta contiene dentro el paquete de ros necesario para ejecutar el entrenamiento y movimientos en el Turtlebot real. Este paquete se llama "turtle_tfg" y dentro contiene varias carpetas. La primera es "data", que contiene dentro el archivo "AccionesQlearning.txt", que nos permitirá generar la gráfica de evolución del entrenamiento de Q-Learning que se ha llevado a cabo en el propio Turtlebot. Para generar esta gráfica, se puede utilizar el programa Python mencionado anteriormente "plotea.py".

La siguiente carpeta dentro del paquete es "models", en la que se encuentra el modelo de Q-Learning tras finalizar todos los entrenamientos "Qlear3x3.pkl", el modelo tras 1 episodio de entrenamiento "Qlear3x3_1.pkl" y tras 20 episodios "Qlear3x3_20.pkl".

Por último, en la carpeta "src" se encentran los programas necesarios para el entrenamiento y movimiento del robot. Los programas son los siguientes:

- "movimiento.py": es el programa que entrena el algoritmos de Q-Learning en el Turtlebot.
- "test.py": es el programa para testear el modelo de Q-Learning entrenado.
- "train_dqn.py": este programa sirve para entrenar la DQN en el Turtlebot real. Además de este programa es necesario lanzar el programa "dqn_turtle.py" de la carpeta "1_CP_TurtleGym".
- "dqn.py": este programa sirve para testear el modelo de DQN entrenado. Para lanzar este programa es necesario también lanzar el programa "servidor.py" de la carpeta "1_CP_TurtleGym", que creará un servidor TCP con el Turtlebot para la comunicación entre ambos.
- "turtlebot_communication.py": es un programa auxiliar para crear la comunicación entre Turtlebot y ordenador.

Para lanzar cualquier programa que se encuentre dentro de la carpeta “src”, se puede lanzar de la siguiente forma:

```
rosrun nombre_programa.py
```

Hay que tener en cuenta que antes de lanzar cualquier programa en el Turtlebot real, primero es necesario tener lanzado el nodo que nos permitirá acceder a la velocidad de los motores y el nodo que nos permitirá acceder a la información del lidar. Para lanzar estos nodos es necesario ejecutar en una terminal los siguientes comandos (cada uno en una terminal):

```
roslaunch kobuki_node minimal.launch
```

```
roslaunch rplidar_ros rplidar_a3.launch
```