Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería de Computadores

Trabajo Fin de Grado

Técnicas de aprendizaje por refuerzo aplicadas a la navegación de un robot movil

Autor: Javier Rufo Paris

Tutor: Ignacio Parra Alonso

2022

UNIVERSIDAD DE ALCALÁ

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería de Computadores

Trabajo Fin de Grado

Técnicas de aprendizaje por refuerzo aplicadas a la navegación de un robot movil

Autor: Javier Rufo Paris

Director: Ignacio Parra Alonso

Tribunal:

Presidente: Iván García Daza

Vocal 1º: Antonio Guerrero Baquero

Vocal 2º: Ignacio Parra Alonso

Calificación: .....

Fecha: .....

Resumen

El objetivo de este proyecto es el diseño, implementación y entrenamiento de una red neuronal. La red será entrenada por medio de aprendizaje por refuerzo y su objetivo será controlar un vehículo. Este vehículo se desplazará por un escenario con obstáculos que deberá esquivar hasta alcanzar la meta marcada.

El proyecto será desarrollado en MATLAB, haciendo uso de su toolbox de entrenamiento por refuerzo.

Para lograr el objetivo de este trabajo, se comienza haciendo un estudio de la situación actual en lo que respecta a los coches autónomos. Tras esto, se realiza el correspondiente estudio teórico de todos los conceptos que son la base de este trabajo: inteligencia artificial, redes neuronales, reinforce learning, la toolbox de Matlab, entre otros.

Posteriormente, se procede al desarrollo práctico del objetivo del proyecto. Se implementa el entorno de entrenamiento, se diseña la arquitectura de la red, se plantea la función de recompensa y se crea el agente. Una vez realizados los desarrollos, se procede al entrenamiento del agente en el entorno y se optimiza el desarrollo mediante el ajuste de los parámetros tras los diferentes entrenamientos realizados.

Finalmente, tras la consecución de unos resultados de entrenamiento adecuados, se seleccionarán diversos agentes que hayan obtenido la máxima recompensa y se procederá a realizar diferentes simulaciones para validarlos.

Palabras clave: Inteligencia artificial, red neuronal, entrenamiento por refuerzo, agente, vehículo autónomo.

Abstract

The objective of this project is the design, implementation and training of a neuronal network. This network will be trained using reinforce learning and its objective will be the control of a vehicle. This vehicle will move through a scenario with obstacles which it shall avoid until it reaches the aim.

The project will be developed by using the reinforce learning toolbox of Matlab.

In order to achieve the project's aim, the state of the art of autonomous vehicles is analyzed. Later, an exhaustive study about main concepts is done: artificial intelligence, neuronal networks, reinforce learning and Matlab's toolbox.

Next step is designing the training environment, the network architecture and the reward function and creating the agent. Then, the training of the agent and the optimization of the development of this work by adjusting the different parameters after the trainings will be done.

Finally, after achieving adecuate results of the trainings, some of the agents with maximus reward obtanied will be selected and, then, some simulations will be performed to validate them.

Keywords: Artificial intelligence, neuronal network, reinforce learning, agent, autonomous vehicle.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

En este proyecto vamos a trabajar con Machine Learning para el control de un vehículo autónomo. La utilización de Machine Learning en esta situación se debe a sus características, pues permite a la máquina aprender sin intervención humana.

Dentro del Machine Learning disponemos de distintas tecnologías: supervisada (Supervising Learning), sin supervisión (Un supervising Learning) y reforzada (Reinforcement Learning).

En este proyecto se dispondrá de un vehículo que debe desenvolverse por un entorno con obstáculos que deberá sortear para lograr llegar a un destino final. El vehículo cuenta con pocas fuentes de información y debe descubrir de forma experimental donde está el destino.

Para el control del vehículo se optó por aprendizaje por refuerzo. Se ha optado por esta tecnología por varias razones. La primera razón para optar por el aprendizaje por refuerzo, es, que no se necesitan de un set de entrenamiento. Para los entrenamientos solo hay que definir un entorno con el que el vehículo pueda interaccionar, probando y cometiendo errores. Otra ventaja es que el sistema no tiene un limite de aprendizaje, pues cuantas mas veces pueda actuar en el entorno mejor desempeño podrá lograr.

El aprendizaje por refuerzo ha aumentado su popularidad en los últimos años, una de las principales causas de este crecimiento de su popularidad es la publicación del articulo *Playing Atari with Deep Reinforcement Learning*, de DeepMind Technologies [1]. En la actualidad han ido surgiendo algoritmos que han proporcionado distintas mejoras, como: el uso de rangos continuos para las acciones a tomar, la reducción de episodios de entrenamiento o la reducción de la cantidad de experiencia necesaria para obtener una buena política. Pese a la evolución de estos algoritmos, los casos en los que su uso es viable sigue siendo limitado.

Capítulo 2

Estado del Arte

2.1 Introducción

En este capítulo se explica la situación actual de las principales marcas de vehículos con respecto a la conducción, así como de diversas empresas del sector tecnológico que mediante colaboraciones o desarrollos propios también están trabajando en estas tecnologías. Por otro lado se hablará de las diversas técnicas de IA que se están utilizando.

2.2 Industria automovilística

En la actualidad, lá fabricación de vehículos autónomos, es un objetivo prioritario para múltiples empresas, como pueden ser: Volkswagen y General Motors, las cuales tiene una larga historia en la fabricación de vehículos, o Tesla y Waymo (una divisón de Google), empresas tecnológicas que han visto en el vehículo autónomo un enorme potencial.

Las empresas citadas previamente no están realizando sus investigaciones en solitario, pues la mayoría de las empresas están llegando a acuerdos y asociaciones entre ellas, como la alianza formada entre el grupo Volkswagen y Ford o la de General Motors y Honda, para lograr el objetivo de un vehículo completamente autónomo. Es estas asociaciones no están participando exclusivamente empresas del sector automovilístico, pues compañías como Argo AI o Nvidia se están aliando para desarrollar un software para el análisis de los datos de trafico y que pueda ser usados por diferentes marcas. [2]

También existen empresas como Lyft y Aptiv las cuales buscan centralizar el funcionamiento de los sensores y las cámaras con los controles de los motores, a través de una serie de redes, para generar unos sistemas independientes del vehículo. Gracias a estos dispositivos y a un asociación con Hyundai han logrado crear una pequeña flota de taxis cuyo grado de autonomía es de 4 y 5 en la escala **SAE** (Sociaty of Automative Engineers) y que operará en Las Vegas a partir de 2022.

Para poner en contexto lo contado anteriormente se va a explicar en que consiste la escala SAE J3016. Esta clasificación consta de 6 niveles, los cuales van del 0 al 5, siendo el nivel 0 el nivel de conducción autónoma más baja y el 5 hace referencia a un sistema totalmente independiente. [3]

- **Nivel 0:** En este grado el vehículo no posee ningún tipo de autonomía y es completamente dependiente de un ser humano, sobre el que recaen todas las responsabilidades de la conducción.

- **Nivel 1:** En este nivel se encuentran aquellos vehículos que son capaces de controlar el movimiento longitudinal o lateral del vehículo, pero nunca ambos a la vez.
- **Nivel 2:** Se trata de una pequeña evolución del nivel anterior, pues en este caso se sigue manteniendo el control longitudinal y lateral, pero con la diferencia, de que en este nivel, se posee control sobre ambos movimientos a la vez. Esto supone una ayuda al piloto, aunque no le evita la tarea de la conducción.
- **Nivel 3:** Este es el primer nivel que exime al conductor de la tarea del pilotaje. Estos sistemas incorporan sensores capaces de detectar el entorno, y aportar respuestas acordes a estos. Este nivel aún tiene importantes limitaciones por lo que su uso esta limitado a unas condiciones muy marcadas. Ademas aunque el conductor puede liberar su atención de la tarea de conducir, debe estar siempre disponible y alerta para retomar el control si el sistema lo requiere, o cambian las condiciones.
- **Nivel 4:** En este grado, se amplían los casos en los que el vehículo es capaz de circular solo, y ademas el conductor no debe de estar atento por si el vehículo le reclama, dado que ahora se incorporan sistemas que ante imprevistos o cambios en el entorno, el vehículo, se estacionará en el lugar más cercano posible, que no suponga ningún riesgo. Pese a estos sistemas se sigue requiriendo la presencia de un conductor, pues el sistema sigue sin estar disponible para todos los escenarios posibles.
- **Nivel 5:** En este nivel se elimina completamente la dependencia al se humano y el vehículo es capaz de circula de una forma completamente autónoma sin importar las condiciones o el entorno. Pues es capaz de aportar respuesta ante cualquier evento que se le presente. Es el nivel más alto de autonomía posible, y aun ninguna empresa lo ha logrado.

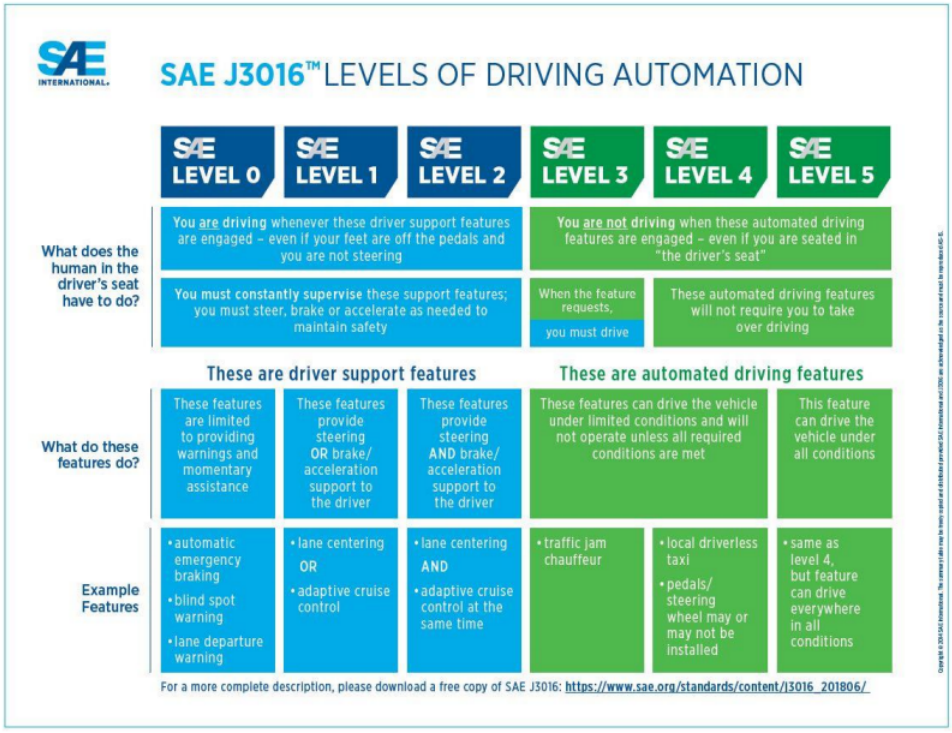

Figura 2.1: Estándar SAE J3016

Actualmente ya hay vehículos de venta al publico que ya incorporan sistemas que se encuentran entre los niveles 3 y 4, como por ejemplo los coches de la empresa Tesla, los cuales incorporan sistemas como:

Cambio automático de carril, sistema autónomo de aparcado y conducción automática en autopistas, con la capacidad de coger las salidas necesarias.

Para lograr esta autonomía las empresas están utilizando un serie de sensores, que trabajando en conjunto aportan una información muy precisa del entorno. De entre estos sensores destacan 2 por encima del resto, los cuales son:

- **LIDAR:** Es un sensor de tiempo de vuelo (TOF, de sus siglas en ingles) que aporta información en 360º, de las distancias a los obstáculos situados en su horizontal. Muchas de las compañías ya están comenzando a utilizar la evolución de LIDAR. Esta nuevo sensor mantiene el concepto de distancia en 360º, pero modifica el sensor de tiempo de vuelo por, una cámara de tiempo de vuelo. Estos sensores no miden la distancia en linea recta a un punto, sino que generan una nube de puntes y de distancias, generando así una bóveda al rededor del vehículo.
- **Cámara:** Los vehículos, sobre todos aquellos mas avanzados, incorporan un conjunto de cámaras al rededor del coche. La imagen aportada por las cámaras no poseen nada en particular, en este punto es donde comienza a brillar la inteligencia artificial de la que se hablará más tarde, pues todas las imagen aportadas por las cámaras son tratadas con redes neuronales convolucionales, y sistemas de visión artificial. Estas redes descomponen las imágenes y las analizan, para lograr asilar cada elemento existente, y posteriormente analizar dichos componentes y determinar que es cada elemento. Mediante esto el coche es capaz de detectar, que es carretera, cuales son sus carriles, identificar otros vehículos, y diferenciarlos de peatones, animales u otros elementos.

Empresas como Tesla han tomado la decisión de prescindir completamente del LIDAR y enfocar todo el esfuerzo en los sistemas de visión. Para solventar el problema que nos presenta las ausencia del lidar y con ello perder la información de la distancias al resto de objetos, Tesla incorpora un sistema de 8 cámaras en 360º al rededor del coche. Estas cámaras no son iguales entre si, ni están colocadas en los mismos ángulos. La idea es tener cámaras de gran angular cerca de los laterales y cámaras de gran distancia en la parte frontal y trasera. Además sus posición y ángulos están pensados para que siempre haya dos cámaras observando el mismo punto. Gracias a aportar esta visión estereoscópica, donde un punto es visto con diferentes ángulos, y gracias a algoritmos de machine learning, el coche es capaz de calculas las distancias a los objetos que le rodean, generando un escenario con profundidad y dimensionado. También se a de añadir que pese a renunciar al LIDAR en pos de la visión, los vehículos siguen necesitando un conjunto de radares, no tan completos como un LIDAR, pero que siguen aportando información en 360º. Estos radares actúan como sistema de respaldo, principalmente en aquellas ocasiones donde la visibilidad se vea reducida y por tanto la información de la visión artificial pierde fiabilidad. [4]

2.3 Inteligencia artificial

En esta sección se va a exponer de una forma genérica el estado en la que se encuentra las tecnologías de Inteligencia artificial en la sociedad actual.

En estos momentos está sucediendo el momento de mayor crecimiento de esta tecnología desde sus inicios en los años 70. Este crecimiento es debido a una multitud de causas. De entre todas ellas las mas importantes han sido el amplio crecimiento en las tecnologías de datos (BIG DATA), y el constante aumento de la potencia de computo. El crecimiento de las bases de datos y la mejora en las tecnologías que gestionan y estudian grandes volúmenes de datos, a desembocado en la posibilidad de generar conjuntos de entrenamientos masivos, y de ámbitos y temáticos muy diversas entre ellas. Estos conjuntos de entrenamiento tiene mucha importancia para el aprendizaje de una inteligencia artificial, pues utiliza

estos conjuntos para realizar sus entrenamientos. Cuantos mas variados y mayor volumen posean mejores resultados obtendrá el entrenamiento. A consecuencia de este aumento en la cantidad de datos que se pueden gestionar, es necesario un crecimiento semejante en la potencia de calculo. Dicho crecimiento se ha realizado de una forma natural siguiendo la ley de moore. Pero la clave en el crecimiento de la computación no se debe ha este crecimiento en el numero de transistores en los procesadores, sino en la utilización de unidades de procesamiento dedicadas. En un inicio se comenzó a utilizar las GPUs, las cuales habían sufrido una enorme evolución debido a la industria del videojuego, estas unidades diseñadas para realizar conjuntos de operaciones masivas sobre matrices, demostraron ser especialmente eficientes a la hora de realizar entrenamientos en las redes neuronales. A consecuencia de esto se comenzaron a producir unidades de procesado de redes neuronales, las cuales son chips diseñadas para realizar de forma exclusiva estos entrenamientos. El crecimiento de estas tecnologías desemboco en la capacidad de poder realizar nuevas tareas y mas complejas con inteligencia artificial. Con el crecimiento de las posibilidades, el interés y el estudio de las IA creció. Gracias al crecimiento en la tecnología de IA empezaron a surgir cada vez mas utilidades y diseños, lo que desemboca en la creación de una industria en la que cada vez mas empresas entran e invierten capital. Todos estos factores se retroalimentan entre si provocando el auge en el que vivimos actualmente.

También hay que tener en cuenta el crecimiento del las tecnologías en la nube (cloud en inglés), las cuales han acercado y facilitado las tecnologías de datos a las empresas. De este modo, el modelo tecnológico que las empresas deben implantar, para sus proyectos de ML, se basa en una arquitectura cloud híbrida entre MLOps y Kubernetes. Siendo una de las ventajas que esta solución aporta la escalabilidad sencilla de los proyectos. Estos avances conllevan que los modelos de ML se estén desplazando de una ciencia de datos a una ciencia de decisión. La diferencia entre ambas es notoria, pues la ciencia de datos se especializa en encontrar modelos que tengan una métrica que medir y cuantificar. Por otra parte la ciencia de la decisión nos permite desarrollar modelos de simulación, capaces de realizar distintas acciones ante diferentes eventos, incluyendo los casos más aislados. Para estos modelos cobra una enorme importancia el aprendizaje por refuerzo.

La Inteligencia Artificial es uno de los mayores avances tecnológicos de la historia, hasta el punto de que se habla de que supondrá una nueva revolución industrial, la llamada cuarta revolución industrial (Granell 2016).

Esta cuarta RI está marcada por un fuerte contexto social en el que la sociedad contemporánea se esta viendo expuesta a una explosión de contenido digital y su demanda se está volviendo cada vez más alta. Esta fuerte demanda está exigiendo a las organizaciones que gestionen datos masivos, un gran flujo de información y conocimiento. Para lograr superar estas nuevas exigencias, se ha encontrado en la IA y la CD, una disciplina capaz de ofrecer un marco teórico, metodológico y aplicativo, capaz de ofrecer a individuos y empresas las herramientas para lograr superar los retos cognitivos y logísticos planteados. Además se habilitan un conjunto de medios para que se desarrollen los sistemas ciberfísicos que constituyen las nuevas fábricas inteligentes. Como se puede comprobar en la Fig.1 donde se detalla un escenario ilustrado en la cual la CD regula un ciclo industrial compuesto por la explotación de conjuntos de datos, su posterior transformación en grupos de información, para posteriormente ser analizados y generar un conocimiento capaz de soportar la toma de decisiones. Mientras la IA se encarga de recoger ese conocimiento y representarlo para buscar e inferir las soluciones para superar los problemas complejos a consecuencia del esfuerzo de intentar simular el comportamiento que tendría una mente humana, estas soluciones son implementadas como aplicaciones capaces de interactuar con su entorno y reguladas para mejorar su eficiencia. [5][5]

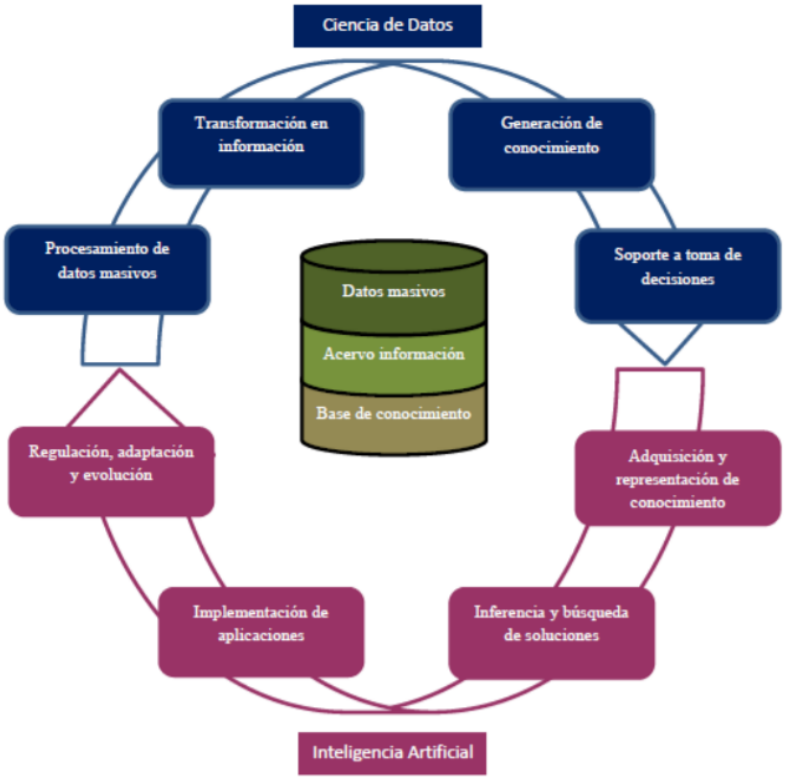

Figura 2.2: Ciclo de la ciencia de datos y la IA

2.3.1 Perfil de la inteligencia artificial

En la actualidad una de las tareas de la IA es lograr que aporte respuestas como lo hace la mente humana. Para evaluar dichas respuestas y lo cercanas que esta se encuentra de la aportada por un ser humano se ideó el "Test de Turing". Los agentes creados se someten a este test para demostrar si su desempeño, ante las respuestas lingüísticas, es similar a las aportadas por un agente humano. Y aunque las respuestas lingüísticas suele ser la que mas peso tiene a nivel social, también se evalúa su desempeño a nivel sensorial y motora. Este aspecto nos es muy importante pues en un mundo donde los vehículos deberán compartir las carreteras con agentes humanos, el lograr una circulación donde los vehículos autónomos puedan participar sin interrumpir el funcionamiento orgánico de la vía sera importante.

2.4 Visión Artificial

La Visión Artificial ha tenido un gran auge en los últimos años, en gran parte debido al rápido crecimiento que han experimentado las tecnologías de IA, de la cuales la muchos sistemas de Visión Artificial utilizan. Por otra parte el auge de la IA, en sectores como la robótica o la automoción, ha provocado un crecimiento paralelo de la Visión Artificial, debido a que estos sistemas requieren poder reconocer el entorno e identificar aquellos elementos que lo componen. Como ejemplo de esto tenemos los brazos robóticos de diversas lineas de montaje o clasificación, los cuales deben reconocer la forma exacta y la posición de aquel elemento que deben recoger. Por otra parte tenemos el ejemplo de los robots de Boston Dynamics, que son cuadrúpedos o bípedos, estos son capaces de desenvolverse con total naturalidad por terrenos irregulares y con obstáculos gracias a sus sistemas de visión. Por último tenemos el sector automovilístico

que es el que nos atañe, donde tenemos en empresas como tesla un claro ejemplo de la importancia de estos sistemas para el funcionamiento de sus vehículos autónomos, pues no poseen ninguna otra manera de reconocer el entorno.

Como un buen ejemplo de un sistema, en España, se están utilizando un conjunto de unidades robóticas para la ayuda en la rehabilitación de personas y en los cuidados de personas dependientes. Los cuales se encuentran en la unidad de Geriatría del Complejo Hospitalario de Cáceres. Por otro lado en el Hospital de Día se utiliza diariamente un robot especialmente diseñado para interactuar y ayudar a pacientes con la enfermedad de Parkinson. [7]

Capítulo 3

Estudio teórico

En este apartado se expondrán las técnicas y el conocimiento requerido para la realización del proyecto. De una forma más concreta se hablara de las redes neuronales como del aprendizaje por refuerzo.

3.1 Redes Neuronales

Las Redes Neuronales son modelos comunicacionales que desempeñan tareas de regresión y clasificación. Las Redes Neuronales pretenden trasladar a un modelo matemático y computacional el funcionamiento de las neuronas. Para ello las redes están formadas por perceptrones, los cuales son su unidad mínima, estos son los encargados de simular el comportamiento de una neurona.

Cada uno de estos perceptrones o neuronas (Inventadas en 1958 por Frank Rosenblatt) son en si un pequeño modelo matemático. Los modelos matemáticos consisten en su mayoría en una de las siguientes funciones: tanh, sigmoide, ReLU o SoftMax (Existen mas funciones que son utilizadas como modelos, pero estas cuatro son las mas utilizadas). Estas funciones tiene como principal característica la capacidad de recibir múltiples entradas y dar como resultado un único valor. La función recoge todas las entradas y les asigna a cada una un peso propio por el que sera multiplicada. Tras aplicarse el peso a cada entrada se realiza un sumatorio de las mismas, para así obtener un único valor al que se le añadirá un sesgo. Dando así como resultado de la ecuación una función lineal.[8]

$$y = \sum_{i}(x_i * w_i) + b \tag{3.1}$$

Donde:

y = Es la salida de la neurona

x_i = Se corresponde a la entrada i de la neurona

w_i = Es el peso aplicado a la entrada i

b = Es el sesgo aplicado a la neurona (bias)

Al conjunto de neuronas se la llama red neuronal, y se organizan en capas. Las capas dentro de la red están conectadas entre si, pues la salida de la primera capa constituyen la entrada de la siguiente capa , conectando todas las salidas de las neuronas que constituyen la capa, a las entradas de todas las neuronas de la siguiente capa. Y así sucesivamente hasta llegar a la ultima capa de la red. Esta secuencia de entradas y salidas en las capas solo es diferente tanto en la primera capa como en la última, pues la entrada que recibe la primera capa no proviene de otras neuronas, sino, que son directamente los datos

de entrada a la red. De tal forma la salida de la ultima capa no va dirigida a otras neuronas, sino, que se trata de la salida definitiva de la red. Esta estructura es común para todas las tecnologías que utilizan redes neuronales. Aunque en los últimos años ha surgido un nuevo modelo llamado redes neuronales recursivas y como evolución de estas, los Transformers, que serán explicadas posteriormente. Visto de otra forma, la red neuronal es una gran función, compuesta de funciones lineales encadenadas. Debido a este encadenamiento de funciones lineales obtenemos como resultado la construcción de una función no lineal.

Como ya se ha expuesta cada neurona consta de una serie de entradas, y a cada una de esas entradas se le es asignado un peso, el cual se trata de un multiplicador, que refleja la importancia de ese dato para la solución final. Aunque existan múltiples neuronas que comparten los datos de entrada, el peso que se le asigna a cada entrada es propio de cada una de las neuronas. Para poder descubrir que peso debe corresponder a cada entrada se realiza un entrenamiento de la red, ajustando los pesos de cada neurona. Ese entrenamiento se realiza analizando la salida aportada por la red y su diferencia con el valor esperado para esa entrada. El ajuste se realiza calculando ese error obtenido a través de la diferencia entre valor obtenido y valor deseado, y propagando hacia atrás ese error desde la ultima capa hasta la primera. A todo el proceso de propagación del error se le conoce como *backpropagation*.

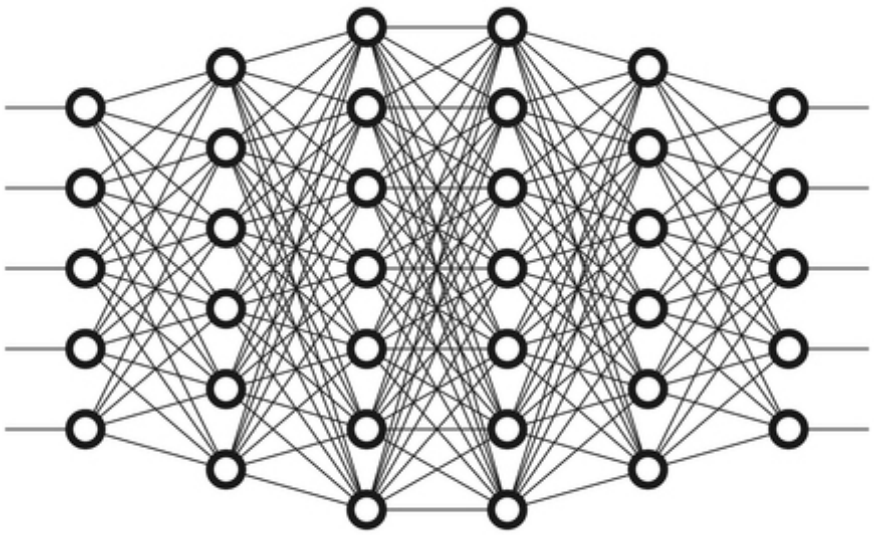

Figura 3.1: Esquema de una red neuronal

En la Figura 3.1 se observa una red neuronal formada por múltiples capas interconectadas entre ellas. Las capas de los extremos representan las capas de salida y de entrada, mientras que el resto representan las capas ocultas.

3.1.1 Introducción

Gracias a los estudios de Ramón y Cajal y Sherrington, en España e Inglaterra respectivamente, se logro un gran incremento en el conocimiento sobre el cerebro humano y su funcionamiento. Estos trabajos desarrollaban el funcionamiento a nivel anatómico de las neuronas y de sus interconexiones o conexiones sinápticas.

Las neuronas están divididas en diferentes elementos, y cada uno de ellos cumplen unas funciones muy concretas. Para comenzar en el cuerpo o soma de la neurona es donde se encuentra el núcleo. Es en

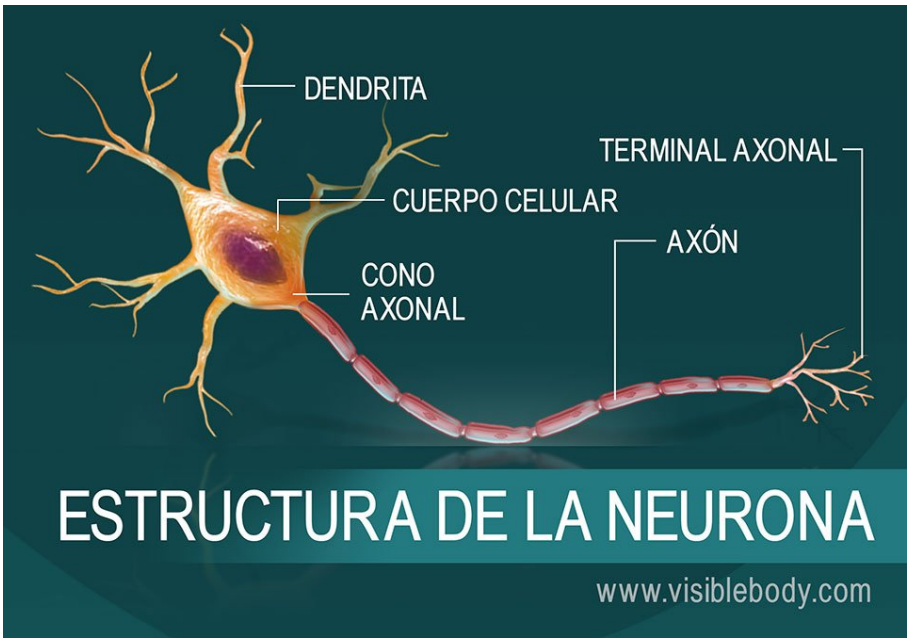

Figura 3.2: Modelo de una Neurona humana

el núcleo donde se realizan las funciones principales de las neuronas, a la vez que recibe la información de las neuronas vecinas a través de las conexiones sinápticas. La estrada de información, proveniente de otras neuronas, entran por las dentritas. Para la salida de información, los impulsos nerviosos dirigidos a las neuronas vecinas, sale por el axón. El axón se ramifica en el extremo, para poder formas conexiones sinápticas con los somas de diferentes neuronas. Estas conexiones entre células se realiza mediante la trasmisión de señales producidas por procesos químicos.

El cerebro humano está formado por sistemas de neuronas con diferentes funciones. La estructura que forman comienza con un conjunto de neuronas de entrada (que podríamos calificar de sensores), las cuales se conectan a una compleja de red de neuronas, denominadas como 'calculadoras'(que se correspondería con una capa oculta), y en ultima estancia se conectan con las neuronas de salida, estas neuronas son las encargadas del control de las acciones, por ejemplo, la respuesta muscular.

3.1.2 Funcionamiento de las redes neuronales

El Deep Learning (DL) o aprendizaje profundo es un subconjunto del Machine Learning mas extendidas. La tecnologías de DL se caracterizan por utilizar redes neuronales profundas (DNN de sus siglas en ingles), cuya principal característica es que su composición consta de múltiples capas ocultas entre la capa de salida y la de entrada. Esta tecnología se inspira en el funcionamiento cognitivo que tiene el cerebro humano, para aprender a resolver sistemas basándose en el proceso de grandes sistemas de datos. Estos sistemas suelen atender a un problema concreto y especifico. Para explicar esto de una mejor forma se va a proceder a explicar el parecido con el cerebro humano de donde se obtiene la inspiración del funcionamiento. El cerebro humano está formado una diferentes conjuntos de neuronas, formando diferentes núcleos de redes neuronales. Estas redes que componen los núcleos están especializadas para resolver tareas muy concretas. Por ejemplo el reconocimiento de que es un rostro humano, recibiendo una imagen y devolviendo si se trata o no de un rostro. Estas redes se comunican entre ellas dentro y fuera de los núcleos, de forma que la resolución de los pequeños problemas para los que están diseñadas la redes, se pueden resolver problemas más complejos.[9]

Si se traslada este esquema a la IA tenemos como resultado que el DL es una imitación de estos

modelos naturales.

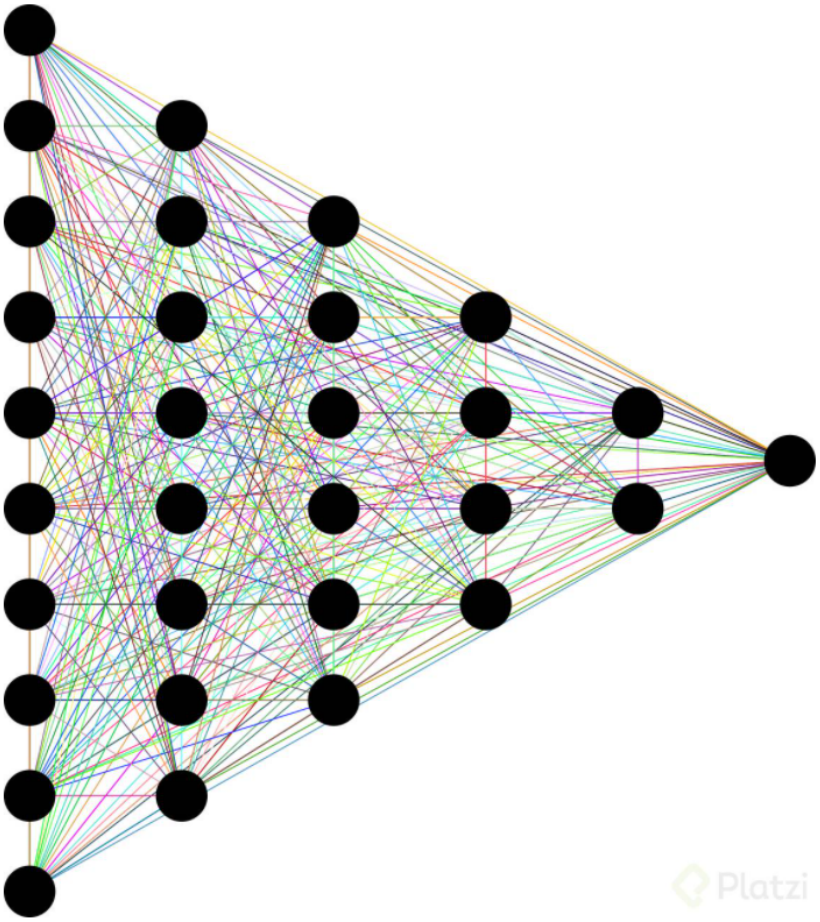

Figura 3.3: Esquema de una red neuronal

El DL recibe su nombre por como se estructuran su neuronas. Como se observa en la Figura 3 donde las neuronas de la izquierda constituyen la capa de entrada y la neurona de la izquierda es la capa de salida. El resto de las neuronas se dispones por capas escalando el tamaño de cada capa, los parámetros de estas capas son desconocidos, recibiendo el nombre de capas ocultas. La información recibida entra a la red por la capa de entrada, y va pasando a través de un gran numero de capas antes de formar una respuesta. Otra forma de definir el DL es como un conjunto de algoritmos no lineales, para la modelización de datos y la detección de patrones. Esto significa que las capas de la red posen jerarquías en función de su nivel de abstracción siendo las capas inferiores, de menor abstracción, las capas de entrada y las de mayor abstracción las de salida, siendo estas las mas complejas.

Las Redes Neuronales Artificiales (RNA) son en su base un modelo matemático, que como ya se ha explicado imitan el funcionamiento de las redes cerebrales. Gracias a su estructura las RNA poseen una gran flexibilidad, lo que permite que con una misma red se resuelvan problemas totalmente distintos. Al igual que en las redes biológicas las RNA tiene como componente básico las neuronas artificiales. La neurona (perceptrón), al igual que la red, es un modelo matemática, que forma un dispositivo simple de calculo con la capacidad de generar una única respuesta ante un conjunto de datos de entrada. Este modelo matemático consiste en una función matemática, mejor denominadas como funciones de activación. Las funciones mas utilizadas son tanh, sigmoide, ReLu o SoftMax. La neurona recibe una o varias entradas y aporta un resultado, y su funcionamiento principal es la realización de una suma ponderada de las

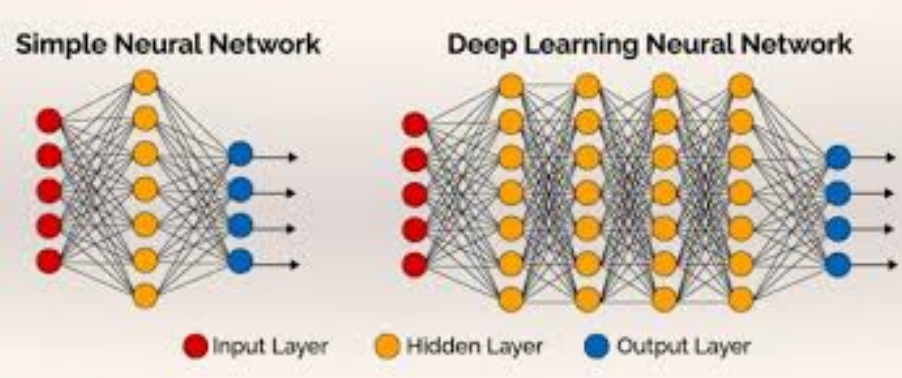

Figura 3.4: Red Neuronal Simple VS Deep Learning Network

entradas, cada entrada es previamente multiplicada por un peso, y finalmente se le aplica la suma de un sesgo. En la ecuación (2.1) se observa el modelo matemático de un perceptrón. [10]

$$y = \sum_{i} (x_i \times w_i) + b \tag{3.2}$$

donde:
y = salida de la neurona
x_i = entrada i de la neurona
w_i = Peso de la entrada correspondiente
b = Sesgo de la neurona

Al igual que con las neuronas biológicas, las artificiales se organizan en tres grupos en función de su situación dentro de la red.

- **Capa de entrada:** Encargada de la recepción directa de la información proveniente del exterior, e introducir la información en la red.
- **Capas ocultas:** Internas de la red. Son las encargadas del procesado de los datos de entrada.
- **Capa de salida:** Transfiere los resultados al exterior.

Dentro de una RNA pueden existir múltiples capas ocultas, o no existir ninguna. Cuando la salida de una neurona se dirige a la entrada de múltiples neuronas de la siguiente capa, estas salidas llegan a las entradas sin sufrir modificaciones, por lo que el valor que todas las neuronas reciben es el mismo. Es dentro de cada neurona donde se le aplican los pesos a las entradas, aplicando cada neurona un peso propio al mimo dato de entrada.

Como se ha explicado previamente, las RNA tienen como unidad básica el perceptrón, o neurona artificial. Estas están compuestas por diferentes elementos.

- Conjuntos de datos de entrada x_i. Para i = 1 .. m
- Pesos sinápticos propio para cada datos de entrada, w_i. Para i = 1 .. m
- Regla para la propagación S. Se trata de una función matemática que se encarga de combinar los datos de entrada con los pesos.

  Por norma suele ser: $s(x_1, ..., x_m, w_1, ..., w_m) = \sum_{i=1}^{m} w_i x_i$

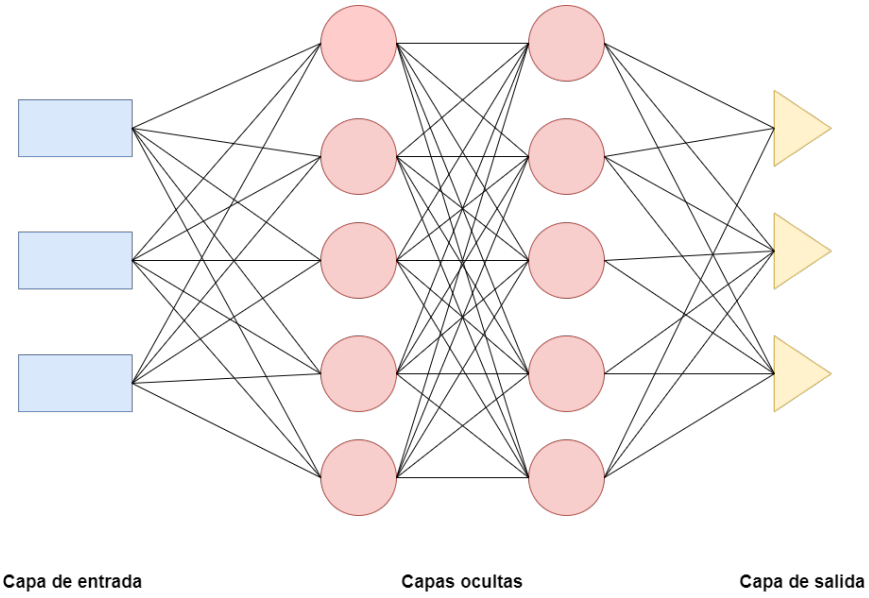

Figura 3.5: Representación de las conexiones entre capas de una RNA

- La función de activación g, la cual se trata de una función que opera S con una constante U(sesgo o umbral). Como resultado de esta operación se obtiene el resultado final de la Neurona o y.

$$y = g(s-u) = g(\sum_{i=1}^{m}(w_i x_i - u) = g(\sum_{i=0}^{m}(w_i x_i)) \tag{3.3}$$

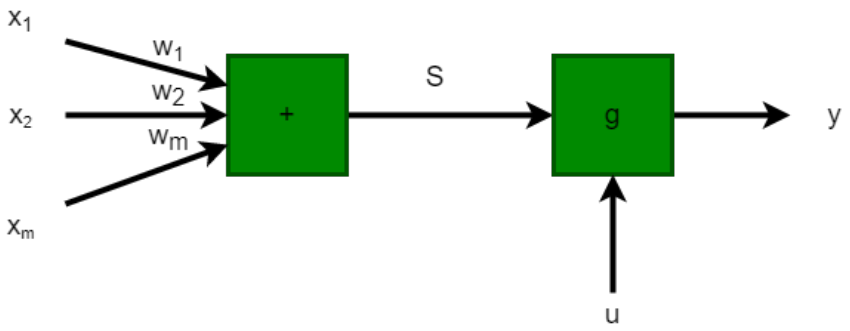

Figura 3.6: Neurona

Existen varias maneras de clasificar las RNA, atendiendo a diferentes criterios.

Si se tiene en cuenta el numero de capas encontramos dos grandes grupos: Redes monocapa, en las cuales solo existe un nivel de de neuronas unidas mediante conexiones laterales; y Redes multicapa, las cuales poseen las neuronas distribuidas en dos o mas capas.

Atendiendo al flujo de datos podemos clasificarlas de dos formas: Redes unidireccionales (feedforward) y Redes recurrente sobrealimentadas (feedback).

La ultima forma en la que se puede clasificar las RNA es atendiendo a algoritmo de aprendizaje, que es cualquier forma por la cual la Red asigne los valores para los coeficientes sináptios:

- **Aprendizaje supervisado:** A la red se le suministra un conjunto de ejemplos, estos ejemplos constan de los datos de entrada y de la salida esperada a cada uno de ellos.

- **Aprendizaje no supervisado:** Los ejemplos que se le suministran a la RNA solo constan de los datos de entrada.

- **Aprendizage hibrido:** En este modelo existen capas con aprendizaje supervisado y capas con aprendizaje no supervisado.

- **Aprendizaje por refuerzo:** En este modelo de aprendizaje solo se disponen de los datos de entrada, pero si se le indica a la red si la solución a la que ha llegado es correcta o no.

3.1.3 Funciones de activación

Como ya se ha introducido previamente las neuronas que forman una RNA cuentan con una función de activación. Esta función se encarga de procesar la información aportada por los datos de entrada y sus pesos. Por tanto su principal función es la transmisión de la información generada por la combinación lineal de los datos de entrada y los pesos. A través de esta función es como se define la forma de los datos que sera enviada a la salida. Debido a que el objetivo de la red es resolver problemas que son cada vez mas complejos, estas funciones hacen que los modelos generados sean no lineales. [11]

Las funciones mas utilizadas son las siguientes.

- **Función lineal (Identidad):** Esta función no modifica los datos de la salida con respecto a la entrada. En redes multicapas en las que se aplica una función lineal se la llama regresión lineal. La Función lineal se utiliza cuando se desea obtener una red neuronal con un resultado único. Por ejemplo, se utiliza para la predicción del numero de ventas.

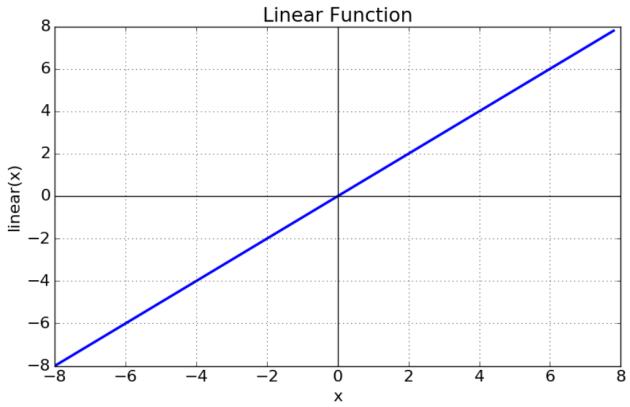

Figura 3.7: Función de activación

- **Función Umbral:** También conocida como función de paso, su funcionamiento es muy sencillo, pues, convierte todos los resultados que estén por debajo de 0 en 0 y todos los que sean iguales o superiores a 0 pasan a valer 1. Es muy utilizada para la clasificación. Por ejemplo cuando se quiere saber si se compro algo o no.

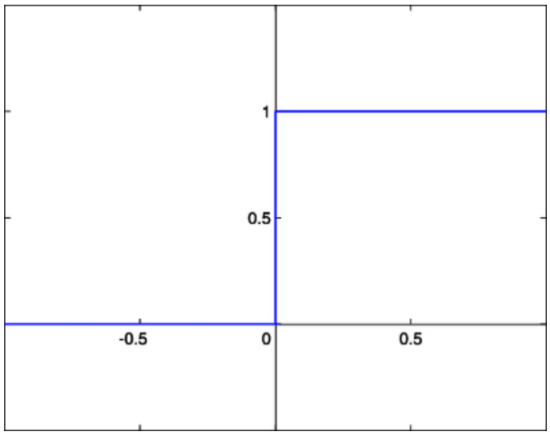

Figura 3.8: Funcion umbral

$$g(x) = \begin{cases} 1x \geq 0 \\ 0x < 0 \end{cases} \tag{3.4}$$

- **Función Sigmoide (Logistica):** Los valores de salida de esta función están comprendidos en el rango de números reales de 0 a 1. La salida de esta función se entiende como una probabilidad. Los resultados que se encuentran por debajo del cero dan como resultado un valor cercano al cero, los resultados que se encuentran en el 0 como resultado dan el valor de 0,5 y todos los que se encuentran por encima del cero, adquieren un valor cercano al 1. Es una función que se suele utilizar en la ultima capa, se utiliza para agrupar datos en dos categorías. Por desgracia esta función ha ido cayendo en desuso debido a que la función no esta centrada, y este hecho tiene un efecto negativo en el aprendizaje y entrenamiento, agravado por el problema de la desaparición del gradiente.

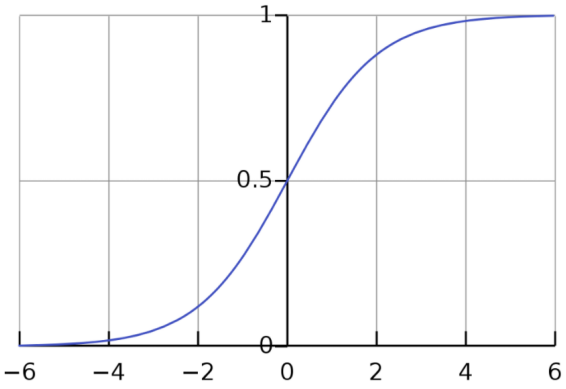

Figura 3.9: Funcion sigmoide

$$f(x) = \frac{1}{1 - e^{-x}} \tag{3.5}$$

- **Función Tangente hiperbólica:** Los valores de esta función están comprendido entre el -1 y el 1. Se trata de una función que escala de la función logística, y a pesar de que esta centrada, mantiene el problema de la gradient vanishing (Problema surgido cuando el entrenamiento genera un error

con el algoritmo de back propagation y debido a esto el error se propaga entre las diferentes capas).

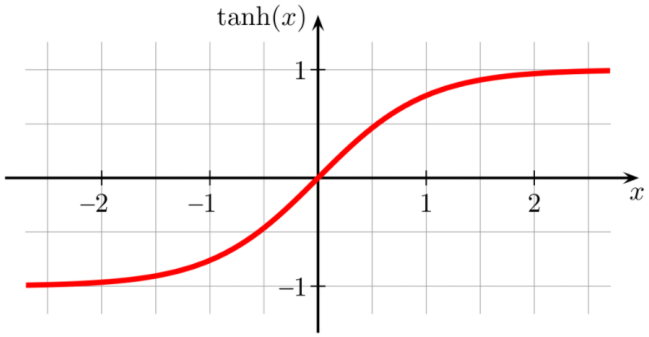

Figura 3.10: Función Tangente hiperbólica

$$f(x) = \frac{2}{1 + e^{-2x}} - 1 \tag{3.6}$$

- **Función ReLu:** Es la función mas utilizada de todas, gracias a que permite realizar el aprendizaje mas rápido de todas en las redes neuronales. Su funcionamiento es muy sencillo, pues para todos los valores inferiores a 0 el resultado es cero, mientras que para aquellos que estén por encima del cero el valor de salida es el mismo que el de la entrada. Otra ventaja que posee es que el gradiente de la función es en el segundo cuadrante y 1 en el primero. Cuando el resultado de la función da 0 y su derivada también se produce lo que se denomina como la muerte de la neurona, y pese a que en algunos casos puede resultar un gran problema, esto nos permite la regularización Dropout. Existe una variante de ReLu llamada Leaky ReLu, en la cual para los resultado inferiores a 0 pasan a generar una pequeña pendiente siendo y=0.01x, de esta forma se previene la aparición de neuronas muertas.

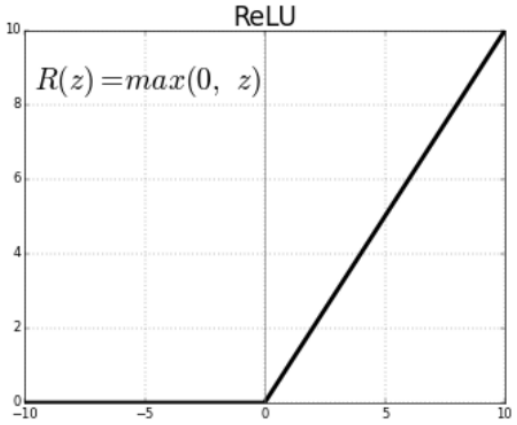

Figura 3.11: Función Tangente ReLu

$$f(x) = max(0, x) = \begin{cases} 0 for x < 0 \\ x for x \geq 0 \end{cases} \tag{3.7}$$

- **Función Softmax:** Es una función para la clasificación de datos. Las función otorga un porcentaje a cada una de las categorías que forman la clasificación, dándoles un valor entre 0 y 1 o entre 0 y 100 según la configuración. Para cada resultado la suma de los porcentajes de todas las posibilidades debe ser 1 o 100 en función del tipo de datos que hallamos escogido. Posteriormente se selecciona la categoría con mayor probabilidad como el resultado de la clasificación. Como ejemplo tendríamos un clasificador de ropa, donde se podrían generar una probabilidad de 0.3 o 30 % de que sea una camisa, 0.2 o 20 % de que sea un calcetín y 0.5 o 50 % de que sea un pantalón, por lo que la red daría como salida que es un pantalón.

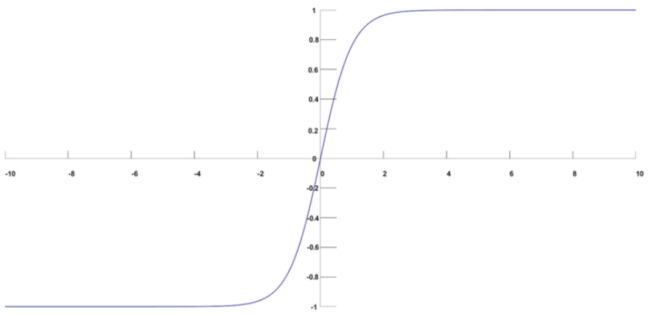

Figura 3.12: Función Tangente Softmax

$$f(x)_j = \frac{e^{z_j}}{\sum_{k=1}^{k} e^{z_j}} \tag{3.8}$$

3.1.4 Perceptrón simple

Se trata de una RNA muy simple, compuesta por dos capas de neuronas unidireccionales, que solo propagan la información hacia adelante. En la primera capa no se realiza ningún tipo de calculo, pues solo introducen los datos de entrada a la red, mientras que en la segunda capa, la única operación realizada es la función de activación paso.

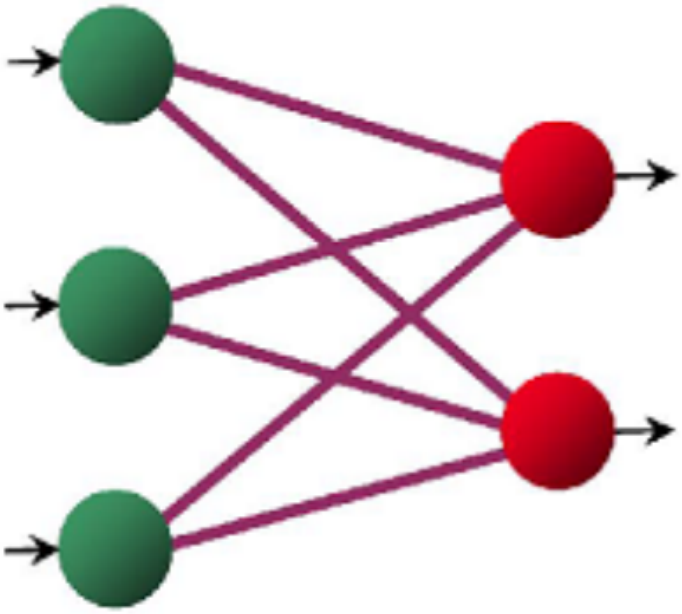

Figura 3.13: Perceptron simple

El perceptrón simple son utiliza como valores de entrada un vector $x = (x_1, x_2, ..., x_m)$, el cual es procesado con la función de activación paso.

$$y = \begin{cases} 1 si \sum_{i=1}^{m} w_i x_i + u \geq 0 \\ -1 si \sum_{i=1}^{m} w_i x_i + u < 0 \end{cases} \tag{3.9}$$

La utilidad de este tipo de RNA es muy limitado, usándose solo como clasificador binario. Es decir que el sistema solo es capaz de proporcionar si el conjunto de datos de entrada pertenece a la clase C1 o C2. Los perceptrones simples solo son capaces de clasificar conjuntos que sean linealmente separables (Fig 14).

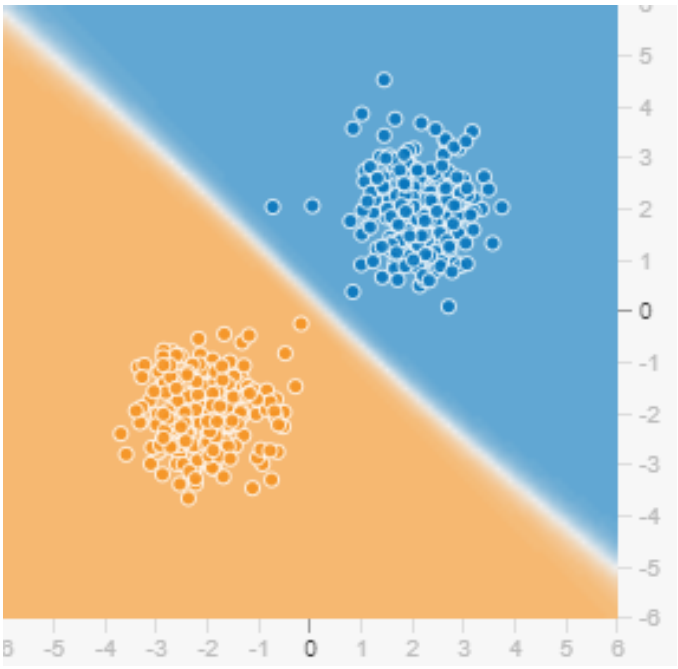

Figura 3.14: conjunto linealmente separable

El algoritmo que se usado para ajustar los pesos de esta RNA fue desarrollado por Rosenblatt, lo ideó como procedimiento de aprendizaje para su modelo de perceptrón del cerebro. El algoritmo consta de 6 pasos, que son los siguientes:

1. Se asigna un valor aleatorio a cada uno de los pesos de la red.
2. Del conjunto de entrenamiento se selecciona un vector de entrada.
3. Para el vector seleccionado se calcula su salida.
4. Se comprueba si la salida obtenida se corresponde a la salida esperada. De ser correcta la salida se vuelve al paso 2.
5. Para todas las neuronas se calcula el error obtenido en ellas
   $\triangle_{error_i} = \varepsilon(S_{esperada} - S_{obtenida})E_{entrada_i})$
   y posteriormente con el error $\triangle$ obtenido se actualiza el peso de cada neurona
   $w_i := w_i + \triangle_{error_i}$.
6. El ultimo paso es regresar al paso 2.

En el algoritmo que propuso Rosenblatt as variaciones de los pesos solo pueden ser $-2\varepsilon, 0o2\varepsilon$, donde ε se denomina coeficiente o tasa de aprendizaje y debe tomar un valor tal que $0 < \varepsilon < 1$. El algoritmo de se detiene en el momento en el que todas las entradas quedan correctamente clasificadas, siempre que se pueda. [12]

3.1.5 Perceptrón multiple

Minsky y Papert (1969) [13] demostraron que el perceptrón simple no puede resolver problemas no lineales. La combinación de varios perceptrones podría resolver ciertos problemas no lineales, pero no existía un mecanismo automático para adaptar los pesos de las capas ocultas.

Rumelhart et al. [14] introdujeron en 1986 el perceptrón múltiple (PM), que es una red compuesta por varias capas de neuronas con conexiones hacia adelante (feed-forward) y en la que el aprendizaje es supervisado.
En 1989, Cybenko [15] probó que el perceptrón multicapa puede aproximar a cualquier función continua, por lo que se considera un aproximador universal.

El perceptrón múltiple es una de las arquitecturas mas utilizadas en la resolución de problemas reales, debido a su capacidad para establecer relaciones entre entradas y salidas no lineales. Ademas provee un uso sencillo y un gran numero de situaciones en las que se puede utilizar. Las condiciones necesarias para que lo Perceptrones multiples puedan funcionar son las siguientes:

- **Entrenamientos de larga duración:** Para que el sistema logre una buena solución debe ser expuesto a entrenamientos muy largos.
- **Respuesta rapida:** Es necesario que ante nuevas instancias se aporte una respuesta muy rápida.

Los perceptrones múltiples consta de 3 capas :

- **Capa de entrada:** Recoge todos los datos de entrada del exterior y los propaga a la siguiente capa.
- **Capa de salida:** Proporciona la respuesta de la red ante un patrón de entrada.
- **Capa oculta:** Realiza el conjunto de procesos no lineales.

El algoritmo que se utiliza para hallar los pesos consta de dos pasos, un primero donde se calcula el resultado que la red provee ante unos datos de entrada, y un segundo paso en el cual el error obtenido en la salida se propaga hacia atrás (backpropagation) con el objetivo de modificar el valor de los pesos probados. Dicho algoritmo fue propuesto por primera vez en 1974 por Paul Werbos, originalmente se propuso como algoritmo general para cualquier tipo de red, siendo las redes neuronales solo un caso especial de su utilización. Originalmente este algoritmo no fue bien aceptado por la comunidad científica dedicad al estudio de redes neuronales debido a su carácter generalista. Posteriormente a mediados de 1980 el algoritmo de Backpropagation resurgió gracias a numerosos investigadores como David Rumelhart, Geoffrey Hinton, Ronal Williams, David Parker y Yann Le Cun que comenzaron a utilizarlo, y alcanzo una gran popularidad cuando David Rumelhart y James McClelland lo incluyeron en su libro *Parallel Distributed Processing Group?* Uno de los avances que trajo el algoritmo de backpropagation y que aumento considerablemente su popularidad, fue la capacidad de aprovechar la naturaleza paralela de de las RNA añadido al hecho de que nadie tiene que deducir el algoritmo que debe utilizar la red, sino que esta lo deduce por si misma.

El punto fuerte de este algoritmo radica en la capacidad que tienen las neuronas de las capas intermedias, de organizarse a si mismas durante el proceso de entrenamiento, esto provoca que las diferentes neuronas aprendan a reconocer distintas características del espacio total de entradas.

3.2 Deep learning

3.2.1 Introducción

"El Deep Learning esa una Ramificación del Machine Learning en la cual se mejora el concepto de aprendizaje no supervisado. En esta rama se hace uso de unas redes neuronales que se construyen tomando

como ejemplo las estructuras del cerebro humano, en la cual las neuronas se conecta entre ellas conformando una red similar al de una araña. Esta arquitectura permite la resolución de problemas no lineales. En DL la red aprende desde la experiencia y entiende el mundo que está percibiendo . Los algoritmos son capaces de aprender sin intervención humana previa, siendo la red la encargada de sacar las conclusiones de todos los datos". (Goodfellowm Bengio, y Courville, 2016)

El Deep learning (DL) es un conjunto de algoritmos que tratan de modelar una abstracción de alto nivel de los datos recibidos mediante el uso de una arquitectura compuesta de transformaciones no lineales múltiples. El Deep Learning pertenece al campo del Machine learning, siendo una evolución de las RNA vistas en el apartado 3.1. El DL en la actualidad esta aportando un excelente rendimiento en campos como el procesamiento del habla, el lenguaje natural y la visión por ordenador.

La principal diferencia con la RNA esta en el numero de capas ocultas que posee, pues estas poseen un gran numero de estas. Ademas, el DL, necesita de un conjunto masivo de datos de entrenamiento, ya que su rendimiento mejora cuanto mayor sea el set de entrenamiento. [16]

Existen tres factores que afectan al despeño de un red profunda (DL): la arquitectura, las técnicas de regulación y los algoritmos de optimización. Para lograr este correcto funcionamiento hay que atender a los siguientes aspectos:

- **Aproximación:** Una red neuronal debe poder aproximar funciones arbitrarias sobre los datos recibidos. Aunque las redes de una única capa oculta son un aproximador de funciones universales, se observa que aquellas arquitecturas que poseen mayor profundidad de capas son capaces de mejorar la captación de propiedades invariantes de los datos.
- **Generalización y regulación:** Las redes neuronales profundas son entrenadas con un numero inferior datos, pero pese a ello se debe tener cuidado del sobreajuste utilizando técnicas de regulación simples, como por ejemplo el Dropout, que consiste en seleccionar muestar aleatorias del set de entrenamiento en cada iteración del mismo.
- **Patrones importantes:** Se trata de una propiedad muy importante. Detectan fácilmente cual es la información más relevante y cuales son los criterios invariantes entre casos similares, lo cual es esencial para la predicción de resultados en el futuro.
- **Optimización:** Se utiliza el algoritmo *Stochastic Gradient Descent* (SGD), el cual es una evolución del algoritmo de *Backpropagation*.

3.2.2 Optimización global en Deep Learning

Las redes neuronales y las redes profundas, poseen el mismos funcionamiento básico, la diferencia que existe entre ellas está en la función de pérdida, la cual es la función a minimizar para encontrar los pesos óptimos. La función matemática con la que se describe el problema de optimización de una red profunda es el siguiente,:

$$\min_{\{W^k\}_{k=1}^{k}} \iota(Y, \phi(X, W^1, ..., W^k)) + \lambda\theta(W^1, ..., W^k) \tag{3.10}$$

Siendo $W = {W^k}_{k=1}^{k}$ el problema de la optención de pesos, en una red de N ejemplos (X,Y), donde $\iota(Y, \phi)$ se trata de la función de de pérdida que calcula la diferencia existente entre la salida obtenida por el modelo $\phi(X, W)$ y la salida correcta Y para ese ejemplo. Ademas tenemos que $\lambda geq 0$ se trata de un parámetro de equilibrio, también tenemos que θ es la función que regulariza y previene el sobreajuste, y por ultimo se observa W^k que representa el conjunto de pesos en la capa k, siendo $k \in \{1, ..., k\}$. [17]

3.2.3 Limitaciones del Deep Learning

Una de las características de los modelos de deep learning y su principal inconveniente es su alto consumo de recursos hardware. Se necesitan computadoras con una alta capacidad de procesado de datos, esto supuso un impedimento en sus orígenes en los años 80, pues no se disponía de la tecnología necesaria para lograr entrenamientos de modelos complejos con resultados satisfactorios. Por otra parte también se necesitan conjuntos de datos de gran magnitud para logra un correcto entrenamiento.

Pese a que estos problemas mencionados anteriormente, se han solventado con la tecnología actual, en la mayoría de los casos, las redes profundas no están exentas de problemas. Principalmente existen dos características con las que hay que tener especial atención, como son, el sobreaprendizaje y su carácter de caja negra. Para el primero de los problemas existe técnicas que permiten sortearlo, aunque no dan una solución definitiva al problema. Por el contrario para el problema de la caja negra no existe una solución, este problema puede suponer un alto riesgo de seguridad para sistemas que utilizan redes neuronales. [18][19]

3.2.3.1 Sobreaprendizaje

El sobreaprendizaje es un problema que surge durante el entrenamiento de una red, en la la función se adapta tan bien al conjunto de datos del entrenamiento, que deja de aportar resultados correctos en otros conjuntos del resto de casos del problema. Es uno de los problemas más habituales cuando se hace uso de técnicas de aprendizaje automático. El problema del sobreajuste viene ocasionado por la gran magnitud de parámetros ajustables: los pesos que configuran las relaciones que existen entre las neuronas.

Figura 3.15: comparación en el aprendizaje

Hay que tener cuidado con los conjuntos de entrenamiento, en ellos se encuentra las reglas que nos permiten construir un modelo de aprendizaje automático, pero también incluyen dos tipos de ruido: error en los datos y error de muestreo.

Los errores que existen en los conjuntos de datos son inevitables y vienen ocasionados por la forma en la que se obtienen los datos, este proceso es mas sensible a error cuando es un ser humano el encargado total o parcial del procesado manual de los datos. Por ejemplo tenemos el caso de que la etiqueta asociada a un caso sea errónea, una situación que es bastante común, principalmente cuando el responsable de realizar el proceso de etiquetado es un ser humano. Ademas hay que tener muy en cuenta que por muy grande que sea un conjunto de casos, no deja de ser nada mas que una población del conjunto de datos reales. En los conjuntos de muestra que se toman existen, de forma accidental, regulaciones ajenas al problema que se desea modelar, sino que son debidas al subconjunto de los datos. En el momento que se ajusta un modelo a un subconjunto de los datos no se puede saber si las reglas internas que el modelo ha encontrado son debidas a las características propias de los ejemplos tomados o son una generalización del conjunto de datos reales. Si el modelo que se ha diseñado tiene un alto nivel de flexibilidad será capaz de

modelar todas las reglas del conjunto de datos de entrenamiento, tanto aquellas propias de los ejemplos como las reglas reales del problema planteado. Por el contrario si el modelo se ajusta demasiado a los datos de entrenamiento, perderá la capacidad de generalizar el resultado y, en un uso real esto puede dar resultados con un gran error.

Existen varias Estrategias que se pueden utilizar par atenuar o sortear este problema. Como una primera opción tenemos por ejemplo el decaimiento de pesos (weight decay), por el cual se penaliza toda aquella presencia de pesos elevados dentro de la red. Otra opción es el uso de pesos compartidos (weigth sharing) que reduce el número de parámetros con los que la red neuronal tiene que operar, de esta forma se reduce la capacidad de aprender de la red y, por tanto reduce el sobreaprendizaje. Otra técnica que se puede utilizar consiste en modificar el propio algoritmo de aprendizaje para incorporarle una condición de parada, que detenga el entrenamiento de una forma temprana (early stopping), esta condición detiene el proceso de entrenamiento cuando se detecta un aumento del error en un conjunto de validación que sea distinto al conjunto de entrenamiento. [20]

3.2.3.2 Redes Neuronales como Caja Negra

Una Característica de las redes neuronales profundas, y que en función de la aplicación que se le de a esta puede resultar un gran inconveniente, es su cualidad de actuar como una caja negra, pues de la red solo se conoce la entrada y la salida, pero no como esta es capaz de llegar a sus deducciones.

En aquellas situaciones donde la importancia del sistema no consista solo en el valor del resultado, sino que también tenga importancia la validez del proceso de obtención del mismo, la cualidad de caja negra supone un gran problema.

Debido a que una red neuronal no llega a sus resultado de la misma forma en la que lo hace una persona, se ocasionan problemas de seguridad en todos aquellos ámbitos en los que son utilizados. [21]

- Los sistemas de recomendaciones han cobrado una gran importancia en nuestra vida cotidiana en el entorno online, la mayoría de esos sistemas ya utilizan redes neuronales para entender el comportamiento de las personas y poder así realizar sus recomendaciones. Estos algoritmos son victimas de constantes ataques que tratan de manipular su configuración interna para que se modifiquen las recomendaciones que realizan. Estos ataques pretenden promover u opacar algún producto. Se denominan ataques por complicidad (shilling attacks), y es debido al carácter de caja negra es muy difícil detectar estas manipulaciones.

- Se ha demostrado que las redes profundas son especialmente útiles en el reconocimiento de patrones complejos, tal y como es el reconocimiento de imágenes o visión artificial. En numerosas ocasiones se ha demostrado que es posible engañar las redes para que detecten cosas que en realidad no existen. Se ha llegado a un punto en el cual se puede crear una imagen a partir de otra que a ojos humanos son idénticas, pero que esconden información que confunden completamente a una inteligencia artificial, dando resultados que no tienen nada que ver con la realidad. Como ejemplo de este problema pensemos en un vehículo autónomo el cual posea una red neuronal encargada del reconocimiento de señales de trafico y semáforos. Si pensamos en esta vulnerabilidad y en la posibilidad de manipular las señales, los semáforos u otros elementos del entorno y el peligro que puede suponer eso para la sociedad. Estas vulnerabilidades son muy difíciles de solucionar dado que es imposible saber que es lo que lleva a la red a esos resultados erróneos.

3.2.4 Tipos de Deep Learning Networks

En los últimos 30 años se han desarrollado multitud de arquitecturas y algoritmos que utilizan aprendizaje profundo, y en este apartado vamos a explorar los más relevantes de ellos.

3.2.4.1 Redes Neuronales Recurrente (RNN)

Las RNN son modelos cuya principal aplicación esta en el reconocimiento de vos y el reconocimiento de texto escrito a mano. Se trata de uno de los primeros modelos que surgieron como derivados del deep learning en los años 90, y han sido utilizada como base para muchos de los modelos que surgieron posteriormente. La principal diferencia entre las redes multicapa y la redes recurrentes es que, las redes recurrente no solo tienen conexiones entre capas contiguas, sino que existen conexiones entre neuronas de la misma capa, capas anteriores e inclusos neuronas que conectan su salida a su entrada. Esta característica genera una retroalimentación que permite a la RNN poseer memoria de las entradas anteriores al sistema y modelas los problemas a tiempo. [22]

Las RNN pueden ser entrenadas con dos algoritmos. Se puede utilizar el algoritmo estándar de Back-propagation o realizar una variante de esta llamada propagación hacia atrás en el tiempo (BPTT).

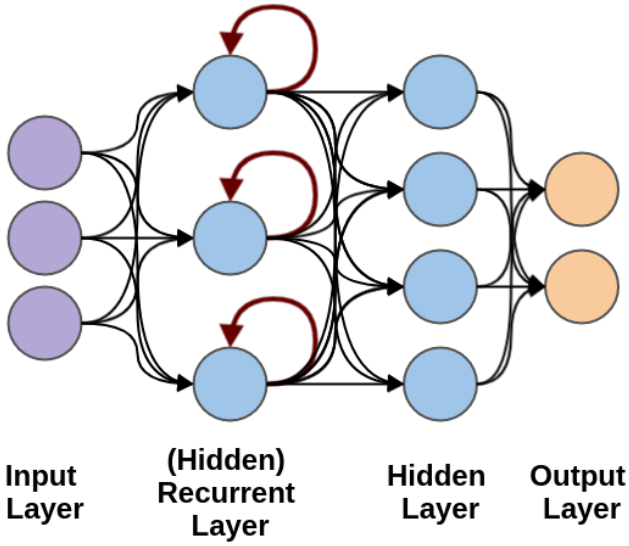

Figura 3.16: Recurrent Neural Network (RNN)

3.2.4.2 Long Short Term Memory (LSTM)

Las redes LSTM son redes que funcionan especialmente bien en la comprensión de texto en lenguaje natural, reconocimiento de escritura manual, reconocimiento de voz y gestos y subtitulos en imágenes. Las redes LSTM surgieron a finales de los 90 y fueron creadas por Hochreiter y Schimdhuber. En los últimos años su popularidad ha ido en aumento como una evolución de una RNN de múltiple propósito. Este auge en su uso está muy influenciado por las nuevas necesidades en el sector de los teléfonos inteligentes, uno de los sectores con mayor crecimiento en la ultima década.

Las redes LSTM cambian un poco el concepto de la red basada en neuronas e incluye un nuevo concepto llamado celdas de memoria. Estas celdas de memoria son capaces de retener su valor durante un periodo corto o largo de tiempo, que depende de las entradas recibidas. Esta característica le concede a la neurona capacidad de recordar valores importantes, y no solo el último valor procesado.

El funcionamiento de las celdas de memoria dentro de las redes LSTM consiste en tres puestas lógicas de control. Cada una de estas puertas controla un aspecto de la memoria. La primera puerta es la de entrada y determina cuando se debe guardar en la memoria un resultado, la segunda denominada puerta del olvido marca cuando el resultado almacenado en la celda debe ser eliminado, para así poder registrar nuevos valores, y por último está la puerta de salida, que indica cuando se debe utilizar el valor almacenado en la celda. Al igual que el funcionamiento de una red neuronal, las puertas que controlan la celda contienen unos pesos que dictan como se comportan. [23]

En la siguiente imagen se puede observar un neurona LSTM internamente.

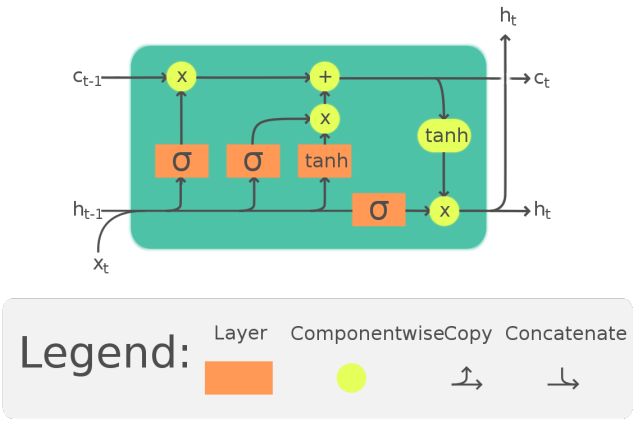

Figura 3.17: Long/Short Term Memory

3.2.4.3 Unidad recurrente cerrada (GRU)

En el año 2014 surgió una simplificación de las redes LSTM llamadas GRU. A diferencia de su antecesora las redes GRU eliminan 1 de las 3 puertas. Por lo general su funcionamiento y rendimiento es similar a las LSTM, pero debido a su mayor simplicidad y reducción en el numero de peros, permite una ejecución más rápida.

En las redes GRU las puertas que controlan las celdas de memoria reciben una modificación en su funcionamiento con respecto a su antecesor. La puerta de entrada se modifica por la puerta de actualización, que determina la cantidad de contenido de la celda anterior que se debe mantener. Por otra parte la puerta de olvido se cambia por la puerta de reinicio, que define cómo se debe incorporar la nueva entrada al contenido de la celda. Las redes GNU se pueden convertir fácilmente en un RNN preestableciendo el valor de las puertas, la puerta de restablecimiento a d1 y la de actualización a 0.

La principal diferencia entre la LSTM y la GRU es la eficiencia y los resultados. Al ser la segunda más simple, conlleva entrenamientos y ejecuciones mas sencillas y rápidas, mientras que la primera, son más expresivas y pueden conducir a resultados mejores. [24]

En la siguiente imagen su puede observar la comparación entre las tres evoluciones que se han visto.

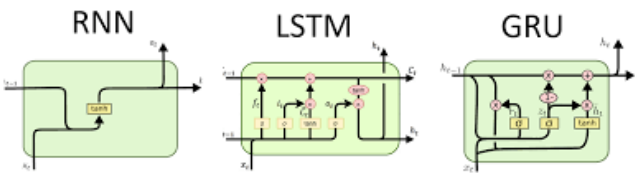

Figura 3.18: Comparación entre RNN, LSTM y GRU

3.2.4.4 Redes Neuronales Convolucionales (CNN)

Las CNN son redes neuronales multicapa que toman inspiración en en la biología de la corteza visual animal. Tiene su principal utilización en el reconocimiento de imágenes, vídeo y el procesado de lenguaje natural. Su utilización apareció rondando el año 2000, y fue creada por Yann LeCun. Su primera utilización fue para el reconocimiento del lenguaje manuscrito, y se utilizo en el reconocimiento de códigos postales. La primera capa de la red, reconoce características como bordes, y ñas capas posteriores las siguientes capas se recombinan todas las características simples en atributos de entrada de un nivel superior.

En las CNN cada una de las capas que la conforman se entrenan por separado para realizar una tarea, de esta forma se reducen el numero de capas ocultas, lo que aligera el proceso de entrenamiento haciéndolo más rápido. Además, es invariante por traslación de sus patrones a identificar.

Como ya se ha dicho anteriormente las CNN son especialmente buenas para el reconocimiento de imágenes. En los primeros procesos la red se encarga de identificar todos aquellos patrones simples, como lineas, bordes, formas geométricas, etc. Y posteriormente en los siguientes pasos se recomponen estas formas simples para generar estructuras complejas para poder detectar las características buscadas. Por ultimo estas estructuras detectadas son introducidas en una red multicapa de conexión total.

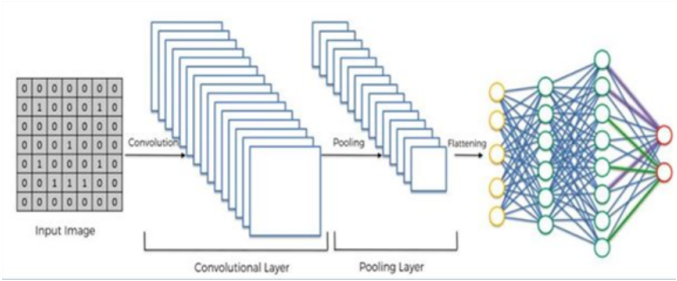

Figura 3.19: Red Neuronal Convolucional (CNN)

La convolución que se realiza consiste en un conjunto de operaciones de sumas y productos entre la capa de partida y los filtros que generan un mapa de características. cada una de las características extraídas se corresponden a todas las posibles ubicaciones que el filtro puede tomar en la imagen original. Su ventaja radica en la posibilidad de que cada neurona sea capaz de extraer la misma característica en cualquier parte de la entrada; con esto se consigue reducir el numero de conexiones y de parámetros a entrenar.

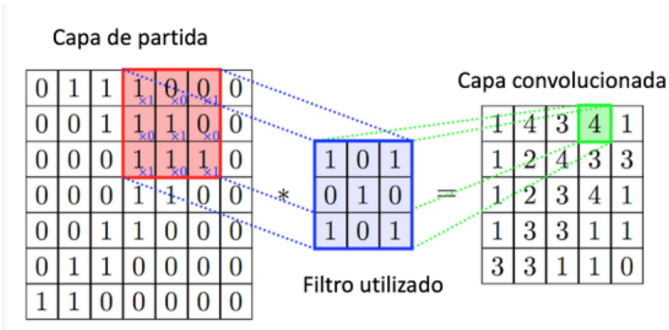

Figura 3.20: proceso de convolución

Tras obtenerse la convolución se aplica una función de activación a los mapas de características. La función mas recomendada es la sigmoide ReLu, aunque se podría utilizar otras, a excepción de la sigmoide logística. En el siguiente paso paso del proceso se realiza la reducción (pooling) por el cual se reduce el

numero de parámetros quedándose solo con las características más comunes. Para reducir el numero de parámetros se utiliza el promedio o el máximo de una región fija. La reducción hace que el método pierda precisión, aunque mejor su compatibilidad. Tras el proceso de convolución y de reducción por norma se utilizan capas completamente conectadas. Por ultimo la ultima capa de la red es una capa clasificadora con tantas neuronas como clases existan en la clasificación. [25]

3.3 Aprendizaje por refuerzo

3.3.1 Introducción

El Aprendizaje por refuerzo (Reinforcement Learning) o RL, es un área del machine learning donde el aprendizaje no se da comparando el resultado obtenido con el esperado, si no que se genera un agente el cual realizara acciones en un entorno y será la superación de ciertos hitos lo que le aportara recompensas o castigos. Es lo que se podría denominar un entrenamiento de prueba y error. Este sistema de aprendizaje es como los seres humanos y los animales aprenden a realizar acciones como las de caminar o manipular objetos físicos.

Esto difiere del aprendizaje supervisado, pues el principal objetivo de este es el de aprender una función determinada, tales como modelar relaciones y dependencias entre los resultados y las características que se tiene de estos. Por tanto se requiere de tener datos con una etiqueta de su solución final y minimizar el error que exista entre el resultado obtenido y la etiqueta.

En el RL el principal objetivo es interactuar con el entorno donde se despliega, para que mediante las observaciones y la experiencia, se realicen acciones que a largo plazo maximicen la recompensa y minimicen el castigo. En estos modelos lo importante siempre es lograr la mayor recompensa a largo plazo, por lo que el agente que desempeña las acciones puede tomar acciones que en primera instancia minimice esa recompensa, siempre y cuando al final, el resultado obtenido sea la mayor recompensa posible. Por ejemplo, en un entorno con un vehículo queremos llegar a un punto, pero entre el vehículo y el objetivo existe una barrera, la cual para sortearla debe alejarse mas del objetivo de lo que se encuentra en ese momento, el agente lo hará, de esa forma, a corto plazo estaría reduciendo su recompensa, pues se está alejando del objetivo, pero en el largo plazo su recompensa sería máxima al alcanzar el destino.

Al igual que en el Deep learning visto anteriormente, su estructura general esta compuesto por redes de neuronas multicapa interconectadas. Pero a diferencia de estas no es necesaria la creación de un conjunto de datos etiquetados para su entrenamiento, pues el aprendizaje se realiza con la experiencia obtenida por el desempeño en un entorno. Es un campo de investigación poco explorado, y el cual todavía no se ha llegado a realizar multitud de aplicaciones, como si sucede con los modelos supervisados, pero se trata de un sector que tiene un gran potencial futuro, y que ha encontrado en la conducción autónoma y la robótica sus principales aliados para logran una gran relevancia en el futuro. [26]

3.3.2 ¿En que consiste?

Como se ha mencionado en la introducción este modelo es muy útil para resolver aquellos problemas en los que no se disponga de los datos necesarios etiquetados, y se necesite realizar una tarea de acción predictiva

El procedimiento de aprendizaje se realiza mediante una secuencia de prueba y error, similar al proceso por el cual una persona aprende a hacer malabares. En un principio a la persona no será capaz de coger las pelotas una vez las lanza, y comenzara un proceso que mediante errores ira logrando poco a poco

ir recogiéndolas y lanzándolas, cometiendo menos errores según van avanzando los intentos. Además en ningún momento se aporta más información que la del objetivo a cumplir.

Partiendo de este ejemplo, cambiamos a la persona por un robot con conocimiento nulo. Para que se produzca el aprendizaje se debe dejar al robot que realice acciones y decisiones aleatorias en un entorno específico. Las acciones y las decisiones que se deban considerar como buenas se recompensaran, por el contrario, las acciones que se deben considerar como malas se castigaran. El valor del castigo o la recompensa queda determinada por una función que es propia para cada problema que se plantee.

De forma general para cualquier algoritmo de aprendizaje por refuerzo, si la función de recompensa esta bien diseñada, al cabo de muchos episodios de pruebas, el robot habrá sido capaz de aprender a tomar las decisiones correctas para un estado determinado.

3.3.3 Elementos del Reinforcement learning

En este apartado se va a especificar cuales son los elementos que componen un algoritmo de aprendizaje profundo.

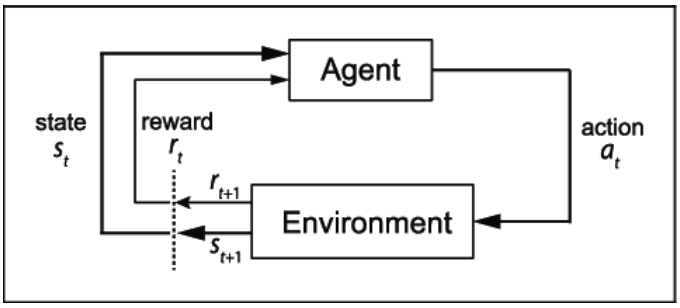

Figura 3.21: Modelo de un entorno de aprendizaje profundo

De una forma general, el proceso de aprendizaje sigue los siguientes pasos:

- El Agente (Agent) observa el estado (state).
- El Agente, mediante una función de toma de decisión, determina la acción (action) a realizar.
- Tras la acción realizada, el entorno (environment) determina la recompensa (Reward) que se le debe aportar al agente.
- Se almacena la recompensa obtenida sobre la experiencia realizada de estado-acción.

3.3.3.1 Agente (Agent)

Es el elemento encargado de aprender para superar el problema definido. El agente es una entidad que posee memoria y deducción. Su principal tarea es enfrentarse al entorno sobre el que se despliega, tomando las decisiones necesarias para lograr el objetivo planteado, y aprender basándose en las recompensas-castigos que recibe del entorno, para así maximizar la recompensa que reciba. El proceso que lleva a la deducción de una acción se la llama política. El objetivo del agente es encontrar la política que optimice sus recompensas. Esta política se actualiza en el proceso de entrenamiento, para su actualización se se utilizan las acciones pasadas, las recompensas obtenidas, y las observaciones recibidas. [27]

$$Politica \begin{cases} acción = funcion(S_t) \end{cases} \tag{3.11}$$

Dentro del RL es dentro del agente donde se encuentra la red neuronal, y es el agente el encargado de probar las múltiples configuraciones para lograr hallar la política optima. Las entradas a la red son las observaciones del entorno, las recompensas y las propias acciones realizadas por el agente; y como salida la red provee una actualización en la función. Como toda red profunda, esta red pude plantearse de diversas maneras, y con un numero de capas variables.

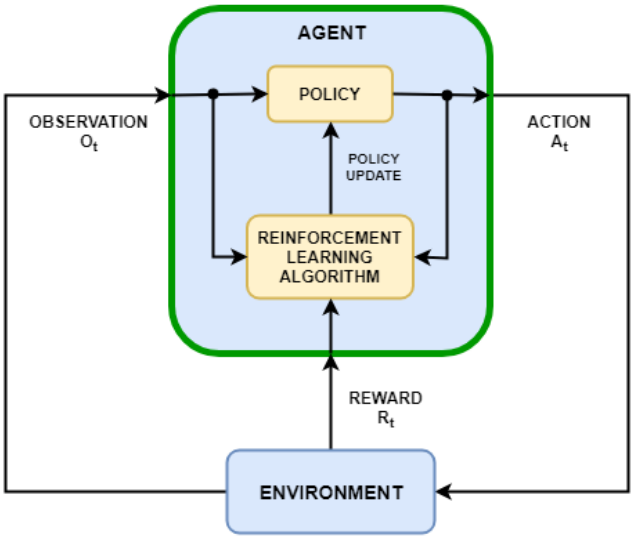

Figura 3.22: Modelo interno de un agente

Por ejemplo en el problema que se nos plantea, el agente sería el vehículo.

3.3.3.2 Entorno (environment)

Es todo aquel espacio en el cual el agente desempeña sus acciones, y con el cual puede interactuar. También es el encargado de determinar la recompensa que se le debe aportar al agente.

Para nuestro caso, el entorno es el simulador, el cual es un mapa plano con obstáculos.

3.3.3.3 Estado (State)

Representa la situación en el estado de tiempo actual en la cual el agente se encuentra dentro del entorno. Como por ejemplo, la posición, el ángulo o la velocidad, que el vehículo tiene en el momento.

El estado se puede representar como S_t donde t denota la instancia de tiempo actual.

3.3.3.4 Acción (Action)

Representa la acción que el agente toma para un estado concreto S_t. En RL el objetivo del agente es hacer que la acción tomada sea la optima para el estado asociado. La acción se representa como a_t, siendo t la instancia actual del tiempo.

Existen dos tipos de valores que las acciones pueden tomar, discretas o continuas. Escoger una u otra depende del problema planteado.

- **Discreta:** Existe un número fijo de posibilidades entre las que el agente tiene que escoger. Por ejemplo un robot clasificador el cual solo, puede agrupar los objetos recibidos en el grupo A o el B.

- **Continuas:** Existe un numero infinito de valores que se encuentran dentro de un rango de valores. Para el coso del vehículo la dirección o la velocidad pueden tomar cualquier valor dentro del rango que permiten las características del vehículo

3.3.3.5 Episodio

Es una división del proceso de entrenamiento, se podría expresar como, cada uno de los intentos en el proceso de aprendizaje. Parte desde una unidad de tiempo 0 hasta el final de las acciones y la entrega de la recompensa. Es la sucesión de los intentos, lo que forma el proceso de aprendizaje.

Por ejemplo, en el caso del vehículo tenemos como unidad de tiempo 0, el origen del vehículo, y como final, la colisión de este con un obstáculo o la superación del recorrido y llegada a la meta.

3.3.3.6 Recompensa (Reward)

El concepto de la recompensa es fundamental para el aprendizaje por refuerzo, dado que es esta misma el único parámetro, que la red recibe, que le indica lo buena que ha sido la respuesta que el agente a dando ante los estímulos del entorno. Para poder cuantificar lo bien que se desempeña el agente en el entorno, se debe diseñar una función de recompensa que lo cuantifique. La recompensa recibida debe ser un valor escalar, de forma que cuanto mejor sea el resultado mayor sera la recompensa, y viceversa.

La función de la recompensa tiene como parámetros el estado del agente en el entorno y la acción que este efectúa para este estado.

$$Reward = function(state, action) \tag{3.12}$$

La función de recompensa se describe matemáticamente como el sumatorio de las recompensas en cada instante de t, y donde k es la duración del episodio.

$$G_t = \sum_{k=0}^{T} T_{t+k+1} \tag{3.13}$$

3.3.4 Técnicas de aprendizaje por refuerzo

3.3.4.1 Cadenas de Márkov

Las cadenas de Markov son procesos que suceden en un tiempo discreto en el cual, una variable aleatoria X_n cambia con el tiempo. Reciben su nombre del matemático **Andréi Márkov**. En las cadenas la probabilidad de que suceda un evento $(X_n = j)$ depende únicamente del evento inmediatamente anterior (X_{n-1}). Cuando en una cadena las probabilidades no depende del tiempo en el que se observa, n, se le llama **cadena homogénea**, e indica que las probabilidades son las mismas en cada paso. [28]

$$P(X_n|X_1...X_{n-1}) = P(X_n|X_{n-1}) \tag{3.14}$$

Si en una cadena Homogénea finita con m posibles estados, y existe la posibilidad de ir de un estado a otro, se denominan **probabilidades de transición**.

$$p_{ij} = P(X_n = j|X_{n-1} = i) \tag{3.15}$$

Esta ecuación describe las posibilidades de pasar de un estado **i** al **j**, para una cadena homogénea finita con m posibles estados $(E_1, E_2, ..., E_m)$. Donde **i,j** = 1, 1, ..., m. En el caso de que $p_{ij} > 0$ significa que el estado E_i puede comunicar con E_j, ademas si la probabilidad $p_{ji} > 0$ se dice que la comunicación puede ser mutua.

La forma mas común de representar una cadena de Márkov es con un grafo dirigido, donde cada nodo representa un estado diferente y la uniones entre los nodos son las probabilidades de transición. La suma de todas las transiciones que salen de un nodo tiene que ser igual a 1.

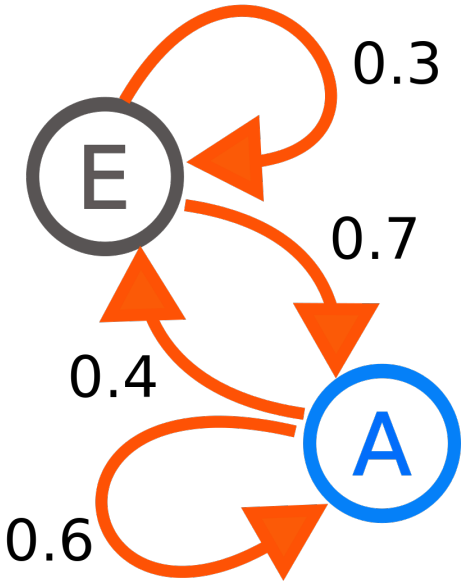

Figura 3.23: Grafo de una cadena de Márkov

3.3.4.2 Proceso de decisión de Márkov

Los procesos de decisión de Markov se utilizar para describir el entorno donde se desarrolla el aprendizaje por refuerzo. Son una extensión de las cadenas que además incluyen las decisiones tomadas por el agente. [29]

Existen tres formas de decidir la política óptima para un problema de decisión secuencial: **Política miope, política de horizonte finito, política de horizinte infinito**.

- Las políticas miopes consideran que el agente solo maximiza la recompensa del siguiente estado.
- En las políticas de horizonte finito se considera un marco temporal concreto sobre la que maximizar la política.
- La política de horizonte infinito considera todos los tiempos futuros, y es en el RL es esta la más utilizada.

3.3.5 Q Learning

El algoritmo de **Q Learning** se encarga de encontrar una política optima en sistemas, de esta forma el sistema se encarga de maximizar la recompensa obtenida durante todos los pasos. Para este proceso el

sistema trata de maximizar la recompensa a largo plazo, lo que se denomina política optima de horizonte infinito.

Para lograr maximizar la recompensa los algoritmos de Q-Learning se enfocan en aumentar el valor de la recompensa total en todos los pasos que se va a realizar (utilidad), empezando desde el estado presente.

Se generan pares denominados Q (estado, acción) en el cual se almacena el estado de la utilidad. En el par se se almacena un sumatorio de todas las posibles recompensas futuras. Este proceso acarrea el problema en sistemas infinitos con recompensas infinitas, ademas de no poder asignar pesos a las recompensas, para priorizar aquellas mas cercanas.

3.3.5.1 On-policy y off-policy

Existen dos tipos de algoritmos en el aprendizaje por refuerzo.

- **On-policy**: Algoritmo que se actualiza utilizando la política actual, a la vez que se redirige hacia una política óptima.
- **off-policy**: El algoritmo es capaz de aprender la política optima independientemente de las acciones de la política actual.[30]

3.3.6 Aprendizaje en Reinforcement learning

En las redes profundas existen un numero elevado de conexiones entre las neuronas, y se conoce proceso de aprendizaje al ajuste de los pesos de esas conexiones, para que la red obtenga la respuesta deseada para las entradas.

Los datos entran por la capa de entrada, y se propaga la información capa a capa hasta la capa de salida. Durante el proceso de aprendizaje esta salida se evalúa y se calcula el error obtenido. El error se trasmite hacia atrás por medio de Backpropagation, de esta forma se va trasmitiendo a cada capa el error cometido. [31]

3.3.7 Deep Q-Networks

Las redes Deep Q-Network (DQN) son algoritmos off-policy basados en Q-learning que se combinan con aprendizaje profundo por refuerzo. Estos algoritmos se utilizan en sistemas que tienen entradas de múltiples dimensiones.

" Este algoritmo utiliza una red neuronal para aproximar la función Q, evitando así utilizar una tabla para representar la misma. En realidad, utiliza dos redes neuronales para estabilizar el proceso de aprendizaje. La primera, la red neuronal principal (main Neural Network), representada por los parámetros υ, se utiliza para estimar los valores-Q del estado s y acción a actuales: Q(s, a; υ). La segunda, la red neuronal objetivo (target Neural Network), parametrizada por υ', tendrá la misma arquitectura que la red principal, pero se usará para aproximar los valores-Q del siguiente estado s´ y la siguiente acción a´. El aprendizaje ocurre en la red principal y no en la red objetivo, la cual se congela (sus parámetros no cambian) durante varias iteraciones (normalmente alrededor de 10000), y después los parámetros de la red principal se copian a la red objetivo, transmitiendo así el aprendizaje de una a otra, haciendo que las estimaciones calculadas por la red objetivo sean más precisas [32]. "

3.3.8 Actor-Critico

Los sistemas que utilizan un algoritmo de Actor-critico en lugar de una única red neuronal, como se ha visto hasta el momento, utiliza dos redes. La primera red llamada **Actor**, se encarga de decidir las acciones a llevar a cabo en base a las observaciones recibidas. Por otra parte el **Critico** es otra red que evalúa las acciones del actor, y actualiza el mismo para lograr la política optima.

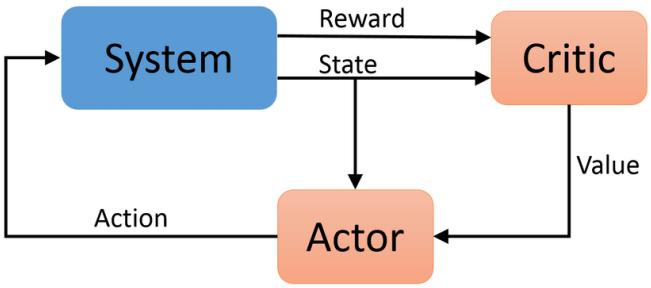

Figura 3.24: Modelo Actor critico

EL algoritmo Actor-critico aportan una importante ventaja, y es que no necesitan una función Q de valor, lo cual lo hace una buena opción para sistemas de acción continua. También plantea un gran problema y es que tanto la recompensa como la evaluación de la misma solo se aporta al finalizar el episodio. Una posible solución es la de aumentar el numero de muestras [33].

Capítulo 4

Toolbox Reinforcement learning

4.1 Introducción

En este apartado se va a realizar una explicación de la características, funciones y opciones que nos ofrece RlTrainingOptions perteneciente a la toolbox de Reinforcement learning. Esta función es la encargada de especificar de que forma se va a realizar el entrenamiento de los agentes.

4.2 Explicación de la toolbox

RlTrainingOptions: Es una función que se encarga de configurar las opciones para el entrenamiento de un Agente previamente creado. Para determinar estas opciones se le pueden pasar una serie de atributos por parámetro. Estas opciones son específicas para un entrenamiento por refuerzo, y configurar algunos aspectos claves que son necesarios para poder realizar el entrenamiento. Para realizar la configuración de estos parámetros necesarios, la función cuenta con unos ajuste predeterminados para cada parámetro. Pero cada uno de ellos puede ser configurado de forma independiente por el usuario, pasándoselo como un argumento a la función, siguiendo la estructura de ('nombre del ajuste ' , valor) [34]. Como ejemplo de la llamada tendríamos.

```
Opciones = rlTrainingOptions
Opciones = rlTrainingOptions (MaxEpisodes, 1000)
```

Siendo el primer ejemplo una llamada con ajustes predeterminados y la segunda llamada una en la que se configura de forma específica el número de episodios y el resto con los parámetros predeterminados.

Los parámetros que esta función configura son los siguientes:

- **MaxEpisodes:** Este ajuste determina el número de episodios que el entrenamiento realizara independiente del resto de criterios de finalización. Siendo un episodio cada una de las iteraciones que el algoritmo de aprendizaje realiza para comprobar la validez de los parámetros de la red. Una vez que el número de episodios que lleva el entrenamiento supere el este valor el entrenamiento finalizará. Este valor debe tratarse de un numero entero y positivo. De forma predeterminada esta función lo configurará con un valor de 500.
- **MaxStepsPerEpisode:** En este ajuste se especifica el número de pasos máximos que se van a realizar en cada episodio. Si durante la ejecución de un episodio no se cumple ninguna condición de finalización, cuando el número de pasos supere este valor el episodio terminará. El valor que

este ajuste debe tomar debe ser de un numero positivo y entero. El valor predeterminado para este parámetro es de 500.

- **ScoreAveragingWindowLength:** Determina la longitud de la ventana para promediar las puntuaciones, las recompensas y el número de pasos de los agentes, que deben ser especificados como vectores o escalares. En este el tipo de dato depende del número de agentes que participan en el entrenamiento. De tal forma que para una simulación con un único agente, se deberá usar un escalar, mientras que para simulaciones multiagente dependerá de si se desea tener la misma longitud para todos los agentes o longitudes personalizadas para cada uno de ellos, de tal forma que para el primer caso se utilizará un escalar y para el segundo un vector.

- **StopTrainingCriteria:** Determina las condiciones en las cuales el entrenamiento debe finalizar. Para determinar cual de las condiciones de parada se desea, se le debe pasar una cadena de texto como valor. La cadena de texto pasada debe coincidir con alguna de las siguientes opciones:

  - AverageSteps: Esta es la opción predeterminada de la configuración. Con esta opción el entrenamiento se detiene cuando el numero promedio de pasos por episodio sea igual al determinado por la opción 'StopTrainingValue'. Y este promedio será calculado por medio de la ventana 'ScoreAveragingWindowLength'.
  - AverageReward: El entrenamiento se detiene cuando el valor medio de la recompensa supera un valor determinado.
  - EpisodeReward: Detiene el entrenamiento cuando la recompensa de un episodio supera un valor determinado.
  - GlobalStepCount: Se detiene el entrenamiento cuando el total de pasos en todos los episodios supere un valor critico.
  - EpisodeCount: Detiene el entrenamiento cuando el numero de episodios alcance un valor.

- **StopTrainigValue:** Es el valor que la condición de parada debe alcanzar para detener el entrenamiento. El valor debe de tratarse de un escalar o un vector. Para entornos con un único agente se debe utilizar un escalar, mientras que para entornos multiagente se utilizara un escalar para dar un único valor a todos los agentes, o un vector para especificar el valor de cada agente por separado. En los entornos multiagente cuando un agente cumple su condición de parada detiene su entrenamiento sin afectar al resto de agentes, los cuales continuaran sus entrenamientos hasta completar sus respectivas condiciones. Su valor por defecto es de 500.

- **SaveAgentCriteria:** Determina cual es la condición para que un agente agente sea guardado. El valor de configuración debe tratarse de una cadena de texto que coincida con alguna de las opciones siguientes:

  - none: No se guarda ningún agente. Esta es la opción por defecto.
  - EpisodeReward: Guarda el estado del agente en un episodio cuya recompensa supera un valor critico.
  - AverageSteps: Guarda el agente cuando el promedio de pasos supera el valor especificado en 'StopTrainingValue'. Este promedio sera calculado por medio de la ventana 'ScoreAveragingWindowLength'.
  - AverageReward: El agente sera guardado cuando el promedio de la recompensa supere un valor critico.

  - GlobalStepCount: Se guarda el agente cuando el numero total de pasos en todos los episodios supera un valor.
  - EpisodeCount: Guarda el agente cuando se llega a un numero de episodios.

- **SaveAgentValue:** Valor que las condiciones de guardado deben cumplir para guardar el agente. Los valores que debe tomar debe ser un escalar para entornos con un único agente o entornos multiagentes donde se utilizar el mismo valor para todos. Por otra parte para entornos multiagente se utilizara un vector. Por defecto el valor es none.

- **SaveAgentDirectoiry:** Se especifica como una cadena de texto, donde se indica la ruta donde se guardaran los agentes guardados.

- **UseParalleles:** Esta opción se estable para ejecutar la simulación del entrenamiento en entornos paralelos, pudiendo así probar diferentes valores a la vez para un mismo agente. Para poder realizar el proceso en paralelo el sistema se vale de los múltiples cores del procesador, o múltiples procesadores derivados de los clusters de ordenadores o la computación en la nube. De esta forma se logra la aceleración por Hardware que disminuye considerablemente los tiempos de entrenamiento. Para esta opción su valor se trata de un booleano, siendo False su opción predeterminada. Para configurar el uso en paralelo se utiliza la opción de ParallelizationOptions.

  Para un entrenamiento en paralelo de DQN, DDPG, TD3 o SAC la propiedad de NumStepsToLookAhead debe estar siempre a 1. debido a que de lo contrario se genera un error. Esto garantiza que se almacene de forma continua los agentes independientemente del valor de StepsUntilDataIsSent.

  Se debe tener en cuenta que la aceleración del los cálculos asociados a las redes neuronales profundas (como son el calculo del gradiente, la actualización de parámetros y la predicción) hace uso de la GPU y no se utiliza UseParallel. Para realizar esta aceleración se utilizará el objeto rlRepresentationOptions donde en la opción UseDevice se debe especificar "gpu". Para poder hacer uso de estas dos tecnologías de aceleración se debe instalar la toolbox de Parallel Computing Toolbox. Mientras que para el uso de clusters de ordenadores o computación en la nube se debe hacer uso de MATLAB Parallel Server.

- **ParallelizationOptions:** Opción que recoge el conjunto de configuraciones para el entrenamiento en paralelo. Esta opción cuenta con un conjunto de propiedades que pueden ser configuradas tras la creación del rlTrainingOptions.

  - Mode: Pude recoger dos valores ”sync” o ”async”
    - sync: Cuando un entrenamiento se realiza de forma síncrona todos los trabajadores que se ejecutan en paralelo se detienen una vez terminan su ejecución hasta que el resto de trabajadores termine. De forma que una vez todos hayan terminado el anfitrión de todos ellos actualiza los parámetros del actor y el criterio en función de todos los resultados obtenidos, para posteriormente devolver estos parámetros actualizados a los trabajadores. Se debe tener en consideración que para paralizaciones basadas en el gradiente, se debe utilizar siempre un entrenamiento síncrono.
    - async: Los trabajadores tras finalizar no esperan al resto, sino que envían sus resultados al anfitrión según terminan y reciben los parámetros del mismo para continuar con el entrenamiento.
  - DataToSendFromWorkers: Es la propiedad que especifica el tipo de datos que los trabajadores enviarán al anfitrión. Pudiendo ser:

    * Experiences: En este modelo de entrenamiento los trabajadores realizan las simulaciones, mientras que el anfitrión se encarga del aprendizaje. De tal forma que los trabajadores simulan las acciones del agente en un entorno recogiendo todos los datos de la experiencia, y mandando los mismos al anfitrión. Cuando los agentes poseen gradiente, es el anfitrión el que se encarga del calculo del gradiente, la actualización del los parámetros de la red, y los envían actualizados a los trabajadores.
    * Gradients: En este modo, tanto la simulación como el aprendizaje son realizados pro los trabajadores. En concreto el trabajador realiza la simulación, calcula el gradiente y envía los resultados al anfitrión, el cual promediará los resultados y actualizara los parámetros de la red para posteriormente enviárselos a los trabajadores. Esto requiere un entrenamiento sincrono.

  – StepsUntilDataIsSent: Se especifica el numero de pasos que el trabajador debe dar antes de enviar los resultados al anfitrión. Este valor debe ser un entero positivo o un -1, siendo -1 la opción predeterminada y que indica que el trabajador esperara a terminar la simulación antes de enviar los resultados.

Capítulo 5

Analisis del problema

En este capítulo se realizara un análisis del problema planteado, de las posibles formas por la cual se puede resolver el problema, y por último una especificación de los requisitos.

5.1 Problema planteado

El problema que se nos plantea consiste en la simulación de un vehículo autónomo que sea capaz de moverse y llegar a un destino, utilizando unicamente la información recibida de un lidar y de si existe colisión.

El comportamiento del vehículo es el mismo que el de un coche cotidiano. El vehículo solo puede desplazarse hacía delante en linea recta, y girar hacia los lados, pero para producirse este giro el coche siempre debe avanzar. Por lo tanto, el vehículo no puede rotar sobre si mismo, ni desplazarse lateralmente. En la siguiente imagen se describe la forma de giro del coche.

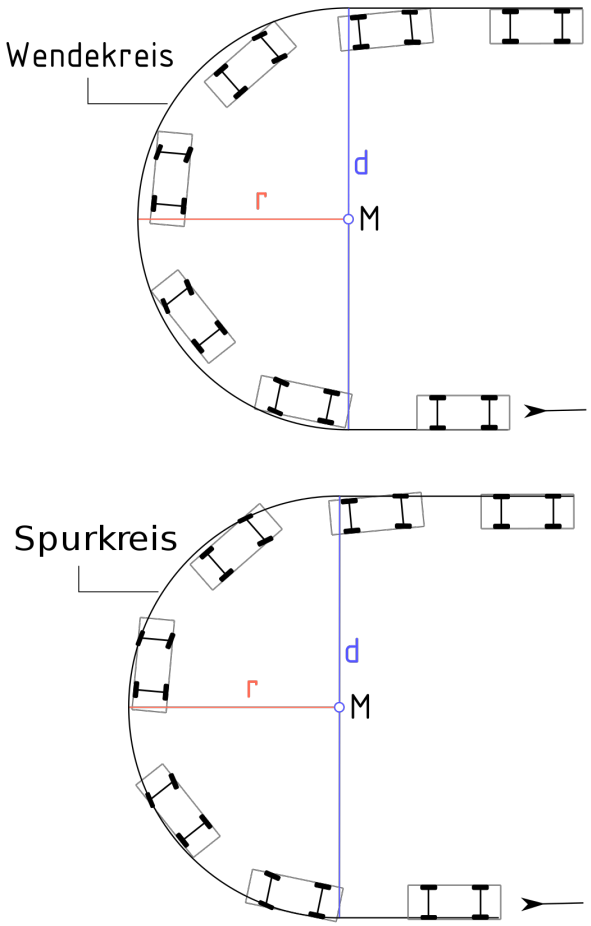

Figura 5.1: diagrama de giro del vehículo

Para que el vehículo gire en una dirección, ambas ruedas deben apuntar en la misma dirección. En la figura 2 se observa el esquema de giro de las ruedas.

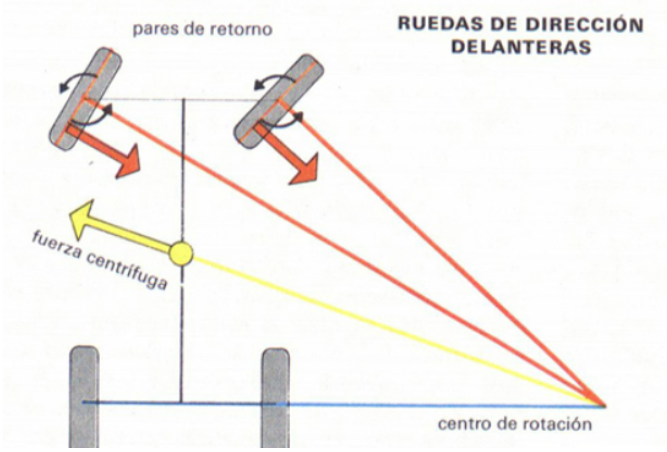

Figura 5.2: Diagrama de ruedas

El vehículo cuenta con un conjunto de dispositivos que le aportan información. Cuenta con un lidar, una cámara, y un detector de colisiones. El LIDAR detecta la distancia existente entre el objeto y el vehículo. El campo de observación es de 360 grados, y con un rango limitado. La cámara se sitúa en la parte

frontal, y solo aporta información de la existencia de un obstáculo en su campo de visión. El detector de colisiones se sitúa alrededor del perímetro del coche, y como su nombre indica, identifica cuando el coche entra en contacto con un objeto.

Este problema debe ser resulto por medio de Aprendizaje por Refuerzo.

Capítulo 6

Desarrollo

A fuerza de construir bien, se llega a buen arquitecto[1].

Aristóteles

6.1 Introducción

Una vez se ha introducido las bases teóricas para el desarrollo de un sistema de aprendizaje por refuerzo, y habiendo expuesto el problema que se plantea, se procede a aplicar dichos conceptos para llevar a acabo la navegación en un entorno con obstáculos mediante la técnica de aprendizaje por refuerzo.

6.2 Identificación de los elementos

Para comenzar con el desarrollo hay que identificar los elementos del aprendizaje por refuerzo que se corresponden con el problema que se nos plantea:

- **Agente**: Sera el controlador del vehículo, y que sera entrenado para ser capaz de tomar las decisiones adecuadas para que el vehículo sortee los obstáculos y llegue al destino final.
- **Entorno**: Será el escenario con obstáculos, y un destino que alcanzar. El entorno tendrá unos limites definidos, con los que el vehículo no tocar, o detectara una colisión.
- **Recompensa**: Su valor depende de la cercanía a la que se quede el agente del objetivo, y de si llega o no; el tiempo que tarde en llegar, y el tiempo que tarda en colisionar. Por otra parte en caso de colisión se produce una penalización.
- **Acciones**: El agente solo puede controlar dos parámetros, la velocidad de desplazamiento del vehículo y el ángulo de giro de las ruedas.
- **Observaciones:** Como información el agente recibe la información de la cámara y el lidar, el detector de colisiones, así como la información del propio vehículo como velocidad y ángulo de giro.
- **Entrenamiento**: Los episodios de un entrenamiento comenzaran con el vehículo, los obstáculos y el destino en una posición fija dentro del entorno. Cada episodio termina cuando se excede el tiempo de entrenamiento, se alcanza el objetivo, o existe una colisión. Y el proceso de entrenamiento concluye tras un numero fijo de episodios o tras alcanzar un valor medio de la recompensa.

[1] Tomado de ejemplos del proyecto TEXIS.

6.3 Herramientas utilizadas

Para el desarrollo de este proyecto se han planteado dos posibilidades. La primera era utilizar Python, el cual consta de múltiples bibliotecas como Tensorflow, que nos facilitaría el desarrollo de proyectos con IA. Debido a sus características, se trata de un lenguaje muy versátil y sencillo de utilizar, el cual nos habría permitido generar todo lo que se necesita para el desarrollo del proyecto. Por otra parte teníamos la opción de Matlab. Esta opción tiene mas limitaciones, pues posee menor versatilidad que la opción de python, pero sigue contando con herramientas para IA, concretamente consta de la toolbox **"Reinforcement Learning Toolbox"**, el cual nos presta las herramientas concretas, que se necesitan para este proyecto. Otra ventaja importante que posee Matlab sobre Python son los ejemplos, pues Matlab nos aportaba un entorno y un vehículo ya diseñados en simulink para la realización de las simulaciones. Este vehículo y entorno suponen una gran ventaja frente a Python en el cual habría que diseñar y desarrollar esto para nuestro problema. A de comentarse que aunque, Matlab nos aporta el entono de simulación y la toolbox, esto nos plantea otro problema, y es que ambos actúan como cajas negras permitiendo pocas o nulas modificaciones en ambas, limitando considerablemente el desarrollo del proyecto.

Tras la exposición de ambas opciones la opción por la que se ha optado es por la de matlab en su versión de 2020 y 2021. El principal motivo que nos lleva a seleccionar esta opción es el entorno de entrenamiento que nos aporta, pues aunque familiarizarse con este y modificarlo para adaptarlo a los requisitos del problema conlleva un tiempo considerable, es muy inferior al tiempo que consumiría su desarrollo completo en Python.

Por otra parte todos las simulaciones se han realizado en dos ordenadores diferentes. El primero de ellos consta de un procesador Intel i7 de 4º generación, con una tarjeta gráfica Nvidia gtx 980m y 32Gb de Ram. El segundo consta de un Intel i7 de 10º generación, con una tarjeta gráfica Nvidia GTX 2070 super y 32Gb de Ram. Se trata de dos equipos con una potencia de calculo considerable, pese a ello los procesos de entrenamiento conllevaran largos periodos de tiempo.

Una vez se determino el entorno en el que se iba a desarrollar el proyecto, se dedico un periodo largo de tiempo a la investigación del funcionamiento de las herramientas seleccionadas. Posteriormente se procede a identificar los dos flujos de diseño de los que consta el proyecto, uno llevado a cabo en **Simulink**, herramienta proporcionada por Matlab que nos permite una programación grafica, modular y entornos de simulación; la otra parte del flujo de desarrollo se realizaría en los propios scripts de **Matlab**.

Para explicar mejor como se separan estos flujos de trabajo, se va a explicar concretamente que partes del desarrollo se realizan en cada uno de ellos.

En Simulink se realizará:

- Implementación del modelo del vehículo.
- Implementacion del agente.
- Bloque de observaciones.
- Bloque de control del vehículo.
- Modelo de Lidar.
- Bloque que calcula las recompensas.
- Conexión entre el Agente y los elementos anteriores.
- Presentación gráfica de entorno y vehículo.

En Matlab de realizará:

- La creación y la definición del entorno.
- Crear las redes neuronales del Agente (Actor-Critico).
- Configuración del Agente.
- Configuración del proceso de entrenamiento.
- Configuración de la representación del entrenamiento.
- Creación de la función local de reset.
- Análisis de los resultados del agente tras el entrenamineto.

6.4 Implementación Del vehículo, agente y entorno en Simulink

6.4.1 Modelo del vehículo

Para este proyecto, se ha escogido como robot móvil un vehículo con una geometría **Ackerman**. En el control inteligente de un robot no e necesario tener las dimensiones exactas de la planta del robot, se puede ver el sistema como una caja negra a la que se le aporta una entrada y se obtiene una salida. El agente se encargara de aprender a utilizar esa caja negra para lograr los objetivos que se le dan.

Para entender mejor el funcionamiento se va a explicar levemente los principales conceptos de la geometría Ackerman. La geometría Ackerman describe los ángulos y los movimientos que realizan las ruedas de un coche. En ella se describe como se comportan las 4 ruedas en la ejecución de una curva. Los robots con geometría ackerman necesitan avanzar para poder efectuar un giro, pues no pueden rotar sobre si mismos.

En la geometría ackerman cuando se realiza un giro, los ejes de rotación de todas las ruedas deben concurrir en un único punto, llamado centro instantáneo de rotación.

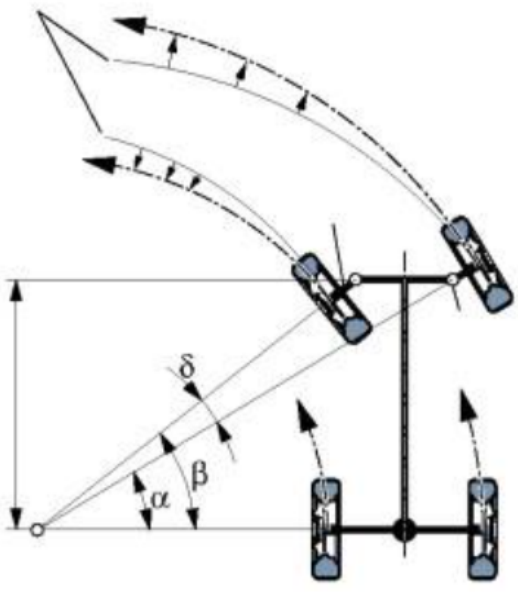

Figura 6.1: comparación en el aprendizaje

El modelo del vehículo se ha simulado a través de simulink por medio de una serie de bloques sencillos. Cada bloque representa una parte del funcionamiento del vehículo con una lógica propia.

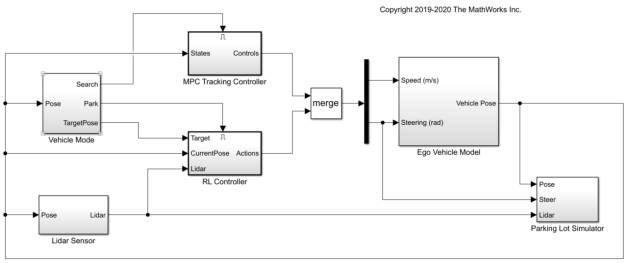

Figura 6.2: Modelo en Simulink del agente y vehículo

6.4.1.1 Bloque Vehicle model

El primer bloque que se utiliza es **Vehicle model**, este bloque simula el funcionamiento del vehículo con una cámara frontal, cuya función es la de la localización del objetivo. El rango de esta cámara es limitado, y solo localiza el objetivo a una corta distancia del mismo. La lógica de este bloque consiste en comprobar los objetos que la cámara tiene a su alcance, cuando detecta un bloque que se corresponde a un destino, establece la posición de este objeto como destino. Por otra parte también evalúa si se ha alcanzado el objetivo marcado, o por si el contrario se debe continuar.

Este bloque tiene como entrada el parámetro **Pose**, esta variable contiene la posición del vehículo en el mapa. Como Salida dispone de tres elementos: **Search, Park, targetPose**. Los parámetros Search y Park son dos booleanos contrarios, es decir, que uno es la negación del otro, se utilizan para indicar si se tiene que seguir buscando la solución o ya ha logrado alcanzar el destino. Por ultimo targetPose indica la posición donde está el objetivo.

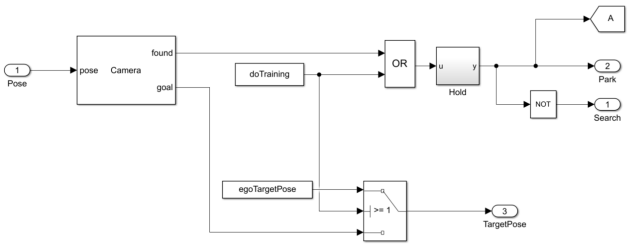

Figura 6.3: Bloque simulador del vehículo

6.4.1.2 Bloque Lidar Sensor

Es el segundo bloque utilizado es el **Lidar Sensor** que se encarga de simular el lidar que posee el vehículo para poder detectar las distancias existentes a los obstáculos. Al igual que pasa con la cámara su rango de alcance es limitado, aunque su campo de visión es de 360 grados, con un numero pequeño de haces. En este bloque se recoge la posición en el mapa del coche y se hace una transformada de su posición y su angulo, para poder generar la información de los haces, observando los alrededores en el mapa.

Como entradas a este bloque se tiene **Pose**, que indica la posición en el mapa del robot. Y como salida del mismo tenemos **Lidar** que contiene la información relativa al lidar.

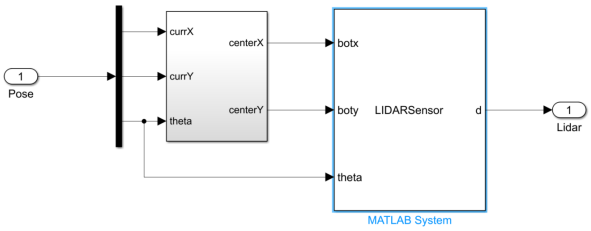

Figura 6.4: Bloque simulador del lidar

En este elemento encontramos el bloque **LIDARSensor** que mediante la información transformada de la posición del vehículo, su angulo y el mapa, es el encargado de realizar un array de información que representa al lidar. Este es un bloque propio de Matlab. Para poder realizar correctamente el proceso de generación del lidar se le debe aportar una información inicial. **Map** con la información del entorno de la simulación, **map.VehicleDimensions** contiene las medidas des coche, **maxLidarDist** indica cuanta longitud tienen los haces del lidar para poder detectar obstáculos, **numSensors** es el numero de haces que componen al lidar.

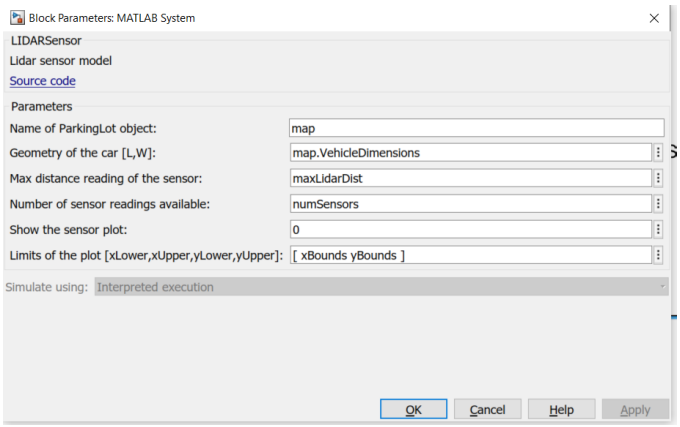

Figura 6.5: Bloque LIDARSensor

6.4.1.3 Bloque MPC Tracking Controller

El bloque **MPC Tracking controller** se utiliza dentro de los vehículos autónomos, se trata de un controlador de seguimiento de ruta, que utiliza el método de control predictivo del modelo (MPC). En este caso se utiliza un modelo propio de las bibliotecas de Matlab que nos proporciona dos modelos de MPC: Adaptive MPC y Nonlinear MPC. La principal diferencia entre ambos es que mientras Adaptive es invariable en el tiempo, el Nonlinear si puede serlo, por lo que para este proyecto se ha optado por Adaptive. Este bloque debe recibir la posición del vehículo en el entorno, y una seña de activación, que mientras se encuentre activa el bloque estará activado. Como salida del mismo tenemos un modelo de control, para las acciones a realizar.

6.4.1.4 Bloque Ego Vehicle Model

El bloque **Bloque Ego Vehicle Model** se encarga de actualizar la posición del vehículo en las coordenadas del entorno, así como de actualizar la pose interna del vehículo que se utiliza para validar cierta calidad en las medidas obtenidas. Este elemento dispone de dos entradas y una salida. Como entrada

disponemos de una velocidad lineal en m/s y un angulo de giro en radianes, que indica el arco de giro que debe describir el vehículo. Como salida del sistema tenemos la posición actualizada del vehículo en el entorno.

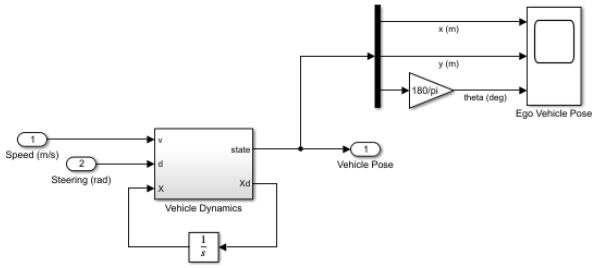

Figura 6.6: Modelo de posicionamiento del vehículo

6.4.2 Modelo de Reinforcement learning

Para la implementación del modelo encargado de la toma de decisiones del vehículo se ha desarrollado el bloque **RL Controller** el cual se compone de diferentes elementos para el desempeño de sus funcionalidades. Dentro de este bloque se encuentra el Agente, del cual ya se hablo en los apartados de teoría, el cual es el encargado de la toma de decisiones y del proceso de aprendizaje por medio de la recompensa obtenida ante sus acciones. Este bloque necesita tres entradas y una señal de activación. Estas entradas deben contener la posición del vehículo en el entorno, el array de señales del lidar con la información de los obstáculos cercanos, y la posición del objetivo que se ha detectado. La señal de entrada sirve para reiniciar los valores una vez se ha logrado el objetivo. Como salida de este se tiene las acciones que deben ser tomadas por el vehículo. La lógica de este bloque la componen 2 bloques: **Observation and Reward** y **RL Agent**. ademas de estos bloques se hace una transformación de la salida de los mimos a una señal que pueda ser operada por el vehículo, y por último también existe una pequeña protección que limita la velocidad máxima del coche. Todo esto da como resultado las acciones del vehículo.

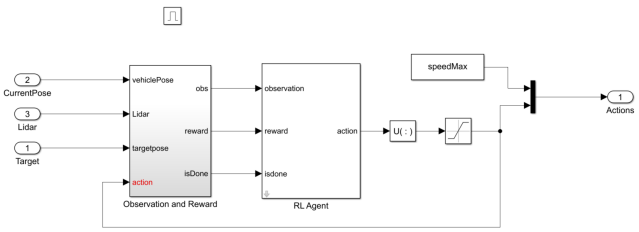

Figura 6.7: Controlador de Reinforcement Learning

6.4.2.1 Modulo Observation and Reward

En el modulo **Observation and Reward** se procesa información como la posición del coche, el lidar, la posición del objetivo, y las acciones realizadas. Con toda la información que se procesa, se determina si el proceso de acciones se ha terminado, y lo más importante se realiza el calculo de la recompensa. Las entradas al modulo, son las ya mencionadas anteriormente: vehiclePose (Posición del vehículo),

lidar, targetpose (posición del destino), action (la acción realizada para esa situación). Como salida, tenemos: isDone (determina si se ha llegado al final del proceso), obs (Conjunto de observaciones), reward (recompensa).

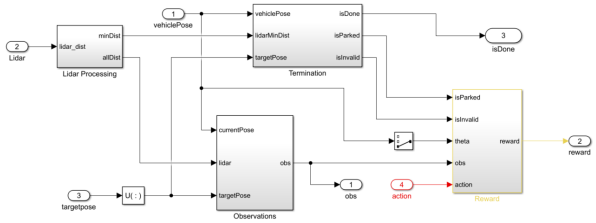

Figura 6.8: Modulo Observation and Reward

El modulo Observation and Reward esta compuesto por una serie de módulos que se encarga de realizar las operaciones a las entradas y así determinar el calculo de la recompensa adecuada. Estos módulos son los siguientes.

- **Lidar Processing**: Este modulo recoge toda la información proveniente del lidar, y dar como resultado dos salidas. Una de ellas es la información del lidar sin modificar, y la segunda es aplicando una resta y extrayendo los min.

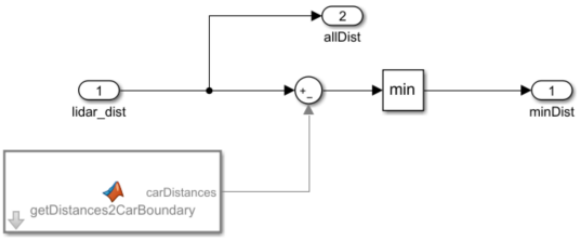

Figura 6.9: Modulo Lidar Processing

- **Termination:** En este módulo se determina si se ha logrado el objetivo o existe una colisión o una posición incorrecta. De esta forma se indica si el episodio del entrenamiento se da por terminado, y si se logro el objetivo o no.

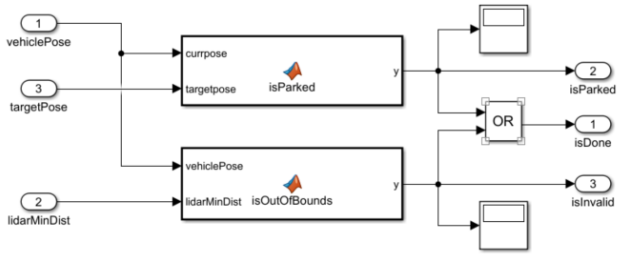

Figura 6.10: Modulo Termination

- **Observations:** Transforma la información de las diferentes fuentes de entrada para que compartan el mismo tipo y complejidad y así generar una única fuente de información con toda la información necesaria del entorno. Para lograr esa transformación, se extrae la información en X e Y de las posiciones del vehículo y del destino, para encontrar la distancia existente entre ellas, y posteriormente convertir el error en un vector de una dimensión y dividir entre 10 este valor. También se realiza el coseno y seno de del angulo de vehículo. Por ultimo recoge toda la información del lidar, la transforma en un vector y la divide por la distancia máxima que puede alcanzar el lidar. recoge todas las salidas y genera un vector unidimensional con ellas.

$$obs = [\frac{X_{destino} - X_{coche}}{10}, \frac{Y_{destino} - Y_{coche}}{10}, \sin\theta_{coche}, \cos\theta_{coche}, \frac{[L^1, L^2, ..., L^n]}{maxLidarDist}] \tag{6.1}$$

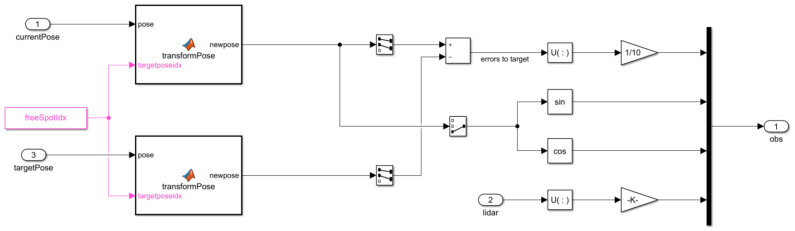

Figura 6.11: Modulo Observations

- **Reward**: En este módulo se recoge toda la información procesada en los módulos anteriores para calcular la recompensa que se ha obtenido en el episodio actual.

$$Reward = 2e^{-(0{,}05X_e^2 + 0{,}05Y_e^2)} + 100\rho - 50\iota \tag{6.2}$$

  - Reward: Recompensa que suministrada al agente.
  - X_e: Dimensión X del error, que se obtiene de la diferencia existente entre la posición del vehiculo y la posición del destino.
  - Y_e: Dimensión Y del error, que se obtiene de la diferencia existente entre la posición del vehiculo y la posición del destino.
  - ρ: Representa si se ha alcanzado el destino, sus valores pueden ser 0 o 1.
  - ι: Representa si se ha llegado un punto invalido, sus valores pueden ser 0 o 1.

6.4.2.2 Modulo RL Agente

El modulo **RL Agente** es un bloque creado en simulink que proporciona Matlab a través de la toolbox Reinforcement Learning Toolboxz se encarga de realizar las tareas del agente diseñado.

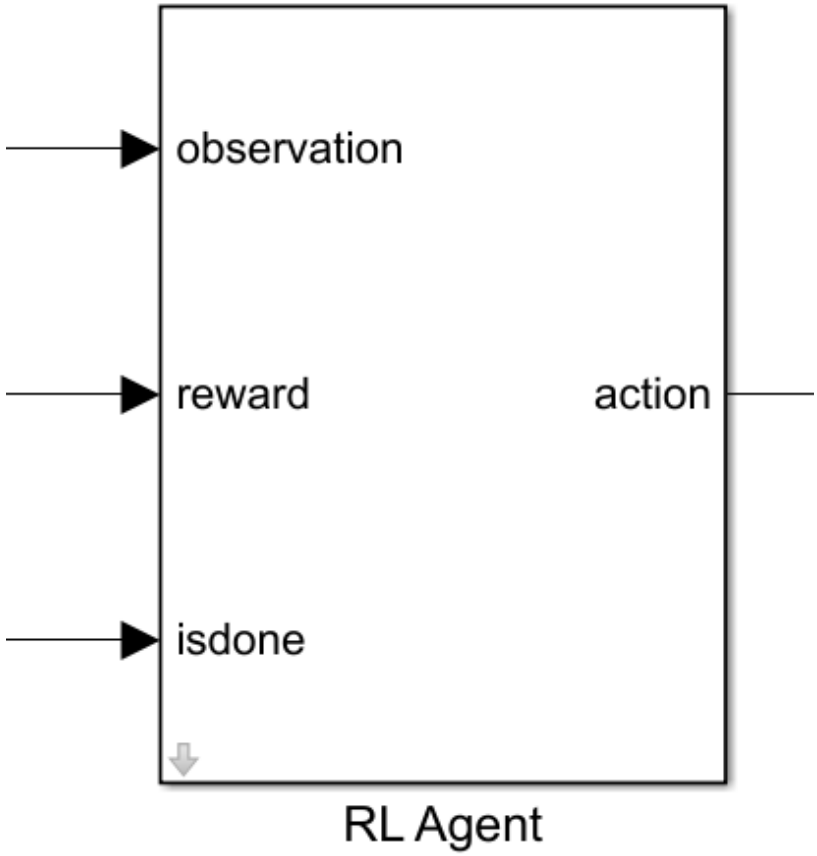

Figura 6.12: Modulo RL Agente

El bloque dispone de tres entradas a su sistema que son las siguientes **observation**, **reward** e **isdone**. En estas entradas se encuentra toda la información del entorno que el agente necesita para desempeñar su función.

- **Observation**: Contiene la información del entorno aportada por el lidar y la posición y angulo en el mapa del vehículo.
- **Reward**: Es la recompensa que el agente recibe por las acciones realizadas con determinada observación.
- **Isdone**: Esta entrada indica al agente que el proceso de alcanzar el objetivo ha finalizado, ya sea por si ha logrado alcanzar el objetivo marcado, o si por otro lado el coche a colisionado con algún obstáculo o se ha salido de los limites del entorno marcado.

Como salida del bloque disponemos de la acción o el conjunto de acciones que el agente decide ejecutar basándose en las observaciones recibidas.

Dentro de simulink el único parámetro que se nos permite modificar es el nombre del agente. Todas las características que describen el funcionamiento del agente se especifican en el código de Matlab, por medio de la utilización de una RNA y el metodo de Actor Critico. Debido a esto es necesario que el nombre que se le de al agente en simulink este asociado a la descripción del mismo en el código. Esta asignación del nombre se realiza dentro de la configuración del agente en el bloque de simulink, en el apartado **Agent object**.

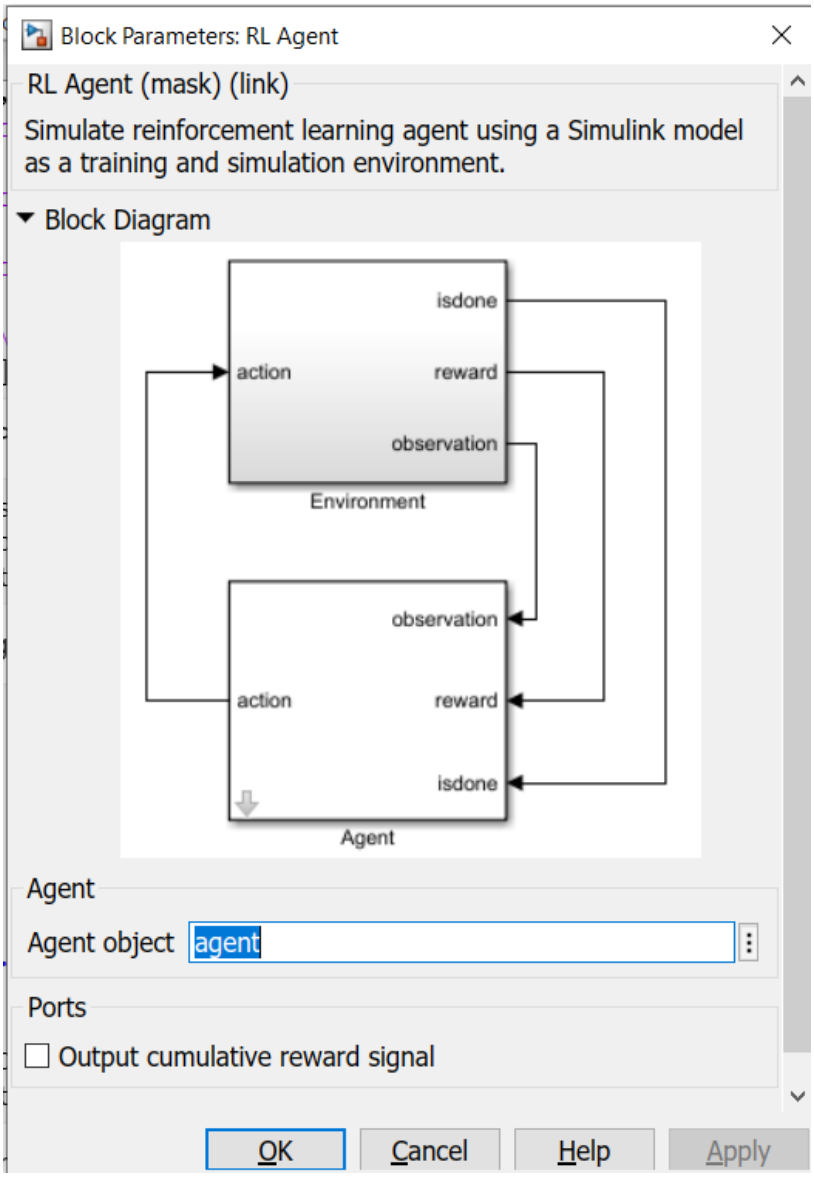

Figura 6.13: Modulo RL Agente

6.5 Creación del entorno

Para la creación del entorno necesario para que se desempeñen las observaciones y las acciones se debe suministras las información acerca de las mismas. Para aportar dicha información se utilizan dos funciones: **rlNumericSpec**, que especifica una observación o acción continua, y **rlFiniteSetSpec**, que especifica una observación u acción descontinua.

En el caso de este proyecto se ha optado por determinar que las observaciones y las acciones sean continuas,

para que se parezca al modo de actuar de un ser humano. Por tanto la función utilizada es rlNumericSpec.

Al momento de determinar el entorno, estas funciones nos permiten determinar el rango de las acciones y las observaciones. Al limitar estos valores se consigue que en el momento en que el agente busque una política óptima no se exploren valores fuera de los rangos fijados. Para este caso, se limitan las observaciones y las acciones. Dentro de las acciones se limita tanto la velocidad mínima como la máxima, y el angulo de giro para evitar movimientos bruscos. En el caso de la velocidad se ha limitado entre **-2** y **6 (m/s)**, y para la velocidad de giro, se ha limitado entre **-π/6** y **π/6 (rad/s)**. La elección de estos valore es personal del programador no existiendo unos valores correctos. Hay que tener en cuenta que cuanto mas cerca del 0 se encuentren los rangos, mas despacio se moverá el vehículo, alargando de este modo los entrenamientos.

```
numObservations = 34 ;
observationInfo = rlNumericSpec([numObservations 1]);
observationInfo.Name = 'observations';

steerMax = pi/4;
discreteSteerAngles = -steerMax : deg2rad(15) : steerMax;
actionInfo = rlFiniteSetSpec(num2cell(discreteSteerAngles));
actionInfo.Name = 'actions';
numActions = numel(actionInfo.Elements);
```

Con toda la información de las observaciones y las acciones, se puede crear el entorno. La creación del mismo se hace por medio de la función **RLSimulinkEnv**. La función rlSimulinkEnv vincula el entorno creado con los modelos de simulink y con el agente del modelo del mismo simulink. Tras enlazar estos componentes con el entorno creado se le debe especificar al entorno una función de reset.

```
mdl = 'rlAutoParkingValet';
open_system(mdl)
blk = [mdl '/RL Controller/RL Agent'];
env = rlSimulinkEnv(mdl,blk,observationInfo,actionInfo);
env.ResetFcn = @rest;
```

6.5.1 Problemas en la creación del entorno

Durante el desarrollo del proyecto se intento implementar una aleatoriedad en la generación del entorno. Pero se presentaron dos problemas. El primero es, que al proceso de entrenamiento se le suministra ya un entorno creado, que no se ve modificado posteriormente, por lo que la aleatoriedad en este caso solo se podría realizar entre entrenamientos . El segundo problemas surgió en la propia implementación, pues al no definirse de una forma concreta las posiciones de los obstáculos, en el momento de generar la caja de colisión de estos objetos, no se lograba hacer que coincidieran. Para poder intentar solucionar este problema había que profundizar en el funcionamiento interno de algunas funciones de Matlab. Debido a esta situación se determino que el coste en tiempo para la solución del problema era mayor al beneficio obtenido con la aleatoriedad. Por esta razón se descarto esta funcionalidad.

6.6 Función Reset

La función de reset se encarga de reiniciar el entorno a unos valores iniciales, fijos o aleatorios. El reinicio del entorno se realiza al finalizar cada episodio, se haya logrado el objetivo o no. En el caso que nos

acontece se ha diseñado un modelo que combina valores fijos y aleatorios, de forma que, en unas etapas tempranas del modelo, el vehículo se inicie siempre en el mismo punto del mapa, y con posibilidad de que en un futuro, el la aleatoriedad de la iniciación se mueva en torno a esa posición inicial, y en una ultima estancia este inicio pueda abarcar todo el entorno de entrenamiento.

```
function in = rest (in) % autoParkingValetResetFcn(in)

        choice = rand;
        if choice <= 0.35
                x = 95 - rand *10;
                y = 50 - rand * 5 ;
                t = deg2rad(-45 + 2*45*rand);
        elseif choice <= 0.70
                x = 95 + rand *10;
                y = 50 + rand * 5 ;
                t = deg2rad(-225 + 2*45*rand);
        else
                zone = randperm(3,1);
                switch zone
                        case 1
                                x = 36.5 + (45.5-36.5)*rand;
                                t = deg2rad(-45 + 2*45*rand);
                        case 2
                                x = 45.5 + (50-45.5)*rand;
                                t = deg2rad(-135 + 2*45*rand);
                        case 3
                                x = 50 + (59-50)*rand;
                                t = deg2rad(-225 + 2*45*rand);
                                if t <= -pi
                                        t = 2*pi + t;
                                end
                        end
                        y = 10 + (20-10)*rand;
        end


        pose = [x,y,t];
        pose = [95 55 pi];
        %speed = 5 * rand;
        speed = 1000 * rand;
        in = setVariable(in,'egoInitialPose',pose);
        in = setVariable(in,'egoInitialSpeed',speed);
end
```

6.7 Creación del Agente

La creación de un agente en Matlab requiere de una serie de procesos que se detallarán en este apartado. Para la creación del agente se hace uso de una red neuronal profunda que utiliza aprendizaje por refuerzo. Como ya se explicó en el apartado teórico de Actor-crítico es necesario generar dos redes neuronales

diferentes.

6.7.1 Creación del Crítico

Como se explico anteriormente, la red neuronal del critico recibe como entrada las observaciones y acciones que se han realizado. Como salida de la red se aporta el error obtenido, que servirá para corregir su funcionamiento.

Para la generación de la red se tienen que definir 5 características de su geometría, que son: numero de capas ocultas, neuronas en las capas ocultas, numero de parámetros de entrada, numero de parámetros de salida y función de activación de la capa. En la red del crítico se ha definido una red de 4 capas, con 16 parámetros de entrada y 1 de salida, y con la función de activación ReLu. En las capas ocultas se han probado diferentes configuraciones de numero de neuronas. La combinación que mejor resultado nos ha dado es con un descenso en el numero de neuronas, con 512, 256, y 128 en la ultima capa culta.

```
criticNetwork = [
featureInputLayer(numObservations,'Normalization','none','Name','observations')
fullyConnectedLayer(512,'Name','fc1')
reluLayer('Name','relu1')
fullyConnectedLayer(256,'Name','fc2')
reluLayer('Name','relu2')
fullyConnectedLayer(128,'Name','fc3')
reluLayer('Name','relu3')
fullyConnectedLayer(1,'Name','fc4')];
```

En la capa de entrada se utiliza la función **featureInputLayer** para normalizar los parámetros de entrada de los que se disponen. Estos datos son el lidar, con sus 12 sensores, la posición en X e Y y por ultimo el $sin(\alpha)ycos(\alpha)$ haciendo un total de 16 entradas. Con la función **fullyConnectedLayer** se generan las capas de forma totalmente conectadas entre si. Por ultimo queda especificar la función de activación que se hace por medio de la función **reluLayer**, como se explica en el apartado teórico gracias a esta función se trabaja unicamente con números positivos.

Una vez que se ha especificado la geometría de la red, se establecen algunas opciones para el Crítico por medio de la función **rlRepresentationOptions**. Se modifican los siguientes parámetros:

- **LearnRate** define la tasa de aprendizaje, que es el tamaño con el que se realiza la variación entre las pruebas, cuanto mayor es el numero mas rápido puede aprender, pero también puede diferir mas sin llegar a converger y no encontrar una politica optima. Por el contrario si la tasa de aprendizaje es pequeña, el aprendizaje es más lento y converge con mayor facilidad. Para este proyecto se han probado diferentes rátios, oscilando entre $1e^{-}3y1e^{-}5$ sin que se llegue a observar una diferencia entre ellas en el entrenamiento.
- **GradientThreshold** esta opción establece un limite en el gradiente del crítico, y así actuar en función de superar ese limite. Este gradiente determina cuanto cambian los parámetros de las red durante cada episodio del entrenamiento.
- **UseDevice** especifica que para el entrenamiento del crítico se debe utilizar las tecnología de la GPU.

Con la función **rlValueRepresentation** utilizando la arquitectura de la red descrita anteriormente, junto con la información de las entradas, y las opciones especificadas, se genera la red neuronal del Crítico.

```
criticOptions = rlRepresentationOptions('LearnRate',1e-3,...
'GradientThreshold',1,'UseDevice',"gpu");
critic = rlValueRepresentation(criticNetwork,observationInfo,...
'Observation',{'observations'},criticOptions);
```

6.7.2 Creación del Actor

Como se explica en el apartado teórico el Actor, es el encargado de tomar las decisiones o las acciones, en función de las observaciones. Para el desempeño de estas funciones, se genera una red neuronal con el mimos proceso que se utilizó para generar el Crítico.

La arquitectura de la red es la misma que la del Crítico ha excepción de la última capa donde el numero de capas depende de los parámetros requeridos para las acciones. Por ultimo a los datos que salen de la red se le aplica la función de activación softmax, que ya se explicó en el apartado teórico.

```
actorNetwork = [
featureInputLayer(numObservations,'Normalization','none','Name','observations')
fullyConnectedLayer(100,'Name','fc1')
reluLayer('Name','relu1')
fullyConnectedLayer(200,'Name','fc2')
reluLayer('Name','relu2')
fullyConnectedLayer(100,'Name','fc3')
reluLayer('Name','relu3')
fullyConnectedLayer(numActions, 'Name', 'out')
softmaxLayer('Name','actionProb')];

actorOptions = rlRepresentationOptions('LearnRate',1e-3,...
'GradientThreshold',1,'UseDevice',"gpu");
actor = rlStochasticActorRepresentation(actorNetwork,observationInfo,...
actionInfo,'Observation',{'observations'},actorOptions);
```

6.7.3 Opciones del Agente

A la hora de crear un agente se le pueden configurar una serie de parámetros que definen como se va a realizar su entrenamiento. Esta configuración se realiza a traves de la función **rlPPOAgentOptions**.

- **SampleTime** establece el tiempo de muestreo que tiene el agente, que indica cada cuanto entra en acción el agente. Para este proyecto se le ha especificado un valor de 0.1
- **ExperienceHorizon** indica cuantos pasos realiza el agente antes de aprender.
- **ClipFactor** limita los cambios producidos en la política en cada episodio del aprendizaje.
- **EntropyLossWeight** añade un factor de aleatoriedad que penaliza a un agente cuando varía poco su política durante un periodo de tiempo. Obliga al agente a realizar un cambio, de este modo se evitan máximos locales.
- **MiniBatchSize** especifica el tamaño de los lotes para el aprendizaje.

```
agentOpts = rlPPOAgentOptions(...
'SampleTime',Ts,...
'ExperienceHorizon',200,...
'ClipFactor',0.2,...
'EntropyLossWeight',0.05,...
'MiniBatchSize',64,...
'NumEpoch',5,...
'AdvantageEstimateMethod',"gae",...
'GAEFactor',0.95,...
'DiscountFactor',0.998);
```

6.7.4 Creación del Agente

El último paso del proceso es la creación del objeto del agente por medio de la función **rlPPOAgent** que utiliza los elementos descritos anteriormente. Es te agente esta ligado a su homónimo de simulink.

```
agent = rlPPOAgent(actor,critic,agentOpts);
```

6.8 Configuración del entrenamiento

Para finalizar con el proceso procede con el entrenamiento del agente creado en el entorno descrito. Pero antes del entrenamiento se debe establecer unas configuraciones para el mismo. Esta configuración se hace por medio de la función **rlTrainingOptions** que se explico en el apartado teórico.

A continuación se va a explicar la razón por la que se han seleccionado los valores:

- **MaxEpisodes**: Este parámetro no tiene un valor critico, aunque si se recomienda dar un valor elevado para que el entrenamiento tenga tiempo suficiente. Durante la experimentación se ha comprobado que con un valor de 10000 es mas que suficiente para que se logre el objetivo. Aunque dependiendo de la complejidad del escenario puede que sea necesario más episodios. La duración de cada episodio se determina desde el reset del vehículo situándose en la casilla de salida, y dura hasta que llega al destino o exista una colisión.

- **MaxStepsPerEpisode**: Debido a la complejidad que puede alcanzar los escenarios, es necesario dejar largos periodos de tiempo, debido al largo recorrido que el vehículo debe hacer para alcanzar el destino.

- **StopTrainingCriteria y StopTrainingValue**: Se ha decidido que cuando la recompensa alcance un valor medio se pueda dar por terminado el entrenamiento. Se ha jugado con el valor que debe alcanzar esta media, y se ha tomado una decisión final de que tenga un valor de 90.

- **SaveAgentCriteria y SaveAgentValue**: Guradaremos aquellos agentes con buenos desempeños, pero se ha optado por no guardar solo los que alcancen el objetivo, sino aquellos que realicen el mejor de los desempeños

Como ultimo paso se ejecuta el entrenamiento con la función **train**.

```
trainOpts = rlTrainingOptions(...
'MaxEpisodes',10000,...
'MaxStepsPerEpisode',1500,...
'ScoreAveragingWindowLength',1000,...
'Plots','training-progress',...
'StopTrainingCriteria','AverageReward',...
'StopTrainingValue',90, ...
'SaveAgentCriteria','EpisodeReward', ...
'SaveAgentValue',120, ...
'SaveAgentDirectory', pwd + "\run1\Agents", ...
'UseParallel',true);
trainingStats = train(agent,env,trainOpts);
```

Capítulo 7

Procesos de pruebas y entrenamientos

Una vez se ha completado el desarrollo del entorno y el agente, se comienza con el proceso experimental de los entrenamientos. A través de un proceso de prueba y error, se prueban diferentes arquitecturas de red y opciones del agente y del entrenamiento, para encontrar aquellas que pueden lograr un correcto desempeño. Para lograr la correcta política se expondrá al vehículo a diferentes entornos y diferentes posiciones iniciales.

Durante la ejecución del entrenamiento se nos presentan abren dos ventanas. La primera, es el mapa sobre el que se va a realizar el entrenamiento. La segunda es la gráfica de las recompensas obtenidas en cada episodio. En la primera ventana, siempre que la opción de ejecutar el entrenamiento en paralelo este desactivada, se podrá ver como se desenvuelve el agente en el entorno.

La siguiente imagen es la gráfica de entrenamiento donde se representa el valor de la recompensa obtenida por el agente en función del episodio de entrenamiento.

Dentro de estas se ven representados tres parámetros en diferentes colores. El valor de la recompensa de un episodio esta representada con un circulo de color azul, mientras que con los asteriscos naranjas se indica el valor medio de las recompensas en ese punto. Por ultimo tenemos las X amarillas, que indican el episodio Q0, es una estimación que realiza el Crítico del valor que tendrá la recompensa a largo plazo. Como información adicional la ventana dispone con información del ultimo episodio.

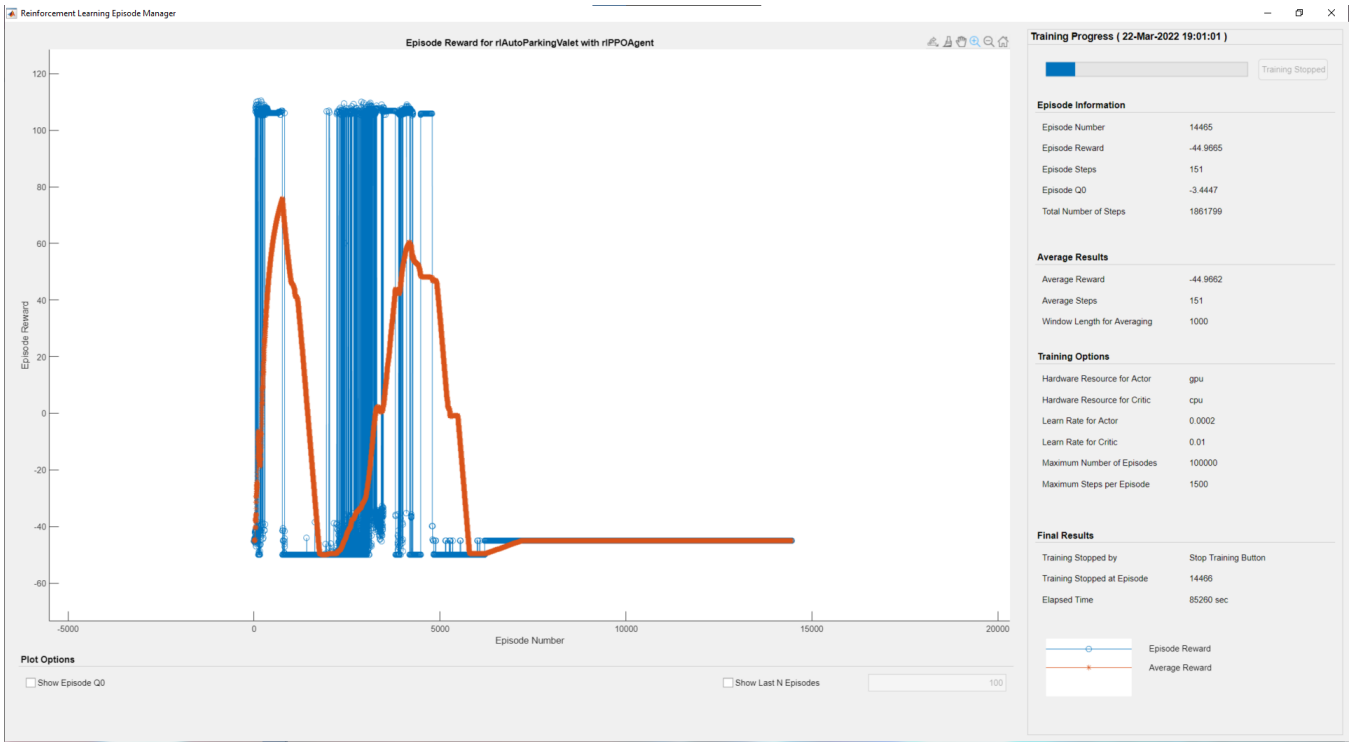

Figura 7.1: Gráfica de entrenamiento

7.1 Primer entrenamiento en un escenario sencillo

Para esta prueba se ha seleccionado un mapa con pocos obstáculos, donde el objetivo se encuentra en horizontal al punto de salida. Las posiciones de salida del vehículo se encuentran en torno a las coordenadas X=90 e Y=50. En este escenario el coche no puede avanzar solo en linea recta al encontrarse un obstáculo en su camino, por lo que debe realizar una ligera curva para esquivarlo.

Las redes de Crítico y de Actor tienen la misma arquitectura, formada por: 16 datos de entrada, 4 capas ocultas con un numero de neuronas asimétrico siendo 512 ->256 ->128 desde la capa de entrada hasta la de salida. Por otra parte al Crítico se le a aportado un learnRate de $1e^{-3}$ mientras que al actor uno de $2e^{-4}$

Tabla 7.1: Configuración de la red para el entrenamiento 1.

Numero de capas	Nueronas por capa	EntropyLossWeight	LearnRate
4	512,256,128,2	0,05	$2e^{-4}$

El mapa sobre el que se va a realizar el entrenamiento es el siguiente.

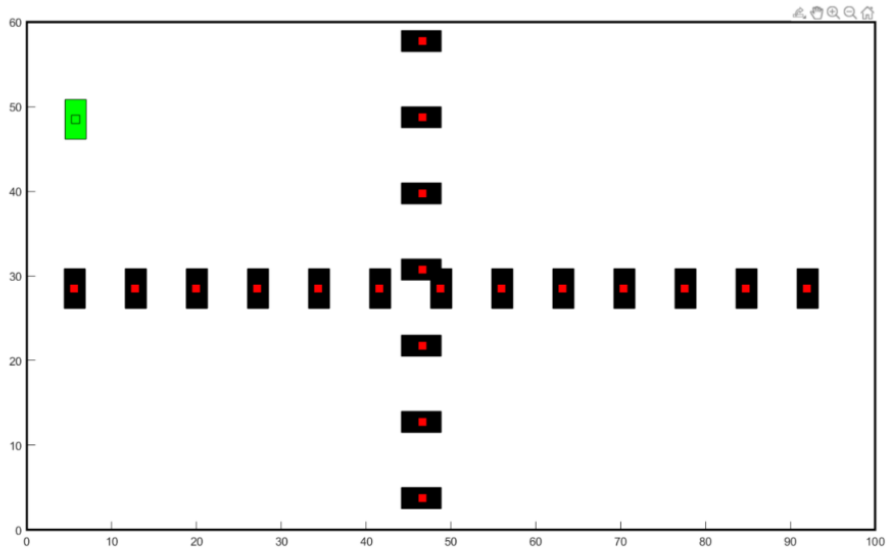

Figura 7.2: Escenario del entrenamiento 1

El entrenamiento se ha realizado en dos ocasiones, una utilizando paralelización y otra sin paralelización.

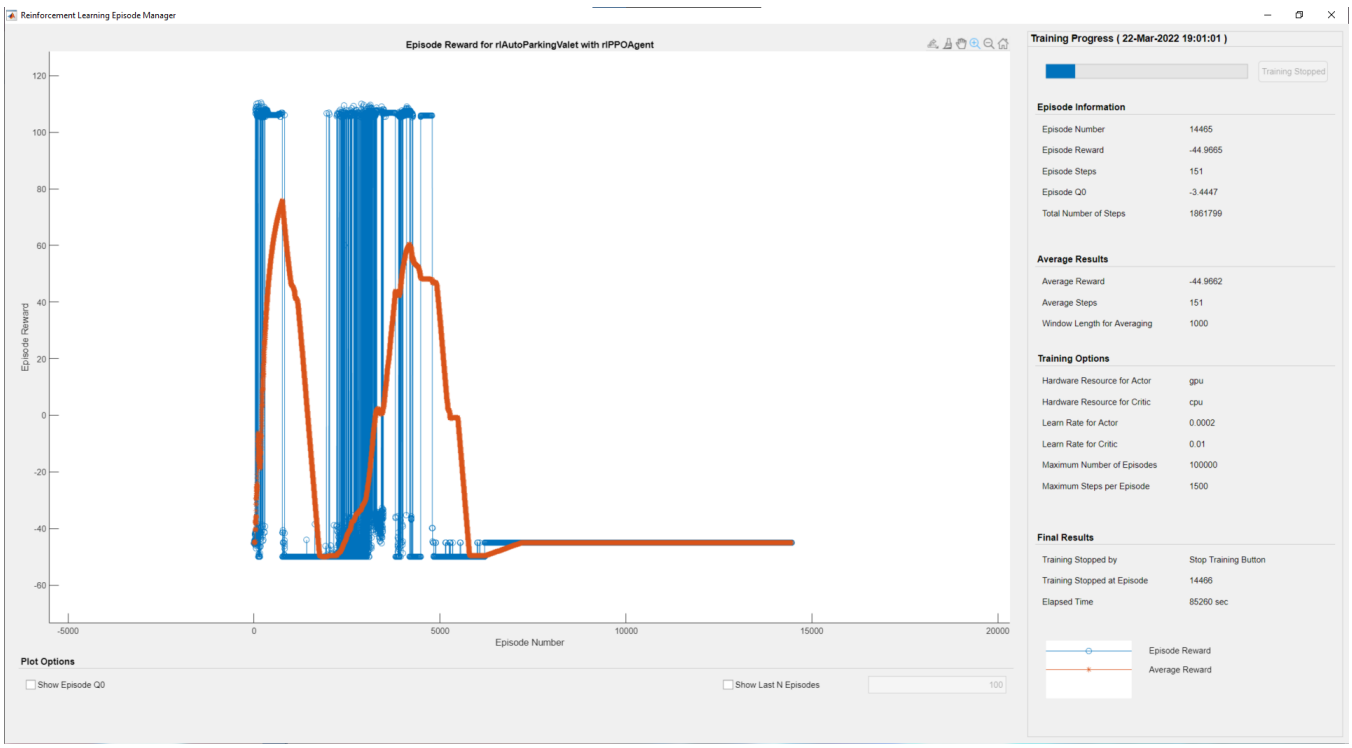

Figura 7.3: Gráfica del entrenamiento con paralelización

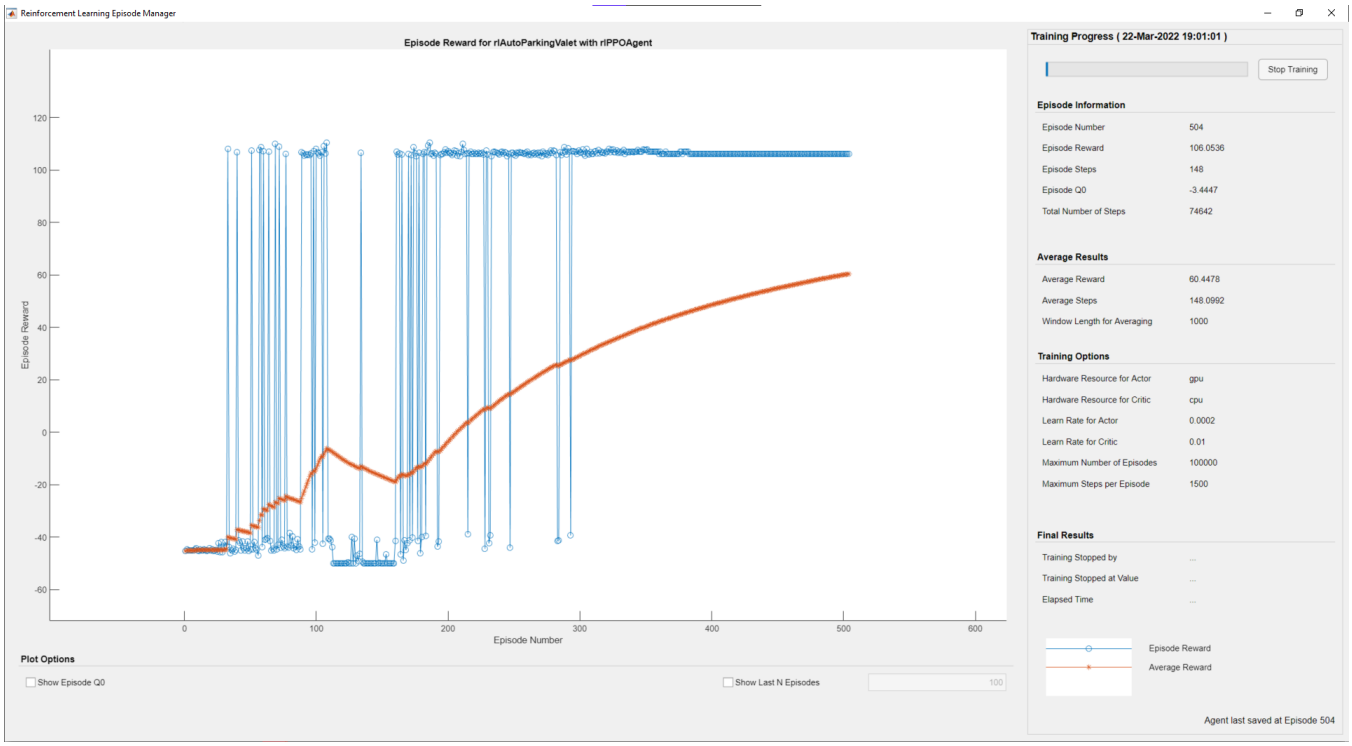

Figura 7.4: Gráfica del entrenamiento sin paralelización

En las siguientes gráficas se observa una diferencia entre la ejecución en paralelo de los sistemas o la ejecución secuencial. El entrenamiento que posee paralelización se observa que el proceso alcanza en dos ocasiones una política optima que logra alcanzar el destino, pero tras unas serie de ejecuciones debido al factor de aleatoriedad pierde la configuración. Por ultimo se observa que localiza un máximo local en -50 que acaba tomando como valido, sin probar nuevas variaciones y dando como resultado un sistema incorrecto.

Por el contrario observamos que en la ejecución secuencial se logra una política correcta. Debido a que la ejecución en paralelo se detuvo el numero de episodios que tuvo el entrenamiento secuencial fue muy limitado como para poder ser valorada.

7.2 Entrenamiento reduciendo el tamaño de la red

Para esta prueba se ha realizado una modificación en la arquitectura de la red para comprobar su variación. Se mantiene el mapa del entrenamiento 1 y no se realizan variaciones en la función de recompensa.

Tabla 7.2: Configuración de la red para el entrenamiento 2.

Numero de capas	Nueronas por capa	EntropyLossWeight	LearnRate
4	100,200,100,2	0,05	1e-3

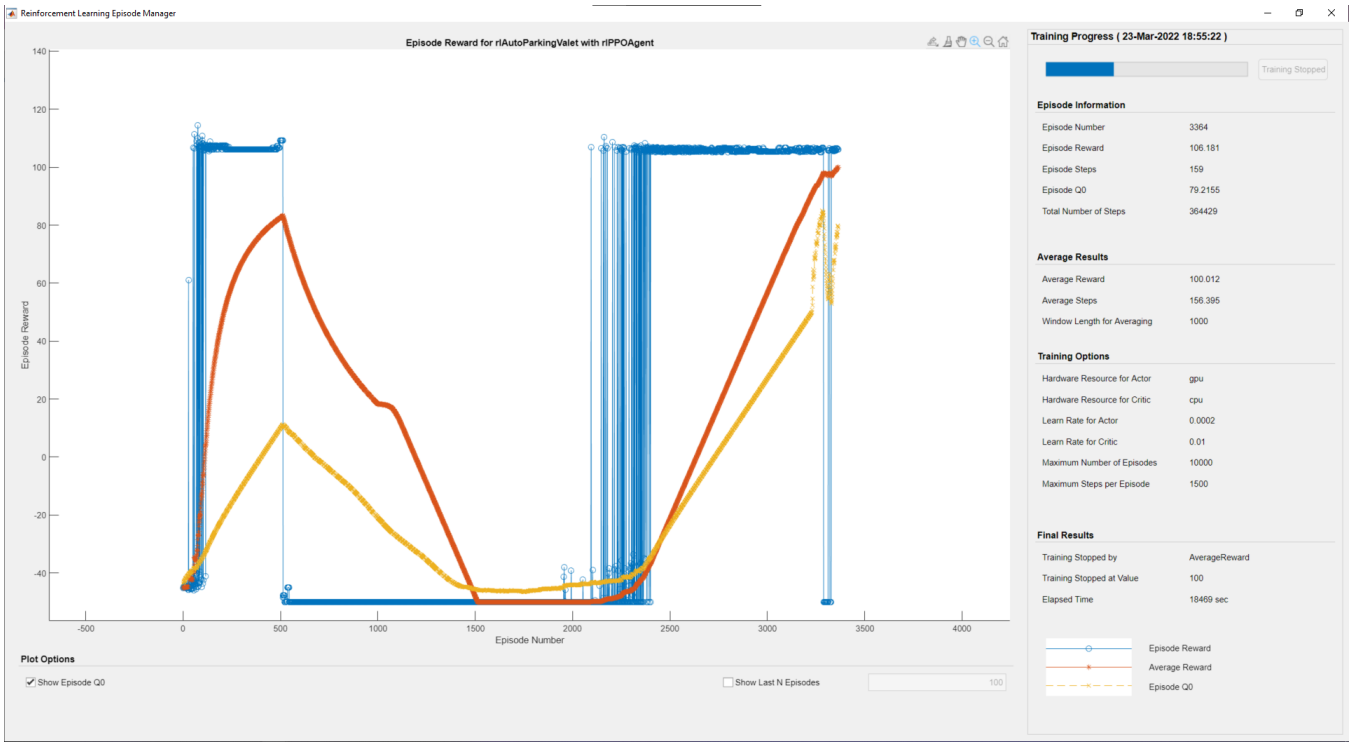

Figura 7.5: Gráfica de entrenamiento con paralelización

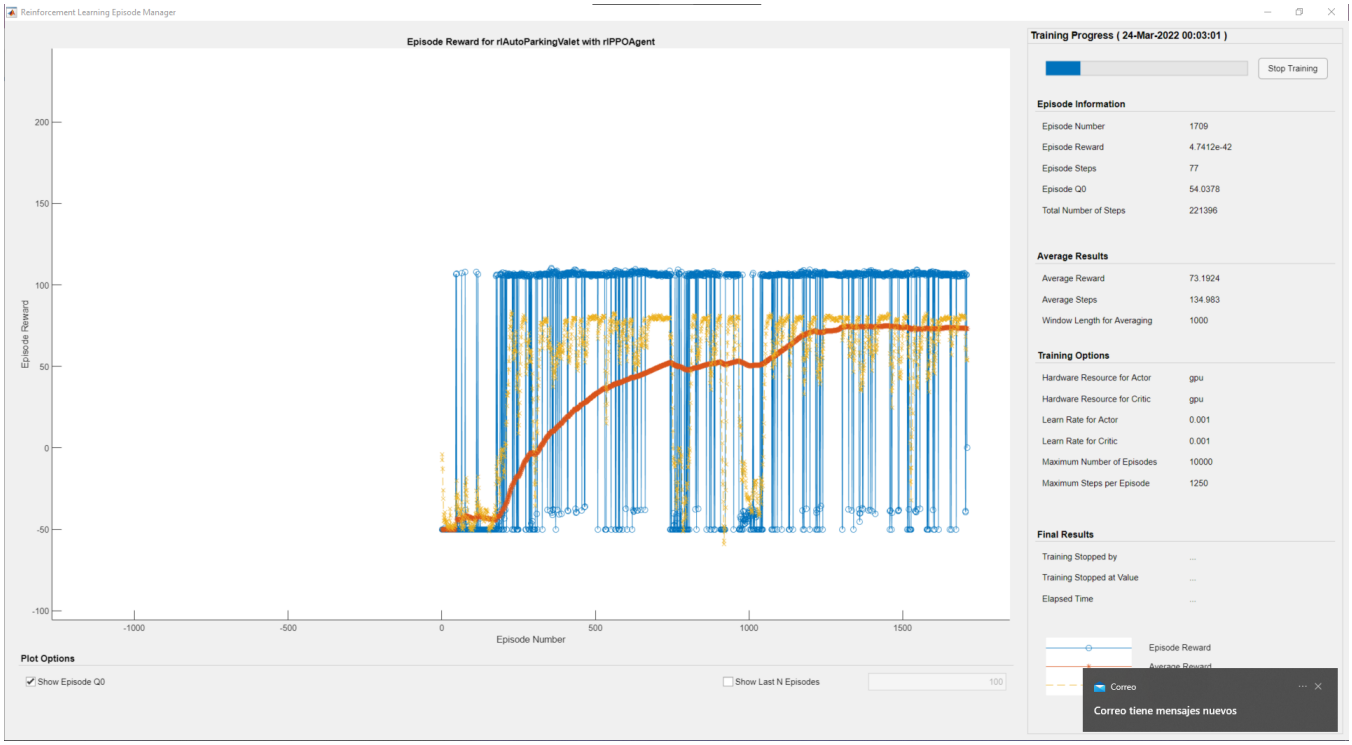

Figura 7.6: Gráfica del entrenamiento sin paralelización

Si comparamos los resultado con los del entrenamiento 1 podemos observar que los entrenamientos en paralelo en esta ocasión son mas fiables, y alcanza una recompensa media que hace que se detenga el entrenamiento. Mientras que los que se hacen secuencialmente muestran un comportamiento mas errático, aunque se podría considerar un resultado prometedor de haber continuado.

7.3 Análisis de los entrenamientos 1 y 2

En estos entrenamientos observamos que la ejecución en paralelo y en serie muestran comportamientos diferentes. Mientras que los entrenamientos en paralelo tienden a intercalar un gran numero de episodios positivos con otros negativos, los secuenciales se mantienen mas uniformes. Ademas se observan dos cosas: las recompensas oscilan muy poco, siendo -50 o 100, y la segunda es que el vehiculo realiza multitud de correcciones durante su movimiento.

Para el primer problema se observa que la función de recompensa esta despreciando la distancia al objetivo, por lo que se ha realizado una revisión de la función para que la distancia al destino cobre mayor relevancia. Para solucionar el segundo problema se ha añadido a la función de recompensa una penalización al numero de acciones totales.

Ademas de estos cambios, se ha decidido añadir un reloj que cuente el tiempo que ha durado el episodio para que recompense aquellos escenarios que aguantan mas tiempo y no llegan al destino y los que lleguen tardando menos tiempo. La implementación de este reloj es incompatible con la ejecución en paralelo, por lo que los siguientes entrenamientos se realizaran exclusivamente en serie.

También se le añaden mas haces al lidar del coche, pasando de 12 a 24.

7.4 Entrenamiento modificando la función de recompensa

Para esta serie de entrenamientos se han añadido los cambios en la función de recompensa, y se ha aumentado en gran medida la recompensa por la distancia al objetivo.

Tabla 7.3: Configuración de la red para el entrenamiento 3.

Numero de capas	**Nueronas por capa**	**EntropyLossWeight**	**LearnRate**
4	100,200,100,2	0,05	$1e^{-3}$

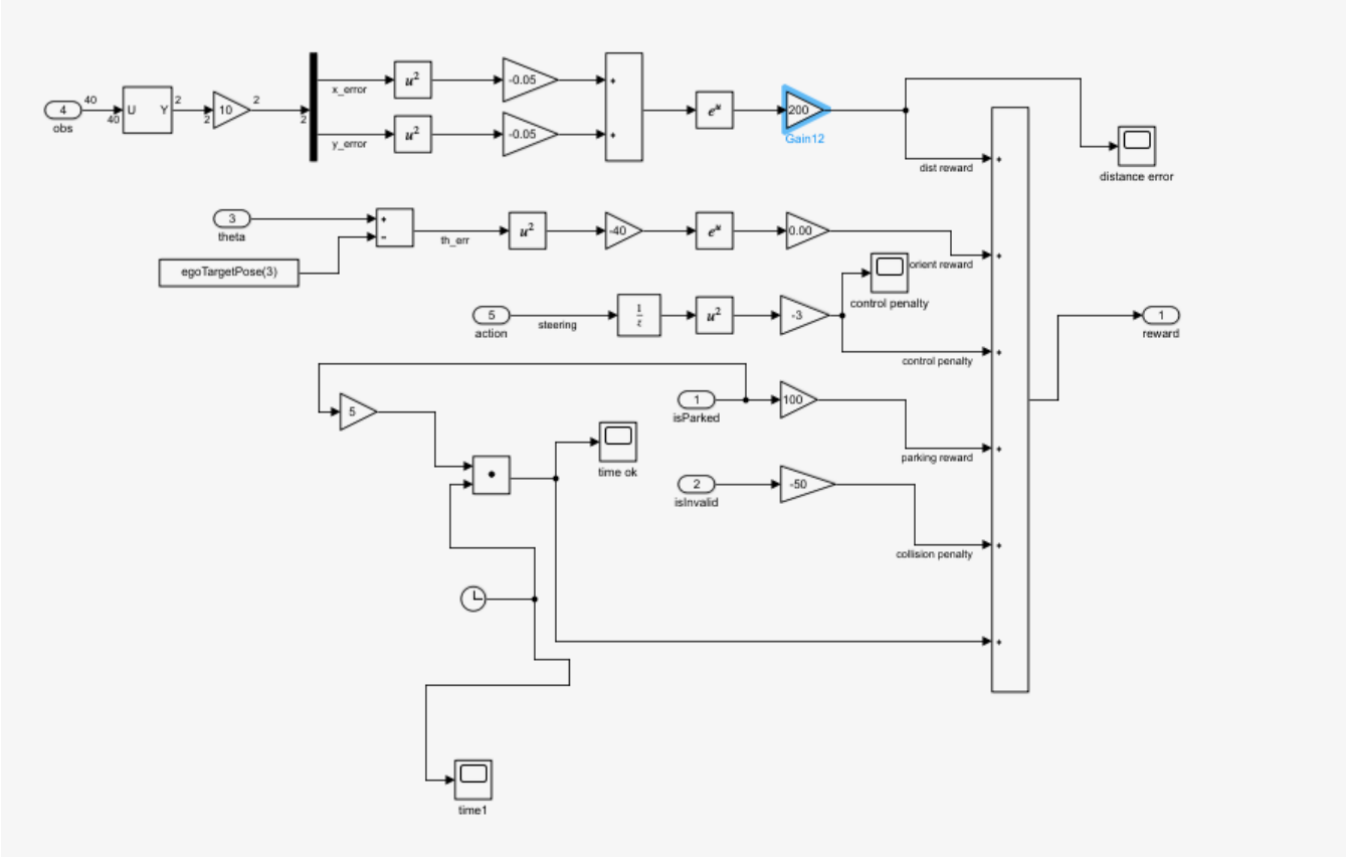

Figura 7.7: Función de recompensa del entrenamiento 3

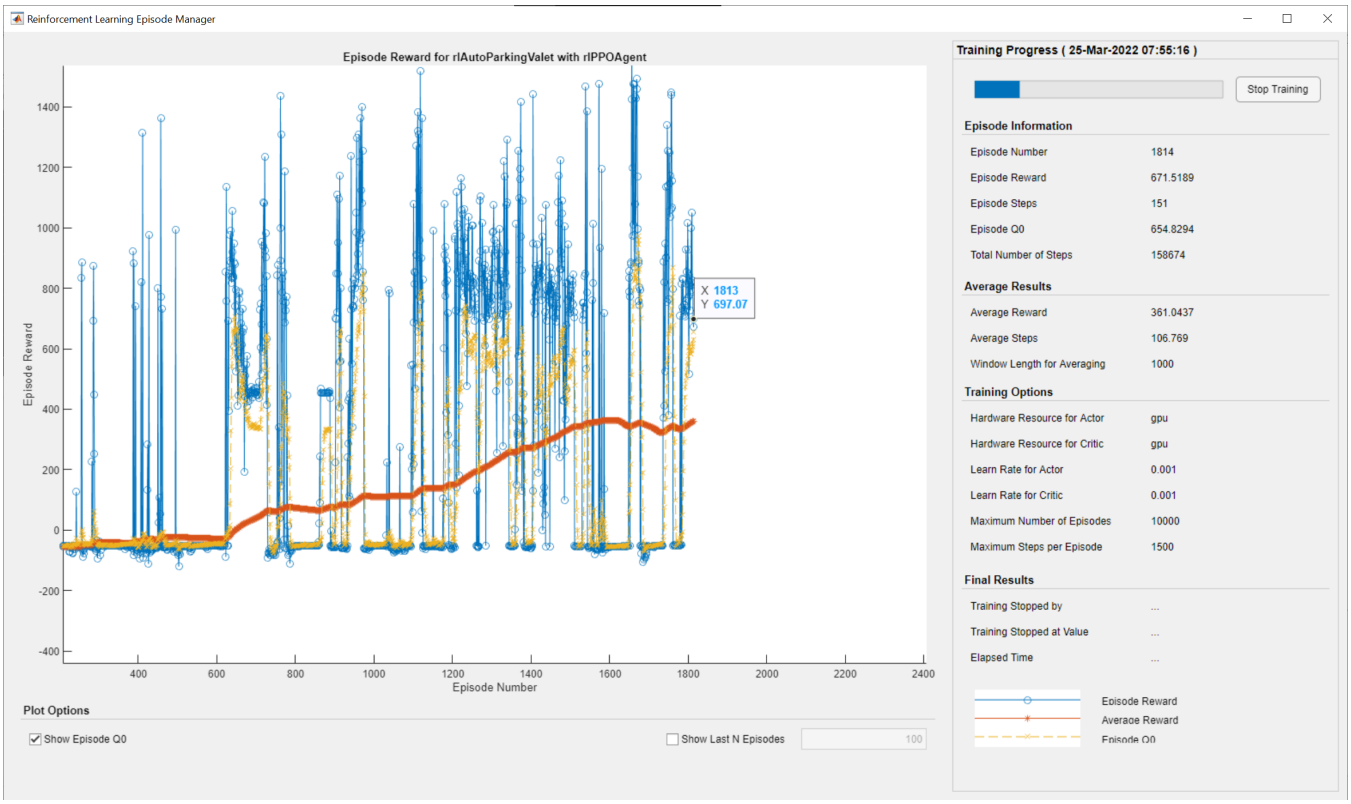

Figura 7.8: Gráfica del entrenamiento 3

Junto con este entrenamiento se realizaron otros 2 con pequeñas variaciones en las los valores de las recompensas. En todos los casos se observo el mismo problema, el valor que la distancia al objetivo aportaba era demasiado elevado en todas, cobrando así las mayor relevancia. En un principio el comportamiento que demuestra es correcto, pero se ha optado por modificar los valores para lograr una correcta lectura, y entrenamiento.

Por otra parte las penalizaciones por movimiento, y por tiempo, hacen que el vehículo realice movimientos mas precisos y con menos correcciones.

7.5 Entrenamiento con ajuste en la recompensa por distancia al objetivo

En este entrenamiento se descubrió el fallo en el valor de la recompensa por distancia al objetivo. Por el diseño que se tenía se premiaba en exceso aquellas distancias muy cercanas al destino con valores elevados, mientras que para la mayoría de las distancias del mapa la recompensa seria 0, debido al carácter exponencial de la función. Ademas de esto, se estaba realizando un sumatorio de las recompensas por distancia durante todo el episodio, por lo que se implementa una condición, por medio de una puerta lógica, para que solo se tome el valor de la distancia en el momento de terminar el entrenamiento.

Tabla 7.4: Configuración de la red para el entrenamiento 4.

Numero de capas	Nueronas por capa	EntropyLossWeight	LearnRate
4	100,200,100,2	0,05	$1e^{-3}$

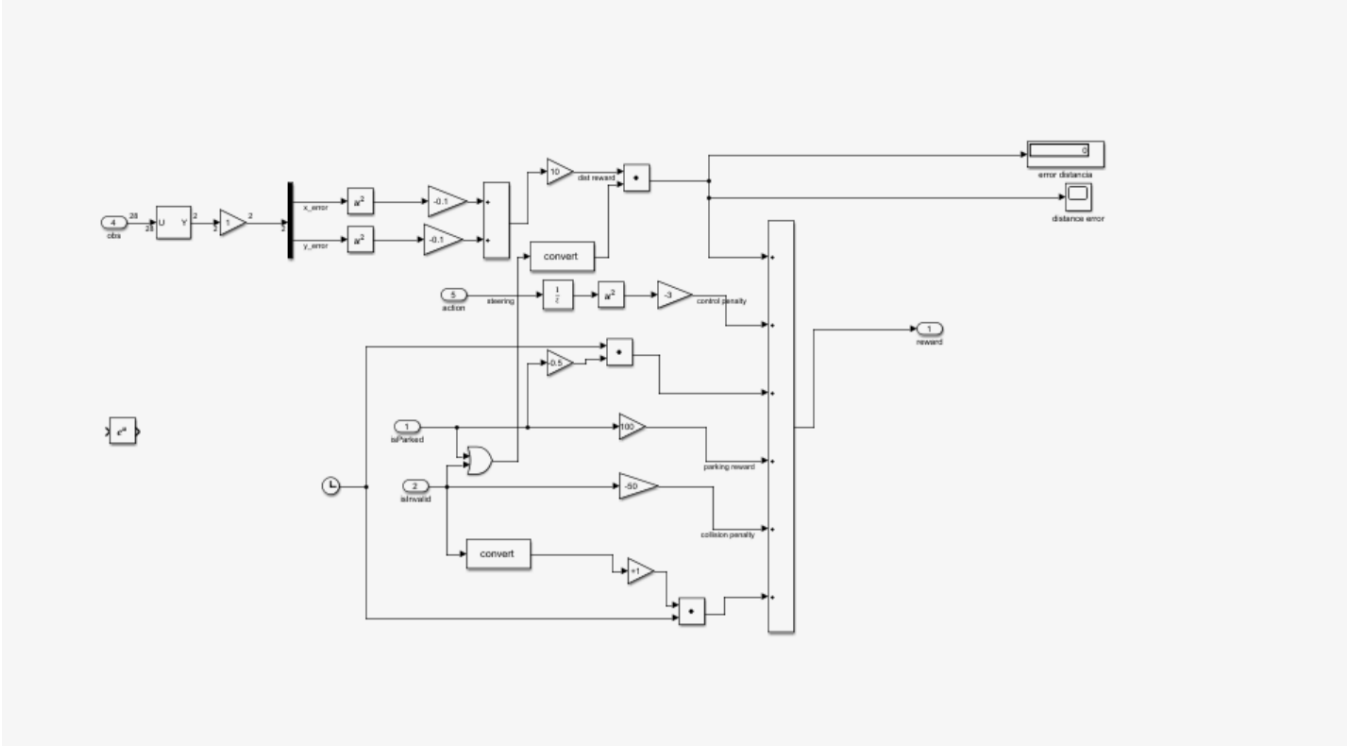

Figura 7.9: Función de recompensa del entrenamiento 4

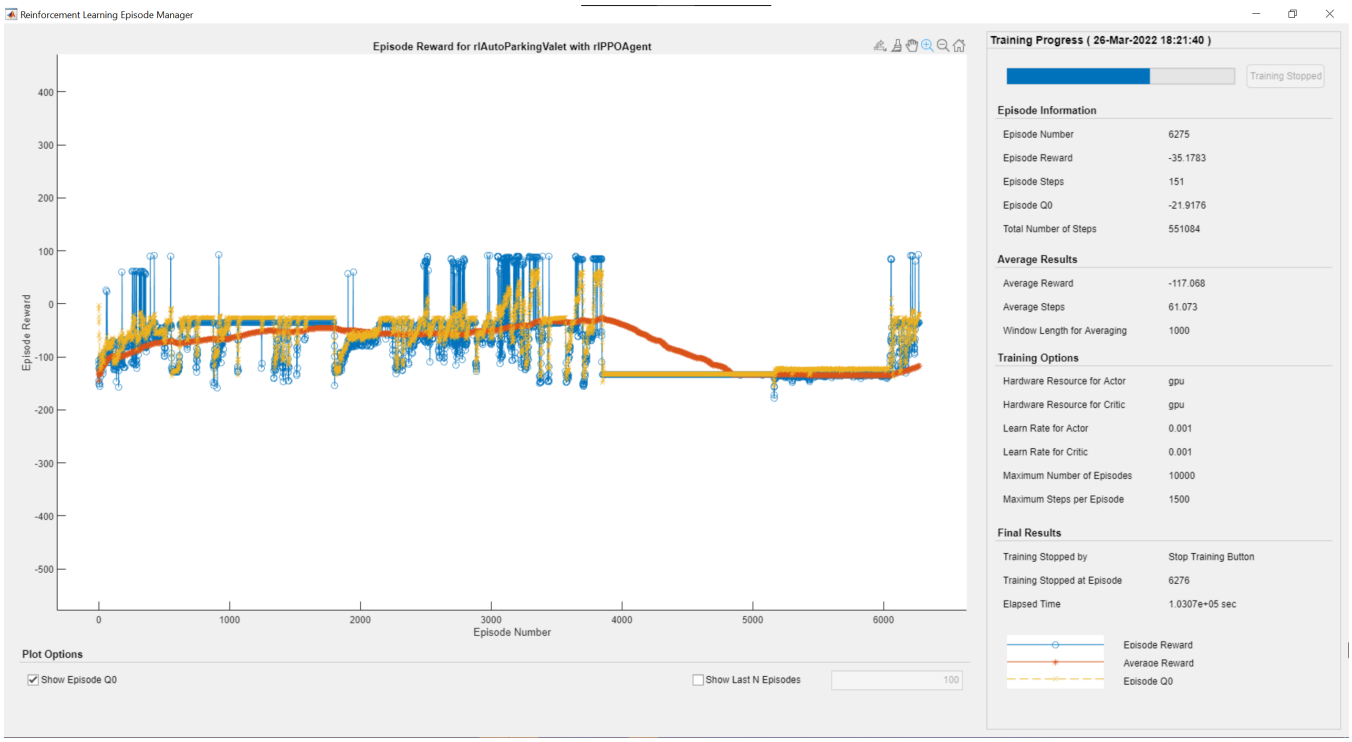

Figura 7.10: Gráfica del entrenamiento 4

Al igual que en el caso anterior se han realizado múltiples entrenamientos con la misma configuración y pequeñas modificaciones en los valores de la función.

Todos los resultado obtenidos son negativos, y muy pocos vehículos logran alcanzar el destino.

7.6 Entrenamiento eliminando el operador exponencial en el calculo de recompensa por distancia al objetivo

Para esta serie de entrenamientos se ha eliminado la componente exponencial, para que la recompensa en todos los puntos tenga la misma importancia he se comporte de una forma lineal. Al tratarse de la diferencia entre el destino y la posición, el valor que se obtiene es negativo, por lo que para transformarlo en uno positivo se ha añadido un offset, aproximado al valor máximo de la recompensa.

Tabla 7.5: Configuración de la red para el entrenamiento 5.

Numero de capas	Nueronas por capa	EntropyLossWeight	LearnRate
4	100,200,100,2	0,05	$1e^{-3}$

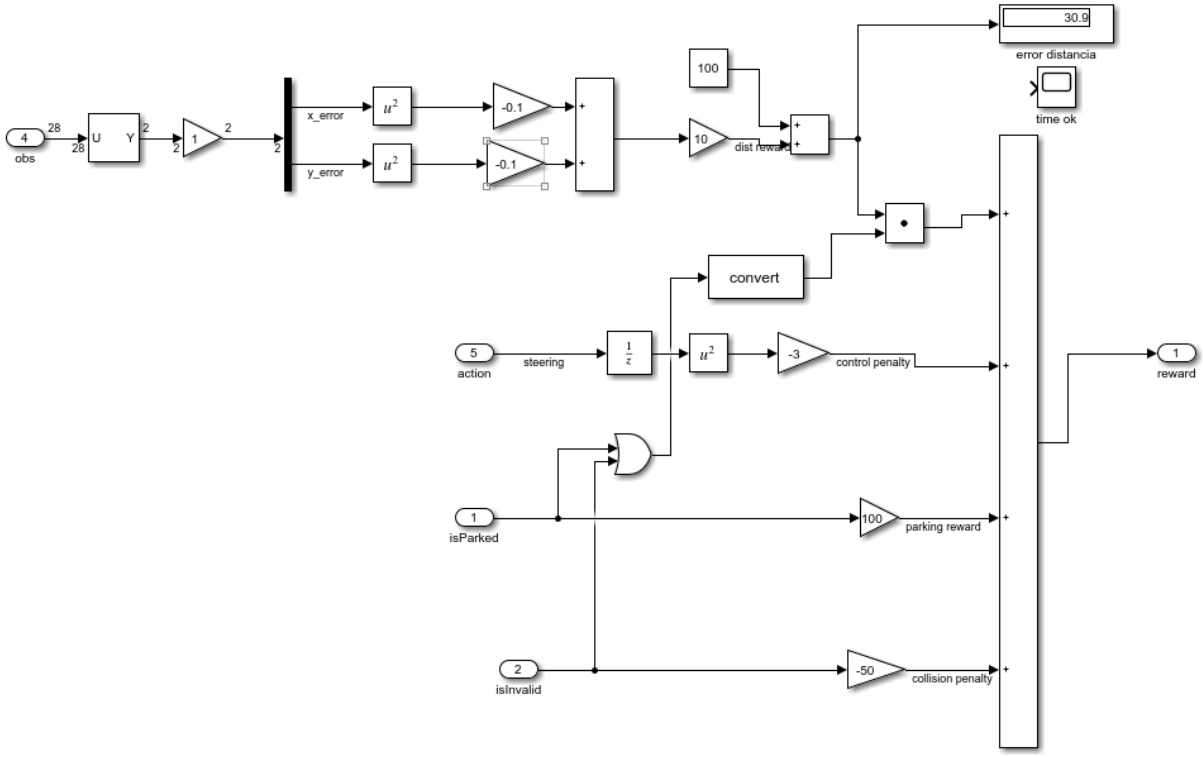

Figura 7.11: Función de recompensa del entrenamiento 5

Figura 7.12: Gráfica del entrenamiento 5

En este entrenamiento se obtiene un entrenamiento mas parecido al deseado, aunque sigue existiendo demasiada oscilación entre los resultados y no se observa un progreso real a lo largo del entrenamiento.

Mientras que para el resto de pruebas los resultado son peores. Quedando la recompensa oscilando en torno al 0 y sin encontrar el destino.

7.7 Entrenamiento añadiendo la exponencial en el calculo de la recompensa y ajustando pesos y dimensión de la red

Para este entrenamiento se ha vuelto a la exponencial ajustando mejor los pesos, también se ha reducido el peso en la recompensa del resto de elementos, y se ha modificado la red para hacerla mas homogénea. También se ha modificad el LearnRate para que aprenda mas despacio y más controlado. Por ultimo se ha reducido el EntropyLossWeight para evitar las caídas en el aprendizaje.

Tabla 7.6: Configuración de la red para el entrenamiento 6.

Numero de capas	Nueronas por capa	EntropyLossWeight	LearnRate
4	200,200,200,2	0,01	$2e^{-4}$

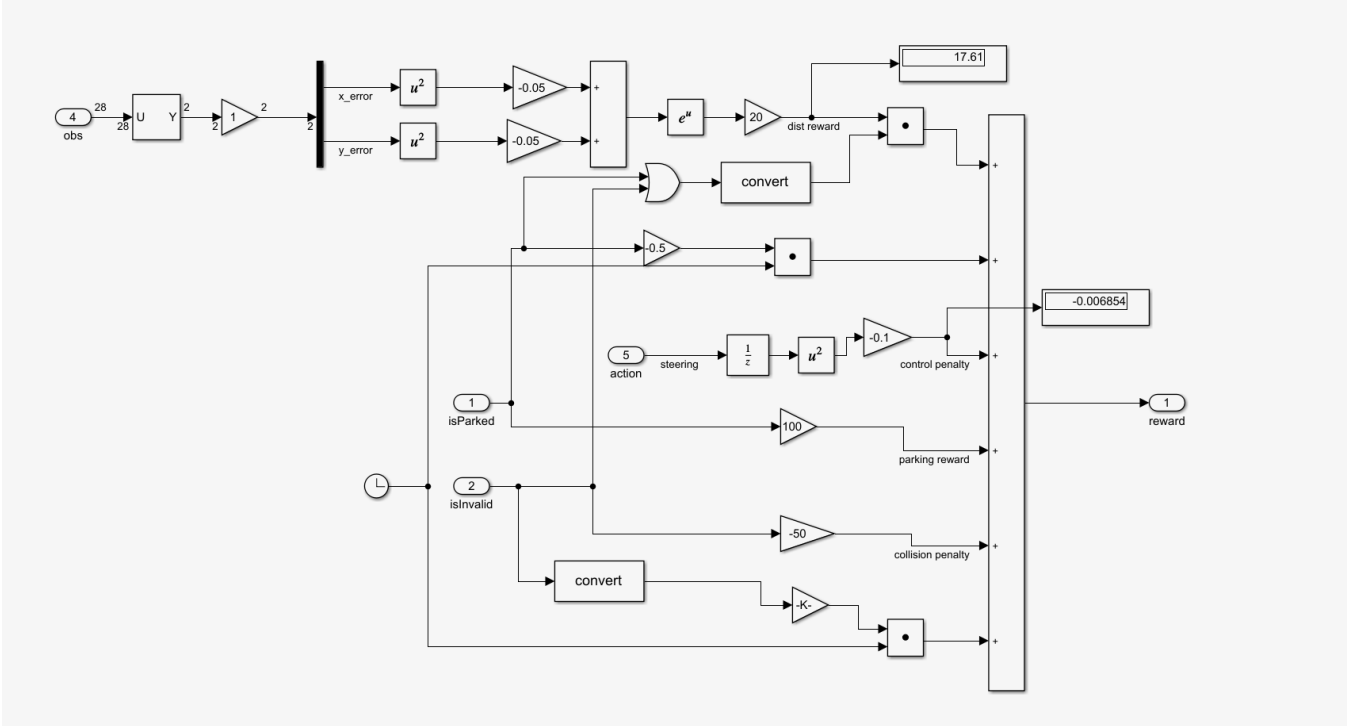

Figura 7.13: Función de recompensa del entrenamiento 6

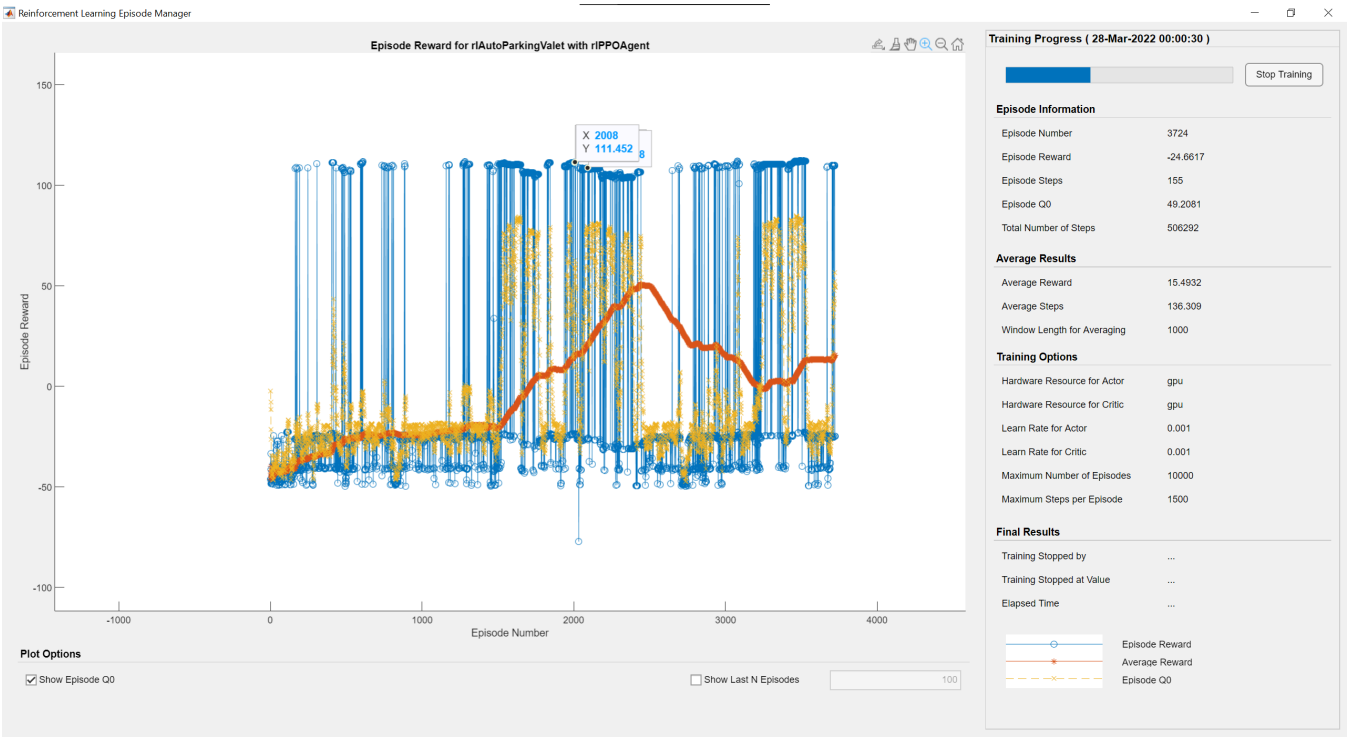

Figura 7.14: Gráfica del entrenamiento 6

Durante el transcurso de este entrenamiento se ha observado que el agente encontró trucos para maximizar su recompensa sin llegar al objetivo, pues se quedo dando vueltas en círculos dejando pasar el tiempo y colisionando cerca del objetivo. Aunque con el trascurso de los entrenamientos esta practica cesó. Como se observa en la gráfica el agente va incrementando su recompensa y mejorando su desempeño a lo largo del entrenamiento, peor al igual que pasaba en los primeros casos, llega un momento en el que deja de lograr el objetivo y el entrenamiento sufre un enorme retroceso. Por otra parte se observa que el valor aportado por la distancia a al objetivo vuelve a ser despreciable.

7.8 Entrenamiento ampliando el tamaño de la red

En este entrenamiento se ha continuado con las premisas del entrenamiento anterior aumentando el tamaño de la red.

Tabla 7.7: Configuración de la red para el entrenamiento 7.

Numero de capas	Nueronas por capa	EntropyLossWeight	LearnRate
4	400,400,400,2	0,02	$2e^{-4}$

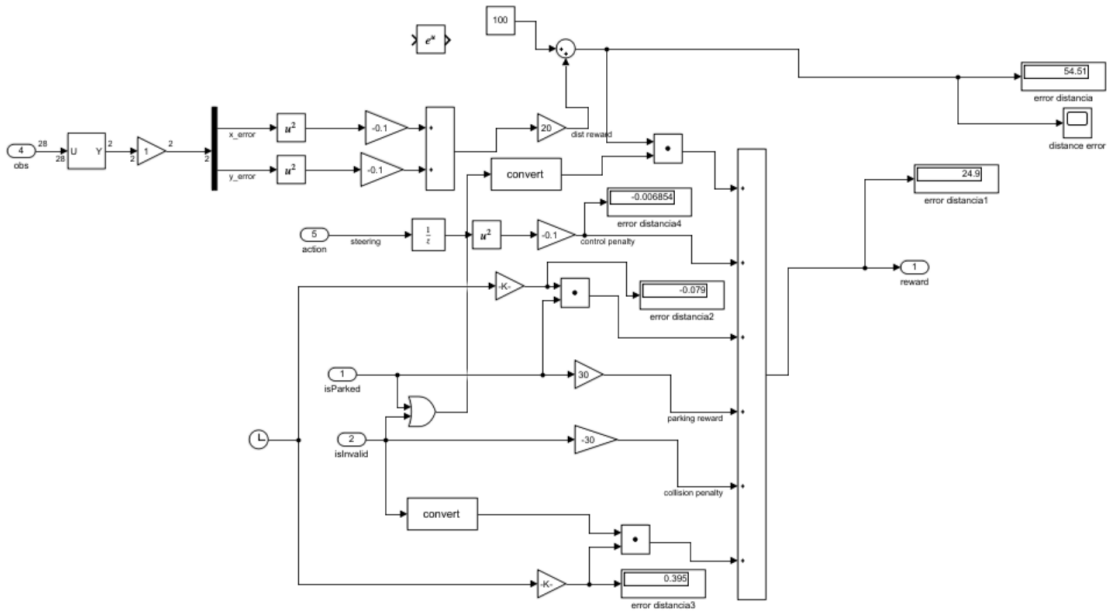

Figura 7.15: Función de recompensa del entrenamiento 7

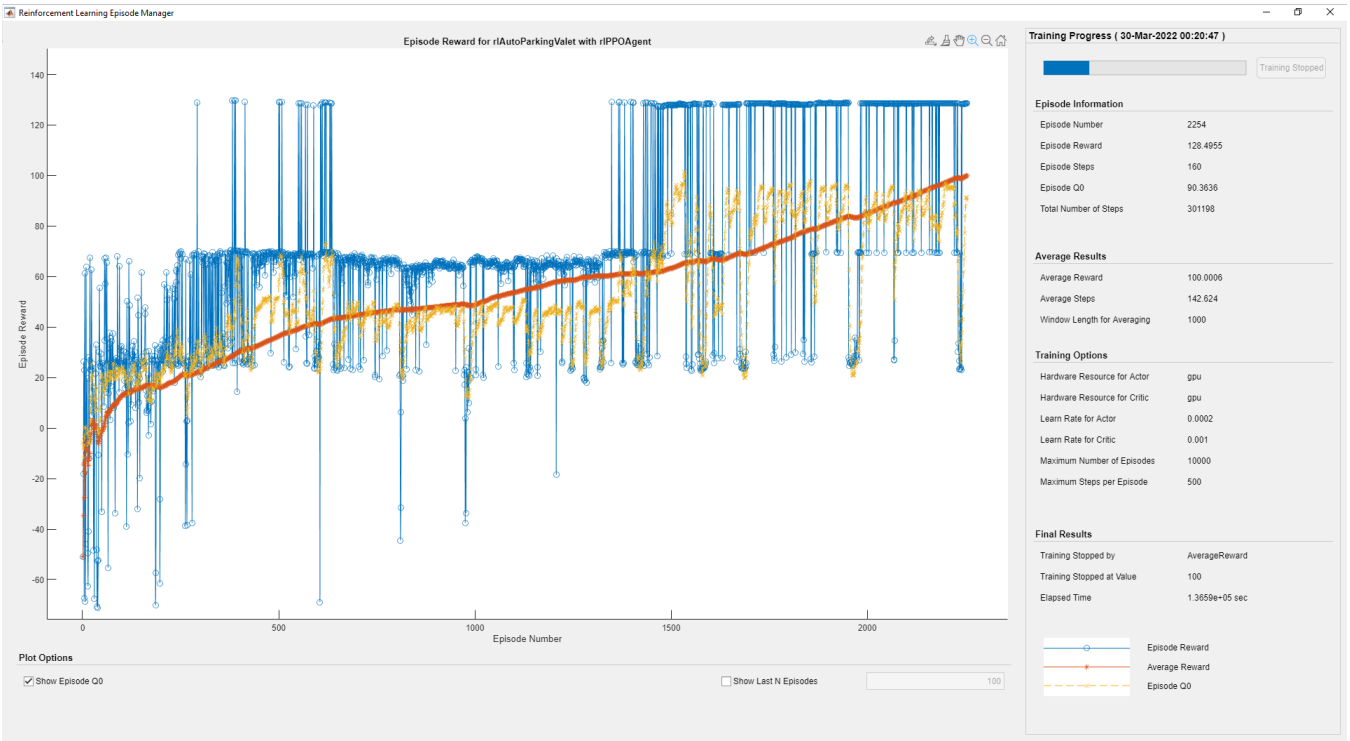

Figura 7.16: Gráfica del entrenamiento 7

Este entrenamiento ha obtenido resultados muy positivos, logrando un agente que alcanza el objetivo en un alto numero de ocasiones, y para aquellos episodios que no alcanza el destino, se puede comprobar como en el momento de la colisión su distancia al destino es pequeña. Si analizamos la gráfica comprobamos que en los primeros episodios con recompensas bajas se deben a la distancia al destino. Según se avanza

en el tiempo se comprueba como el vehículo comienza a acercarse al objetivo, sin llegar a alcanzarlo. Y en la ultima etapa alcanza el destino en la mayoría de ocasiones.

7.9 Entrenamiento con un inicio aleatorio

En este entrenamiento se ha implementado una pequeña aleatoriedad en la función reset, por la cual la posición y el ángulo en el que comienza el vehículo es aleatorio dentro de unos rangos concretos. Para mejorar el rendimiento de la red se ha ampliado el tamaño de la red.

Tabla 7.8: Configuración de la red para el entrenamiento 8.

Numero de capas	Nueronas por capa	EntropyLossWeight	LearnRate
4	1000,1000,1000,1000,2	0,05	$2e^{-3}$

Figura 7.17: Función de recompensa del entrenamiento 8

Figura 7.18: Gráfica del entrenamiento 8

Los resultados observado de este entrenamiento no son positivos, pues podemos observar su incapacidad para alcanzar el objetivo.

7.10 Entrenamiento en un circuito complejo con un inicio aleatorio

Para este entrenamiento se ha modificado el circuito generando un entorno más complejo, manteniendo la aleatoriedad de la función reset, por la cual la posición y el ángulo en el que comienza el vehículo es aleatorio dentro de unos rangos concretos. Para mejorar el rendimiento de la red se ha ampliado el tamaño de la red, además el actor y el critico ya no comparten la misma arquitectura de red.

Tabla 7.9: Configuración de la red para el entrenamiento 9.

Numero de capas	Nueronas por capa	EntropyLossWeight	LearnRate
5	500,500,500,500,1	0,05	$1e^{-3}$

Tabla 7.10: Configuración de la red para el entrenamiento 9.

Numero de capas	Nueronas por capa	EntropyLossWeight	LearnRate
6	2000,2000,2000,2000,,2000, 2	0,05	$1e^{-3}$

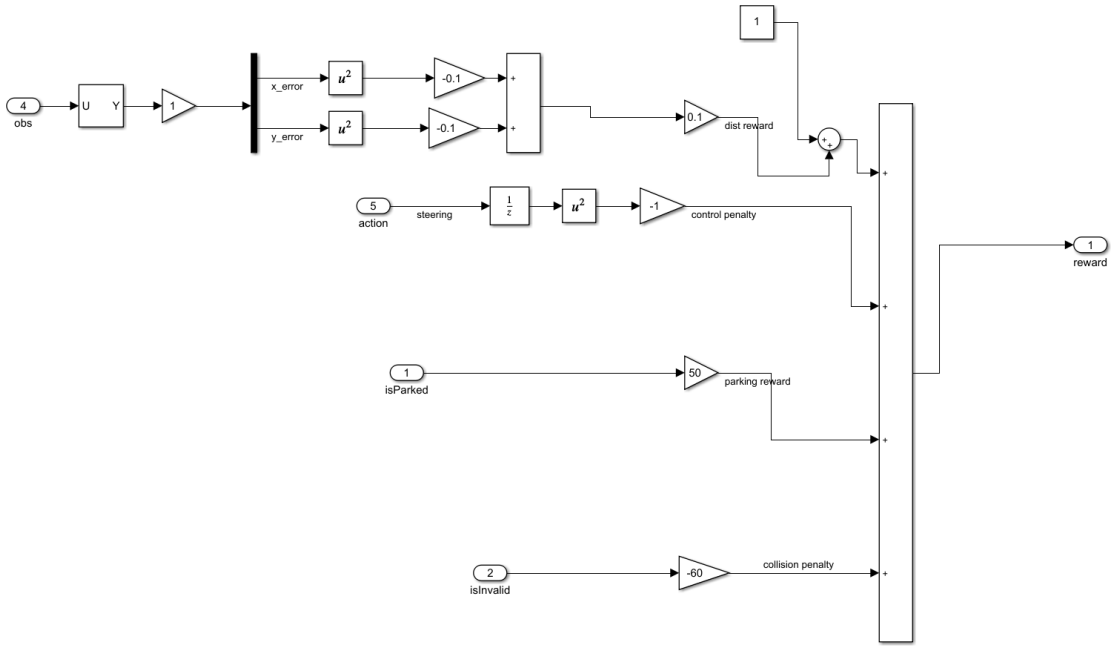

Figura 7.19: Función de recompensa del entrenamiento 9

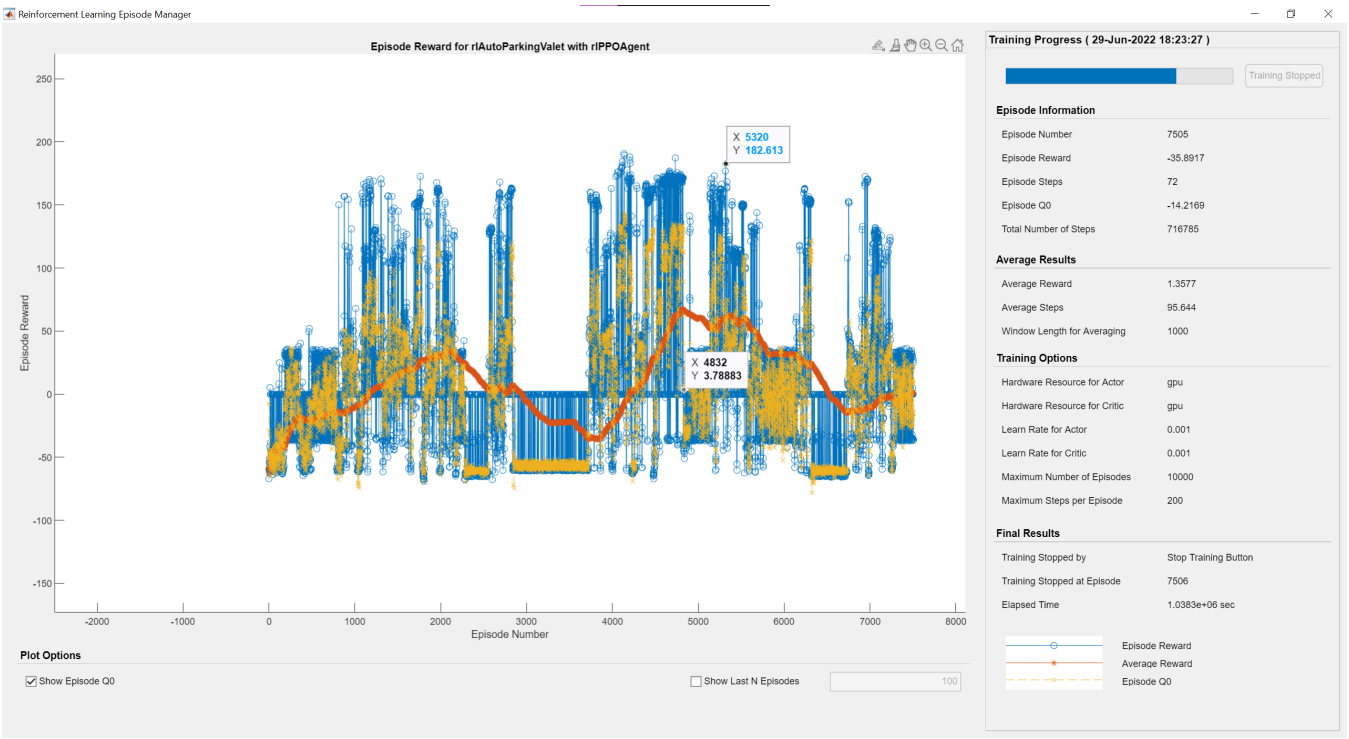

Figura 7.20: Gráfica del entrenamiento 9

Los resultados observado de este entrenamiento no son positivos, pues podemos observar su incapacidad para alcanzar el objetivo.

7.11 Entrenamiento en un circuito complejo con un inicio aleatorio con Matlab2022a

Para este entrenamiento se ha modificado el circuito generando un entorno más complejo, manteniendo la aleatoriedad de la función reset, por la cual la posición y el ángulo en el que comienza el vehículo es aleatorio dentro de unos rangos concretos. En este caso se han probado diferentes redes. Esta ejecución se ha hecho sobre la versión 2022a de MATLAB.

Tabla 7.11: Configuración de la red para el entrenamiento 9.

Numero de capas	Nueronas por capa	EntropyLossWeight	LearnRate
5	128,128,128,1	0,05	$1e^{-3}$

Tabla 7.12: Configuración de la red para el entrenamiento 9.

Numero de capas	Nueronas por capa	EntropyLossWeight	LearnRate
4	256,256,256, 2	0,05	$5e^{-4}$

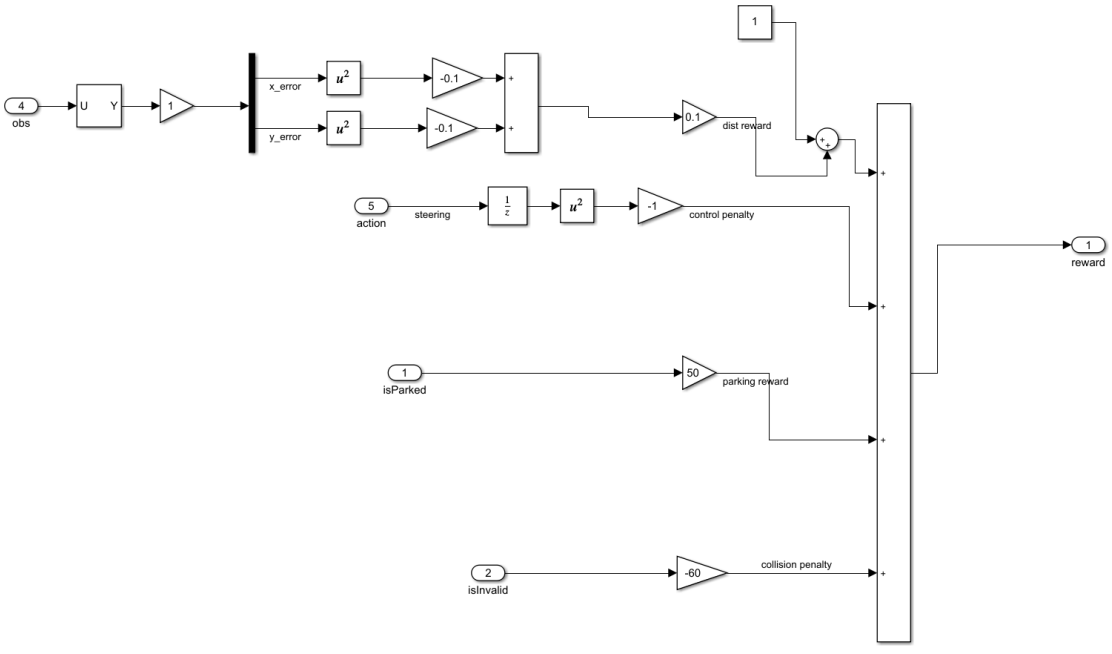

Figura 7.21: Función de recompensa del entrenamiento 10

Figura 7.22: Gráfica del entrenamiento 10

Los resultados observado de este entrenamiento son positivos, en el se observa que el agente alcanza el objetivo en un buen porcentaje de las veces.

Capítulo 8

Resultados

Tras el análisis de los resultados de todos los entrenamientos, se ha observado de la importancia de la recompensa en el desempeño del agente. No existe una recompensa ideal o correcta, pero según los resultados que se han obtenido, la recompensa con mejores resultados ha sido la siguiente.

$$Reward = (100 - 20(0{,}1X_e^2 + 0{,}1Y_e^2))(\rho + \iota) - 0{,}1(1/\gamma)^2 + (100 - 0{,}1t_f)\rho - (50 - 0{,}1t_f)\iota \tag{8.1}$$

En esta ecuación se puede ver la diferencia con la ecuación inicial. Se elimina la exponencial de e, para generar una recompensa mas lineal, como resultado a la distancia. También se añade la condición de finalización en la recompensa por distancia, para que solo se calcule la distancia al destino en el último instante. Se ha añadido el factor gama para penalizar el numero de acciones, y que así el agente realice menos acciones y más suaves. Por último se añade una recompensa a aquellos agentes que no alcancen el destino, pero se hallan mantenido más tiempo entrenando, para premiar a aquellos que buscan el destino. Mientras que se ha penalizado a aquellos que si alcanzan el destino, pero lo hacen por una ruta muy larga.

Otro de los parámetros que se ha ido variando a sido la arquitectura de la red, la cual se ha aumentado su dimensión y se ha homogeneizado el numero de neuronas entre todas las capas. Durante las pruebas de diferentes arquitecturas se probo a generar una con multitud de capas y menos neuronas por capas. Como resultado a esta prueba se generaron agentes incapaces de aprender. La causa de que estos agentes no fueran capaces de aprender se debe al descenso del gradiente. un problema común en el diseño de las redes. Tras estas pruebas se volvió a generar redes de 4 capas.

Como se puede comprobar en el entrenamiento 7 mostrado anteriormente, se ha logrado un entrenamiento con una gráfica creciente, que no sufre de decaídas durante el proceso. Si se observa ese entrenamiento, se pueden ver 4 franjas de acción:

- La primera franja esta situada por debajo del valor de recompensa 20. Aquí se encuentran aquellos episodios donde el agente se alejo del objetivo. Sufriendo así la mayor penalización.
- La segunda franja esta situada entre el 20 y el 40 de valor de recompensa. Aquí se encuentran todos los agentes que se acercaron al objetivo, pero chocaron con algún obstáculo.
- La siguiente franja se sitúa entre 60 y 80, estando aquí aquellos episodios que pasaron cerca del destino y se chocaron con el muro posterior.

- En el ultimo escalón están los que alcanzaron una recompensa de 120. Son aquellos que no sufren la penalización por colisión y que reciben el premio por llegar. Por esa razón se nota tanto el salto. contenidos...

si se observa bien dentro de cada escalón existe una ligera variación en la recompensa, esta variación se debe por una parte a la distancia al objetivo y por otra a la penalización por número de acciones o tiempo. En los primero episodios podemos ver que existe una gran variación entre ellos. Pero en los últimos prácticamente no varía, convirtiéndose en una recta horizontal.

8.1 Estadísticas modelo simple

Con los resultados obtenidos de los entrenamientos anteriores, se ha procedido a obtener unas estadísticas del desempeño de los agentes.

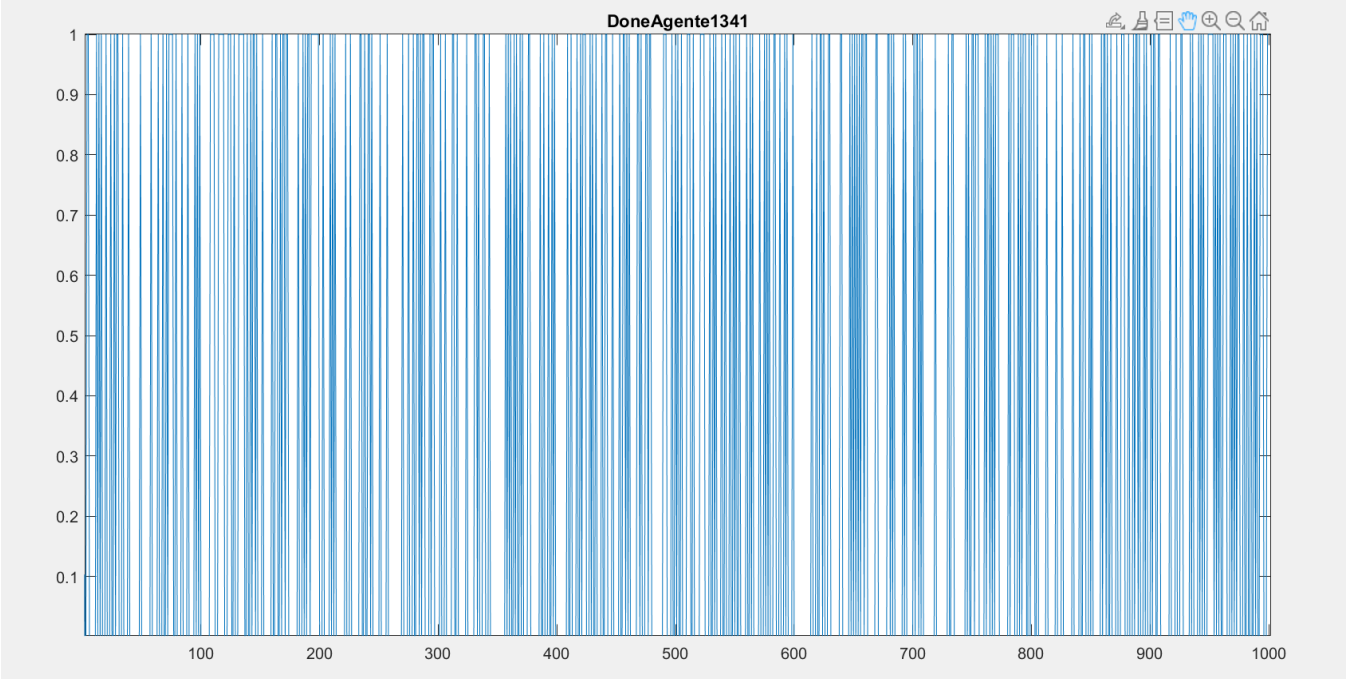

Figura 8.1: Resultados de las estadísticas

Ha continuación se observa una gráfica con los resultados de las estadísticas pasadas al agente en un modelo simple. En estas gráfica se observa si el agente alcanzo el objetivo o no en cada iteración. En este caso tenemos un desempeño del 31.5 % de aciertos.

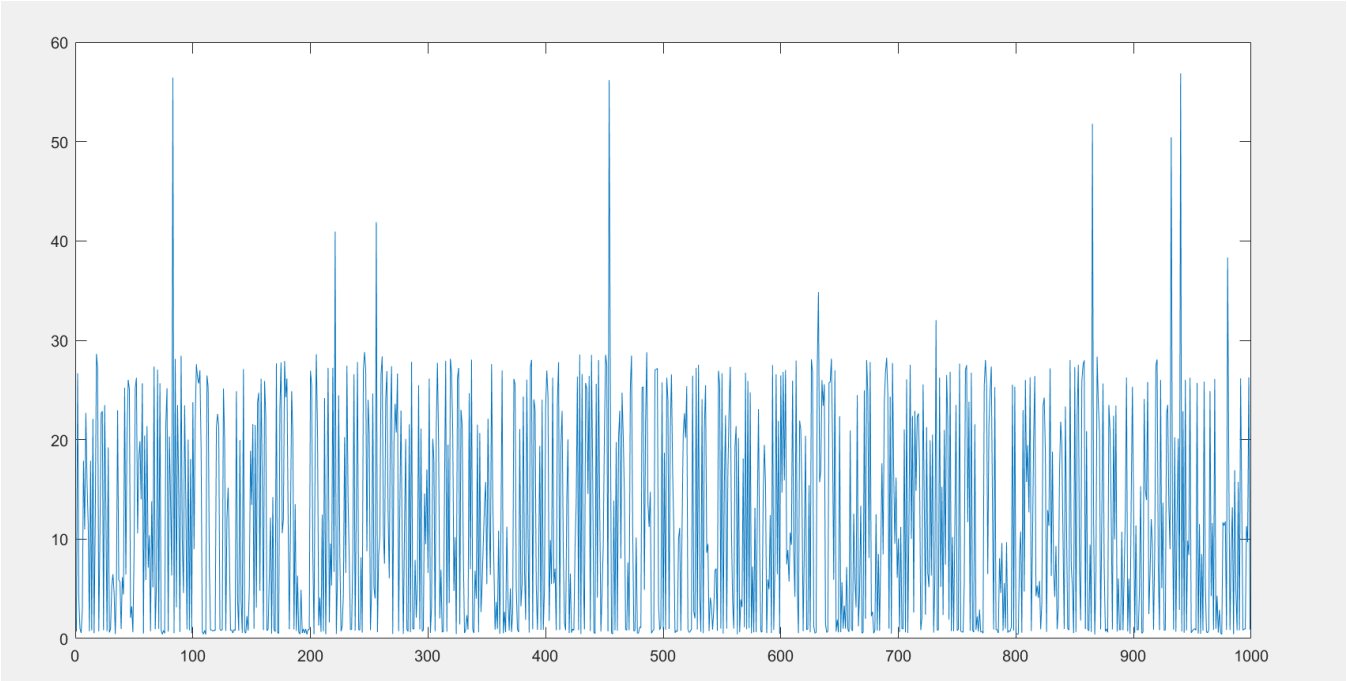

Figura 8.2: Distancia al objetivo

En esta gráfica se observa la distancia a la que se ha quedado el agente del objetivo al momento de terminar la simulación.

8.2 Estadísticas modelo complejo

Con los resultados obtenidos de los entrenamientos anteriores, se ha procedido a obtener unas estadísticas del desempeño de los agentes.

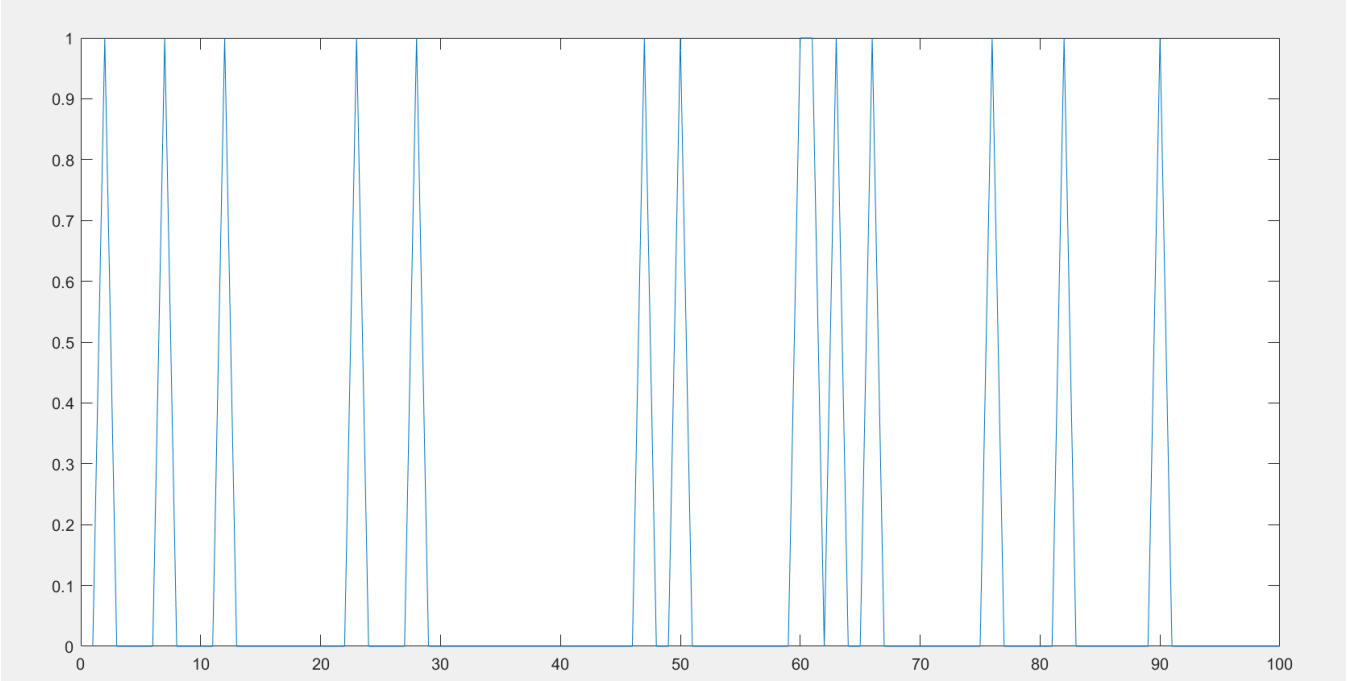

Figura 8.3: Resultados de las estadísticas

Ha continuación se observa una gráfica con los resultados de las estadísticas pasadas al agente en un modelo simple. En estas gráfica se observa si el agente alcanzo el objetivo o no en cada iteración. En este caso tenemos un desempeño del 14 % de aciertos. El cual es inferior al escenario simple.

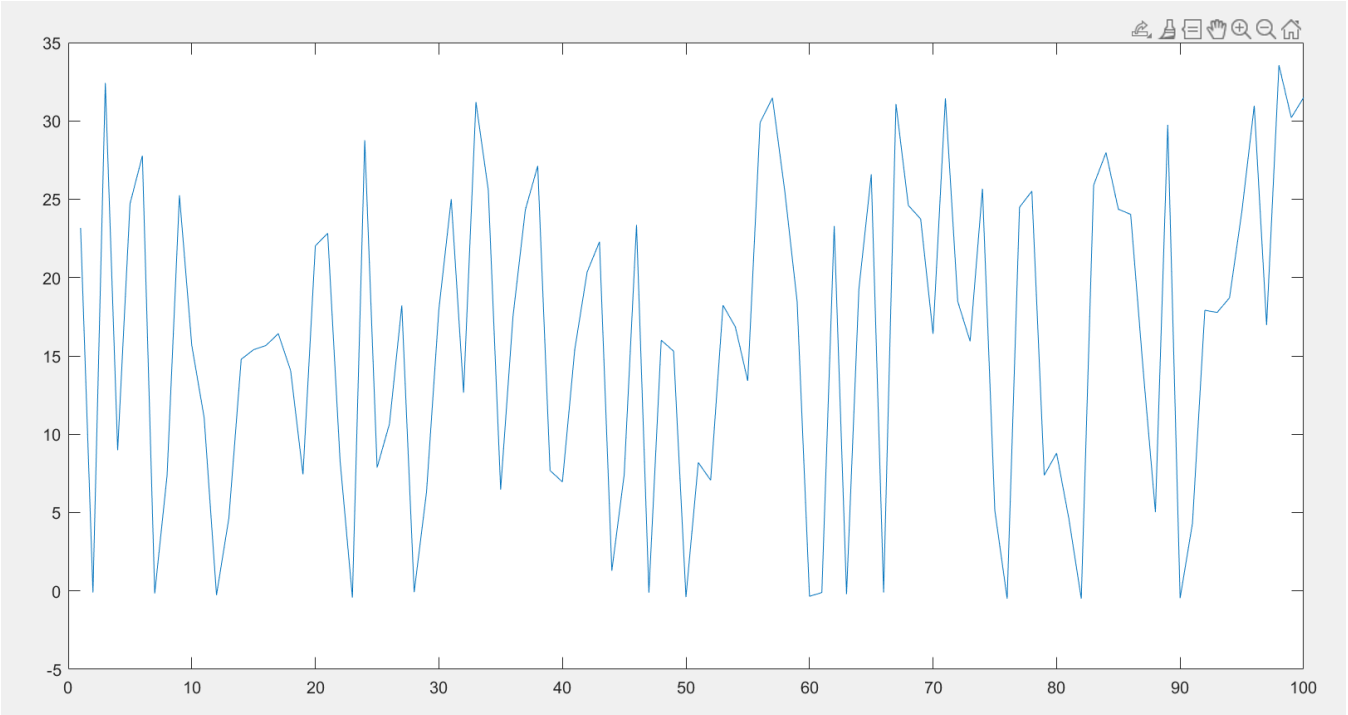

Figura 8.4: Distancia al objetivo

En esta gráfica se observa la distancia a la que se ha quedado el agente del objetivo al momento de terminar la simulación. Con una distancia media al destino de 15.89 metros.

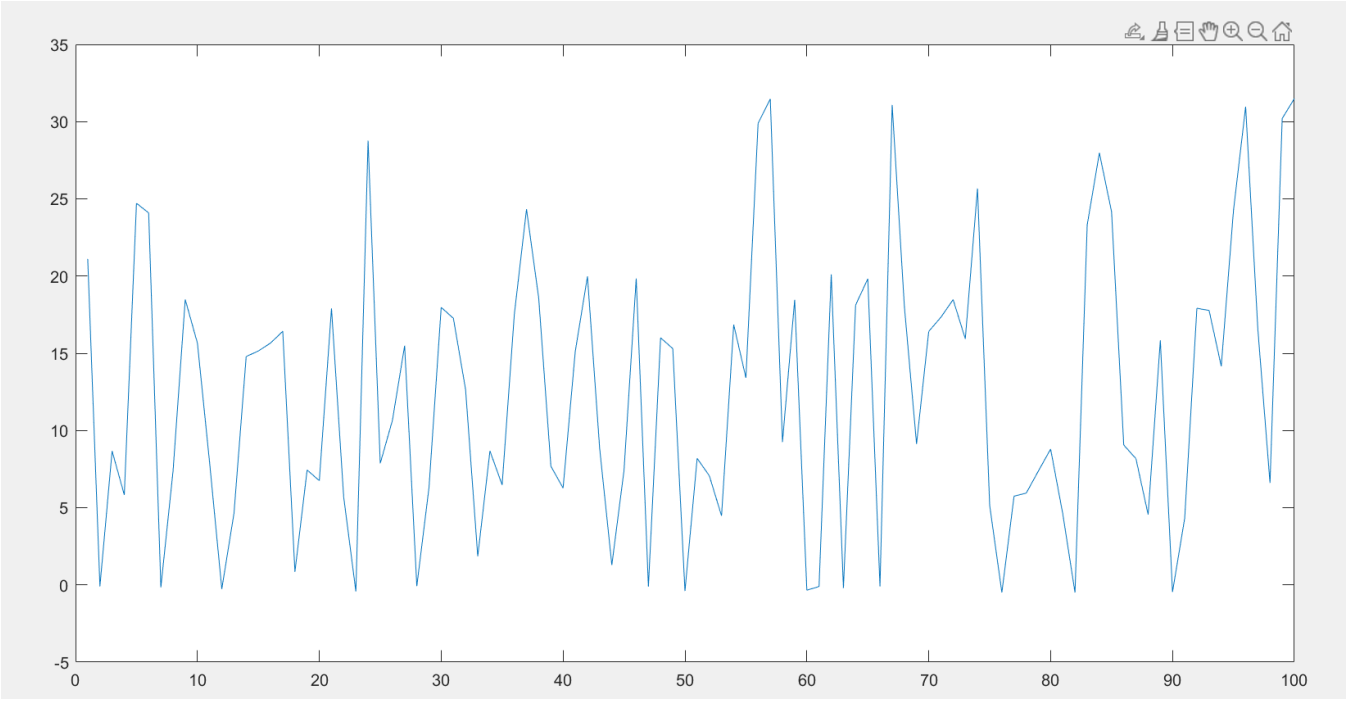

Figura 8.5: Distancia mínima al objetivo

En esta gráfica se observa la distancia mínima que ha existido entre el agente y el destino durante la ejecución de la simulación. La distancia media mínima es de 12.4175 metros, siendo esto un indicativo de que han existido simulaciones donde el agente ha pasado cerca del objetivo sin llegar a alcanzarlo.

Capítulo 9

Conclusiones

Durante el transcurso de este proyecto se ha logrado diseñar una red y una función de recompensa capaz de entrenar agentes que logren alcanzar el destino en escenarios sencillos. Por el contrario no se han logrado los mismos resultados en escenarios mas complejos.

Al no disponer de un sistema de visión capaz de detectar el objetivo al que se debe dirigir, el vehículo va a ciegas hasta alcanzar de forma aleatoria el destino. Esto imposibilita la generación aleatoria del entorno, ni del vehículo.

Por otro parte se debe considerar los periodos de entrenamiento, pues estos se pueden alargar en el tiempo llegando a alcanzar varios días, llegando los mas largos a alcanzar la semana de entrenamiento. Esto variación se debe a la complejidad de la red, la dificultad del entorno, y el numero de elementos del lidar. Los entrenamientos en paralelo aceleran en gran medida este entrenamiento, Pero por los resultados observado no es igual de fiable ademas de no ser compatible con la medición del tiempo de cada episodio.

Otro parámetro a tener en cuenta es LearnRate (velocidad de aprendizaje) el cual se ha llegado a deducir que debe de tener un valor pequeño. Esto hace que el agente alcance una política correcta de una forma mas precisa, mientras que por el contrario, tarda muchos mas episodios en alcanzar la misma. En este proyecto actor y crítico han contado con un LearnRate diferente siendo mucho mayor el del critico que el del actor, pues el actor necesita mayor precisión en su política.

Capítulo 10

Líneas futuras

En este apartado se van a explorar las posibles mejoras o evoluciones del sistemas.

- La primera posible mejora es añadir al vehículo una cámara que sea capaz de detectar el destino. De esta manera el sistema al tener mas información diferente, mejoraría su rendimiento.
- Ajustar mejor el sistema para lograr que se desenvuelva en sistemas mas complejos, donde halla mas obstáculos.
- Si se implementa la cámara, se podría generar el vehículo de forma aleatoria. Para esto habría que controlar el punto en el que se va a generar el vehículo no este ocupado.
- Al igual que la aparición del vehículo, se podría generar los obstáculos de forma aleatoria.

Bibliografía

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra y M. Riedmiller, Playing Atari with Deep Reinforcement Learning, DeepMind Technologies, 2013.

[2] La gran carrera por el coche autonomo. PTCarretera
https://www.ptcarretera.es/la-gran-carrera-por-el-coche-autonomo-quienes-son-los-competidores/.

[3] SAE J3016 automated-driving graphic.
https://www.sae.org/news/2019/01/sae-updates-j3016-automated-driving-graphic.

[4] Los niveles de conduccion autonoma. Xataka
https://www.xataka.com/automovil/de-0-a-5-cuales-son-los-diferentes-niveles-de-conduccion-autonoma.

[5] Juan Humberto Sossa Azuela y Alejandro Peña Ayala, Estado del arte en Inteligencia artificial y ciencia de datos. Septiembre 2019.

[6] Alberto de Torres, Estado del arte de la Inteligencia Artificial en 2021. 23 de Febrero de 2021.
https://www.sage.com/es-es/blog/estado-del-arte-de-la-inteligencia-artificial-en-2021/

[7] Ventajas del Machine Learning y Deep Learning para la evolución de la visión artificial. 13 noviembre, 2019.
https://blog.infaimon.com/machine-learning-deep-learning/

[8] F. Rosenblatt, The Perceptron-a perceiving and recognizing automaton, Ed., Cornell Aeronautical Laboratory, 1957.

[9] F. J. G. Quesada, M. A. F. Graciani, M. T. L. Bonal, and M. A. Díaz-Mata, Aprendizaje con redes neuronales artificiales, Ensayos Rev. La Fac. Educ. Albacete, no. 9, 1994.

[10] Izaurieta, Fernando and Saavedra, Carlos,Redes neuronales artificiales, Departamento de Física, Universidad de Concepción Chile,2000
https://disi.unal.edu.co/ lctorress/RedNeu/LiRna003.pdf

[11] Redes neuronales
https://bootcampai.medium.com/redes-neuronales-13349dd1a5bb

[12] B.G.M. Vandeginste, J. Smeyers-Verbeke,Handbook of Chemometrics and Qualimetrics: Part B in Data Handling in Science and Technology, 1998
https://www.sciencedirect.com/topics/computer-science/simple-perceptron

[13] Minsky, M., y Papert, S. Perceptron: an introduction to computational geometry, 1969. . The MIT Press, Cambridge, MA.

[14] Rumelhart, D. E., Hinton, G. E., & Williams, R. J , Parallel distributed processing at : Further explorations in the microstructure of cognition, Wiley Online Library 1969

[15] ybenko, G. 1989. Approximation by superpositions of a sigmoidal function. Mathematics of control, signals and systems. 2, (1989),

[16] Gu, Shixiang Shane and Lillicrap, Timothy and Turner, Richard E and Ghahramani, Zoubin and Schölkopf, Bernhard and Levine, Sergey, Interpolated policy gradient: Merging on-policy and off-policy gradient estimation for deep reinforcement learning, Advances in neural information processing systems,30,2017

[17] Bouktif, Salah and Fiaz, Ali and Ouni, Ali and Serhani, Mohamed Adel, Optimal deep learning lstm model for electric load forecasting using feature selection and genetic algorithm: Comparison with machine learning approaches, Energies,2018, Multidisciplinary Digital Publishing Institute
https://www.mdpi.com/1996-1073/11/7/1636/htm

[18] Saiz Allende, Alejandro and others, Implementación de redes neuronales para conducción autónoma, 2021
https://repositorio.unican.es/xmlui/bitstream/handle/10902/21899/434502.pdf?sequence=1&isAllowed=y

[19] Centeno Franco, Alba,Deep learning,2019
https://idus.us.es/bitstream/handle/11441/90004/Centeno

[20] Aguado López, Inmaculada,Deep Learning: Redes Neuronales Convolucionales en R,2020
https://idus.us.es/bitstream/handle/11441/114957/GM

[21] Obregón, N and Fragala, F and Prada, LF,Redes neuronales artificiales en hidroinformática, SISTEMA,2017
$https://www.researchgate.net/profile/Nelson-Obregon-Neira/publication/228723557_Redes_neuronales_artifi$

[22] Cruz, Isis Bonet and Martínez, Sain Salazar and Abed, Abdel Rodríguez and Ábalo, Ricardo Grau and Lorenzo, Maria Matilde García, Redes neuronales recurrentes para el análisis de secuencias, Revista Cubana de Ciencias Informáticas,2007,Universidad de las Ciencias Informáticas
https://www.redalyc.org/pdf/3783/378343634004.pdf

[23] Hochreiter, Sepp and Schmidhuber, Jürgen,Long short-term memory, Neural computation,1997,MIT Press
https://ieeexplore.ieee.org/abstract/document/6795963

[24] Cho, K., Van Merrienboer, B., Bahdanau, D., and Bengio, Y. (2014). On the properties of neural machine translation: Encoder-decoder approaches. arXiv preprint arXiv:1409.1259

[25] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard y L. D. Jackel, "Backpropagation Applied to Handwritten Zip Code Recognition,"Neural Computation, vol. 1, n 4, pp. 541 - 551, 1989

[26] F. Sancho, .ªprendizaje por refuerzo: algoritmo Q Learning",Mar. 02, 2019.
http://www.cs.us.es/˜ fsancho/?e=109.

[27] Mathworks, Reinforcement Learning with MATLAB: Understanding the Basics and Setting Up the Environment", p. 20, 2020.

[28] Martínez, Susana Carvajal,Aplicación del Problema de la Ruina del Jugador en Opciones Financieras,2008
https://www.fcfm.buap.mx/hcs/tesis/Susana.pdf

[29] J. M. MarÃn, Çadenas de Markov."
http://halweb.uc3m.es/esp/Personal/personas/jmmarin/esp/PEst/tema4pe.pdf.

[30] S. S. Gu, T. Lillicrap, R. E. Turner, Z. Ghahramani, B. Schölkopf, and S. Levine, Interpolated policy gradient: Merging on-policy and off-policy gradient estimation for deep reinforcement learning, in Advances in neural information processing systems, 2017.

[31] S. Vasile, Sistema de conducción autónoma de vehículos mediante inteligencia artificial, 2019.

[32] M. Sanz, Aprendizaje por refuerzo. DQN: Q-Learning con redes neuronales | Medium, Apr. 03, 2020.
https://medium.com/@markelsanz14/introducción-al-aprendizaje-porrefuerzo-parte-3-q-learning-con-redes-neuronales-algoritmo-dqn-bfe02b37017f

[33] I. Grondman, L. Busoniu, G. A. D. Lopes, and R. Babuska, A survey of actor-critic reinforcement learning: Standard and natural policy gradients, IEEE Trans. Syst. Man, Cybern. Part C Applications Rev., vol. 42, no. 6, 2012.

[34] rlTrainingOptions, MathWorks, 2022
https://es.mathworks.com/help/reinforcement-learning/ref/rltrainingoptions.html

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR

Universidad
de Alcalá