



*ugr* | Universidad  
de **Granada**

TRABAJO FIN DE MÁSTER  
MÁSTER EN INGENIERÍA INFORMÁTICA

# Aprendizaje profundo por refuerzo en el entorno Google Football

**Autor**  
Ángel Murcia Díaz

**Director**  
Juan Gómez Romero



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE  
TELECOMUNICACIÓN

—  
Granada, 10 de diciembre de 2020









# Aprendizaje profundo por refuerzo en el entorno Google Football

---

**Autor**  
Ángel Murcia Díaz

**Director**  
Juan Gómez Romero



# Aprendizaje profundo por refuerzo en el entorno de Google Football

Ángel Murcia Díaz

**Palabras clave:** aprendizaje automático, aprendizaje por refuerzo, aprendizaje profundo, google research football environment, redes neuronales

## Resumen

En este trabajo se ha realizado una investigación acerca de una rama de la *Inteligencia artificial*, el *Aprendizaje por Refuerzo*, tanto de forma teórica como práctica. El objetivo que persigue el proyecto es doble: por un parte, estudiar los fundamentos teóricos de los algoritmos de *Aprendizaje por Refuerzo Profundo*, un tipo de *Aprendizaje por Refuerzo* que utiliza *Redes Neuronales*; por otra, comprobar si es posible aplicar estos algoritmos para realizar *Aprendizaje Progresivo* (*curriculum learning*, en inglés) en problemas que se organizan en niveles de dificultad creciente.

Para ello, en primer lugar se ha llevado a cabo una extensa revisión del marco teórico del *Aprendizaje por Refuerzo Profundo*, estudiando los conceptos clave, los algoritmos propuestos en los últimos años y las implementaciones públicas disponibles. En segundo lugar, se han implementado y ejecutado una serie de experimentos utilizando el entorno virtual *Google Research Football Environment*, que simula un partido de fútbol y permite controlar a los jugadores como si se tratara de un videojuego.

Las conclusiones más importantes del trabajo son: (1) que los algoritmos de *Aprendizaje por Refuerzo Profundo* son muy útiles incluso en problemas con espacios de estados y acciones complejos, (2) que el *Aprendizaje Progresivo* es más rápido y efectivo que el aprendizaje desde cero, incluso cuando las tareas más complejas no contienen completamente a las más simples.



# Deep reinforcement learning in the Google Football environment

Ángel Murcia Díaz

**Keywords:** machine learning, reinforcement learning, deep learning, google research football environment, neural networks

## Abstract

In this work, an investigation has been carried out on a branch of Artificial Intelligence, *Reinforcement Learning*, both theoretically and practically. The objective pursued by the project is twofold: on the one hand, to study the theoretical foundations of *Deep Reinforcement Learning* algorithms, a type of Reinforcement Learning that uses *Neural Networks*; on the other, to check if it is possible to apply these algorithms to carry out *Curriculum Learning* in problems that are organized in levels of increasing difficulty.

To do this, in the first place, an extensive review of the theoretical framework of *Deep Reinforcement Learning* has been carried out, studying the key concepts, the algorithms proposed in recent years and the public implementations available. Second, a series of experiments have been implemented and executed using the virtual *Google Research Football Environment*, which simulates a football match and allows players to be controlled as if it were a video game.

The most important conclusions of the work are: (1) that the *Deep Reinforcement Learning* algorithms are very useful even in problems with complex state and action spaces, (2) that *Curriculum Learning* is faster and more effective than learning from scratch, even when the most complex tasks do not completely contain the simplest ones.



---

Yo, **Ángel Murcia Díaz**, alumno de la titulación Máster en Ingeniería Informática de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI **26821637A**, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Máster en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Ángel Murcia Díaz

Granada a 10 de diciembre de 2020.



---

D. **Juan Gómez Romero**, Profesor del departamento de ciencias de la computación e inteligencia artificial de la Universidad de Granada.

**Informa:**

Que el presente trabajo, titulado *Aprendizaje por refuerzo profundo en el entorno de Google Football*, ha sido realizado bajo su supervisión por (**Ángel Murcia Díaz**), y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 10 de diciembre de 2020.

**El director:**

**Juan Gómez Romero**



# Dedicatoria

Me gustaría dedicar este trabajo fin de máster a la memoria de mi abuelo Pepe.

¡Hola abuelo, soy Angelillo!, me gustaría poder volver a darte un beso, decirte que te quiero y enseñarte este trabajo, porque sé con certeza que hubieras estado orgulloso de mí.

Mi abuelo era una persona maravillosa, en la que me veo reflejado en muchos de los aspectos de mi vida. Él nunca morirá porque siempre vivirá en mi recuerdo y en el recuerdo de las muchas personas que compartieron su vida con él, porque era una persona de las que trasmitten, de las que inspiran, de las que te hacen ver la vida de otra manera, de las que te enseñan todo lo que saben, de las que te hacen sonreír, de las que te hacen pensar que la vida realmente puede ser un viaje genial.

Él me enseñó a querer a todo el mundo que forma parte de tu vida por igual, diciéndonos constantemente: “Os quiero mucho a todos”, esa frase jamás la olvidare e intentaré trasmitirla a través de las generaciones futuras.

Él me enseñó que la vida no es solo alegría y color, que también hay que trabajar, esforzarse y formar tu propia historia, pero siempre utilizando el corazón. Las muchísimas anécdotas de la vida de mi abuelo me han servido como modelo de vida, a mi me encantaba escucharlas todas una y otra vez, y hoy daría lo que fuera porque me las volviera a contar.

Él me enseñó a no rendirme jamás, siendo mi fiel admirador y disfrutando cuando conseguía éxitos tanto a nivel personal como a nivel profesional.

Una de las cosas que se ha grabado a fuego en mi memoria era ver como disfrutaba simplemente cuando sabia que las personas eran felices. Le doy gracias a la vida por haberme dejado compartir tantos años con mi abuelo Pepe, estoy seguro que de no ser por él no sería como soy ahora, porque mi abuelo me ha sumado en todos los aspectos.

Fiñana el pueblo donde pase la mayoría de veranos de mi niñez, siempre tendrá un aroma a tí y a la abuela Chita, porque allí fui realmente feliz. Nunca olvidaré aquellos ratos en la “placeta” hablando de la vida, aquellos abrazos y besos en el ascensor, las charlas sobre las fiestas del pueblo, las noches buenas más mágicas de mi vida, ni mucho menos tu voz.

“Fui a un baile y me miré a un espejo y que rabia me dio verme tan viejo”, esa frase que tú tanto decías últimamente, lo cierto es que una vez más tenías muchísima razón, quizá intentábamos no creérnoslo, por la gran vitalidad que atesorabas dentro, porque si algo se daba cuenta uno cuando te miraba es que la vida te encantaba y disfrutabas con cada pequeño detalle de la misma, por eso también me enseñaste a querer levantarme cada mañana con ganas de vivir y de ser feliz. A pesar de que intentábamos no creérnoslo con todas nuestras fuerzas, era verdad, pero puedo decir que he sido muy feliz a tu lado todos estos años y que tu vida tuvo sentido hasta el último día, por eso intento recordarte con alegría cada día de mi vida, los ratos que he pasado contigo no los cambiaría por nada del mundo.

Abuelo dejaste huella en nuestros corazones y por eso te quiero dar las gracias y decirte que serás eterno para siempre, porque en cada acción de cualquiera de nosotros siempre habrá una “gótilla” de ti.

Te quiero abuelo, tu nieto Angelillo.

# Agradecimientos

Me gustaría agradecer a todas las personas que me han apoyado tanto de manera emocional como de manera académica.

En primer lugar, quiero dar las gracias a todos los profesores que he tenido durante todos estos años tanto en el grado como en el máster, gracias a ellos he aprendido mucho. En especial a mi director Juan, por haberme guiado, ayudado, apoyado en todo momento y confiado en mí para el proyecto, sin tu ayuda no habría logrado terminar este trabajo.

También, quiero dar las gracias a mi maravillosa familia, por dedicarme siempre su tiempo, apoyarme en todo momento y darme la oportunidad de seguir estudiando lo que me gusta haciendo en ocasiones un gran esfuerzo. Mamá gracias por hacer que sienta que soy la persona que más quieras del mundo, por ayudarme siempre que te lo pido dando igual la hora del día a la que sea y por cuidarme tanto. Papá, gracias por ser “superpapá” en todo momento y por desde pequeño hacer que me interese por el maravilloso mundo de la informática, no sé que haría sin tí. También quiero agradecer a mis abuelas, por compartir vuestra sabiduría y cariño conmigo, los momentos que hemos pasado y que quedan por pasar juntos ya forman parte de los mejores momentos de mi vida. Así como a mis abuelos, os echo muchísimo de menos, solo os puedo decir gracias, gracias y gracias. A todos mis primos, en especial a mi primo Fran y mi prima Carmen, por hacerme sentir como un hermano pequeño para ellos, preocupándose por mí durante los 365 días del año y compartiendo su tiempo conmigo. En general a toda mi familia, que me ha hecho disfrutar de la vida desde pequeño y ser buena persona, muchas gracias a todos.

Por supuesto, quiero dar las gracias a mi novia, mi querida Elenita, que me ha soportado miles de días complicados durante estos increíbles 4 años y me ha enseñado a amar, así como a aprender que todo en la vida se hace mejor en buena compañía.

Por último, quiero dar las gracias a mis amigos, por escucharme en todo momento, darme consejos y ayudarme en lo que podían. Pero sobretodo por compartir los mejores y más divertidos momentos de este curso a mi lado, aunque no han sido demasiados por culpa del covid, nunca los olvidaré.

Tanto a mis amigos de toda la vida, como a los nuevos amigos del máster.

En definitiva, gracias a todas las personas que han pasado por mi vida  
y han aportado una “gotita” a lo que soy. 1

# Índice general

<b>1. INTRODUCCIÓN</b>	<b>1</b>
1.1. INTRODUCCIÓN AL TEMA . . . . .	1
1.2. MOTIVACIÓN PERSONAL . . . . .	4
1.3. OBJETIVOS . . . . .	5
1.3.1. OBJETIVOS GENERALES . . . . .	5
1.3.2. OBJETIVOS PERSONALES . . . . .	6
1.4. RESTRICCIONES TÉCNICAS Y DE GESTIÓN . . . . .	6
1.4.1. FACTORES DATO . . . . .	7
1.4.2. FACTORES ESTRATEGICOS . . . . .	7
1.5. PROGRAMA DE TAREAS . . . . .	8
1.6. PLANIFICACIÓN DEL PROYECTO . . . . .	9
1.7. RECURSOS DEL PROYECTO . . . . .	10
1.7.1. RECURSOS HARDWARE . . . . .	10
1.7.2. RECURSOS SOFTWARE . . . . .	11
1.7.3. RECURSOS HUMANOS . . . . .	11
1.8. ESTRUCTURA DEL DOCUMENTO . . . . .	12
<b>2. MARCO TEÓRICO</b>	<b>13</b>
2.1. INTELIGENCIA ARTIFICIAL . . . . .	13
2.2. APRENDIZAJE AUTOMÁTICO . . . . .	16
2.2.1. APLICACIONES REALES . . . . .	17
2.3. APRENDIZAJE POR REFUERZO . . . . .	26
2.3.1. COMPONENTES . . . . .	27
2.3.2. PROCESOS DE DECISIÓN DE MARKOV (MDPs) .	29
2.3.3. POLÍTICA . . . . .	33
2.3.4. MÉTODOS PARA EVALUAR Y MEJORAR LAS POLÍTICAS . . . . .	39
2.4. REDES NEURONALES . . . . .	43
2.4.1. FUNCIONES DE ACTIVACIÓN . . . . .	45
2.4.2. FUNCIONES DE SALIDA . . . . .	46
2.4.3. FUNCIONES DE ERROR . . . . .	47
2.4.4. ALGORITMOS DE OPTIMIZACIÓN . . . . .	47
2.5. REDES NEURONALES CONVOLUCIONALES . . . . .	49

2.5.1. CAPA CONVOLUCIONAL . . . . .	50
2.5.2. CAPA POOLING . . . . .	51
2.5.3. CAPA FLATTEN . . . . .	52
2.6. APRENDIZAJE POR REFUERZO . . . . .	52
PROFUNDO . . . . .	52
2.6.1. MÉTODOS BASADOS EN VALOR: DEEP Q-NETWORK (DQN) . . . . .	53
2.6.2. MÉTODOS BASADOS EN POLÍTICA . . . . .	57
2.6.3. MÉTODOS ACTOR-CRÍTICO . . . . .	59
2.6.4. PROXIMAL POLICY OPTIMIZATION (PPO2) . . . . .	62
<b>3. HERRAMIENTAS . . . . .</b>	<b>65</b>
3.1. ENTORNO: GOOGLE RESEARCH FOOTBALL ENVIRONMENT . . . . .	65
3.1.1. INTRODUCCIÓN AL ENTORNO . . . . .	65
3.1.2. FOOTBALL ENGINE . . . . .	68
3.1.3. FOOTBALL BENCHMARK . . . . .	69
3.1.4. FOOTBALL ACADEMY . . . . .	71
3.1.5. MEDIO AMBIENTE DEL ENTORNO . . . . .	80
3.1.6. CREAR UN ENTORNO: PARÁMETROS . . . . .	84
3.1.7. JUGAR EN MODO MANUAL . . . . .	85
3.1.8. CURIOSIDAD: COMPETICIÓN KAGGLE . . . . .	86
3.2. GOOGLE COLAB . . . . .	87
3.3. BIBLIOTECAS DE APRENDIZAJE POR REFUERZO . . . . .	89
3.3.1. POSIBLES ALTERNATIVAS . . . . .	89
3.4. MULTI-ENTORNOS . . . . .	91
<b>4. METODOLOGÍA DEL PROCESO EXPERIMENTAL . . . . .</b>	<b>93</b>
4.1. PRIMERAS PRUEBAS . . . . .	93
4.2. PRUEBAS CON DIFERENTES ALGORITMOS . . . . .	95
4.2.1. SELECCIÓN DE ALGORITMOS A UTILIZAR . . . . .	95
4.2.2. PARÁMETROS DE LOS ALGORITMOS . . . . .	96
4.3. EXPERIMENTACIÓN BÁSICA . . . . .	99
4.4. EXPERIMENTACIÓN AVANZADA . . . . .	101
<b>5. EXPERIMENTACIÓN Y DISCUSIÓN DE LOS RESULTADOS . . . . .</b>	<b>109</b>
5.1. EXPOSICIÓN DE RESULTADOS . . . . .	109
5.1.1. RESULTADOS DE EXPERIMENTACIÓN BÁSICA . . . . .	109
5.1.2. RESULTADOS DE EXPERIMENTACIÓN AVANZADA . . . . .	112
5.2. ANÁLISIS DE RESULTADOS . . . . .	114
5.2.1. ANÁLISIS DE EXPERIMENTACIÓN BÁSICA . . . . .	115
5.2.2. ANÁLISIS DE EXPERIMENTACIÓN AVANZADA . . . . .	120

---

5.2.3. DISCUSIÓN DE RESULTADOS . . . . .	125
<b>6. CONCLUSIONES Y FUTURAS MEJORAS</b>	<b>131</b>
6.1. CONCLUSIONES GENERALES . . . . .	131
6.2. FUTURAS MEJORAS . . . . .	132
<b>A. ANEXO DE INFORMÁTICA</b>	<b>135</b>
A.1. De forma local . . . . .	135
A.2. De forma remota: Google Colab . . . . .	137
<b>B. ANEXO DE VÍDEOS</b>	<b>139</b>



# Índice de figuras

1.1.	Tipos de Aprendizaje Autómatico . . . . .	3
1.2.	Estructura de las Redes Neuronales Artificiales . . . . .	3
1.3.	Imagen de coches autónomos . . . . .	4
2.1.	Proceso de <i>Aprendizaje Autómatico</i> . . . . .	16
2.2.	Imagen de Alexa, extraída de powerplanetaonline . . . . .	18
2.3.	Icono semáforo, extraído de freepik . . . . .	19
2.4.	Icono de Redes sociales, extraído de concepto.de . . . . .	22
2.5.	Icono de Detección de Fraude, extraído de itravelservices . .	24
2.6.	Reconocimiento de voz, extraído de linuxadictos . . . . .	25
2.7.	Coche autónomo, extraído de motorpasion . . . . .	26
2.8.	Esquema de componentes del proceso de <i>Aprendizaje por Re-fuerzo</i> , extraido de wikipedia . . . . .	28
2.9.	Gráfica de recompensa en relación con el tiempo influenciado por el factor de descuento, extraída de [24] y modificada . .	32
2.10.	Estructura <i>Red Neuronal Artificial</i> . . . . .	44
2.11.	Función de activación Sigmoide . . . . .	45
2.12.	Proceso del descenso del gradiente estocástico, imagen extraída de cs.us.es . . . . .	49
2.13.	Convolución: aplicación de filtro o kernel a una matriz de píxeles (imagen) . . . . .	50
2.14.	Aplicación de capa Max Pooling a mapa de características .	51
2.15.	Capa Flatten en CNN, imagen extraída de mc.ai . . . . .	52
2.16.	Red Neuronal Artificial para DQN en un problema con valores discretos . . . . .	54
2.17.	Red Neuronal Artificial para DQN en un problema con acciones continuas . . . . .	55
2.18.	Red Neuronal Artificial para métodos basados en políticas .	58
2.19.	Funcionamiento método Actor-Critic . . . . .	59
2.20.	Método A3C . . . . .	60
2.21.	Método A2C . . . . .	61
3.1.	Logo Google AI . . . . .	65

3.2.	Captura del entorno en funcionamiento . . . . .	66
3.3.	Componentes de <i>Google Research Football Environment</i> . . . . .	67
3.4.	Gráfica de <i>Google AI</i> utilizando <i>Google Benchmark</i> : Reward = Scoring, Parte 1, extraída de [19] . . . . .	70
3.5.	Gráfica de <i>Google AI</i> utilizando <i>Google Benchmark</i> : Reward = Checkpoint, Parte 2, extraída de [19] . . . . .	70
3.6.	Escenario: academy_3_vs_1_with_keeper . . . . .	72
3.7.	Escenario: academy_corner . . . . .	73
3.8.	Escenario: academy_counterattack_easy . . . . .	73
3.9.	Prueba de <i>Google AI</i> escenarios con <i>PPO2</i> ; Rewards = Checkpoint, Parte 1, extraída de [19] . . . . .	75
3.10.	Prueba de <i>Google AI</i> escenarios con <i>PPO2</i> ; Rewards = Checkpoint, Parte 2, extraída de [19] . . . . .	75
3.11.	Prueba de <i>Google AI</i> escenarios con <i>IMPALA</i> ; Rewards = Checkpoint, Parte 1, extraída de [19] . . . . .	76
3.12.	Prueba de <i>Google AI</i> escenarios con <i>IMPALA</i> ; Rewards = Checkpoint, Parte 2, extraída de [19] . . . . .	76
3.13.	Prueba de <i>Google AI</i> escenarios con <i>PPO2</i> ; Rewards = Scoring, Parte 1, extraída de [19] . . . . .	77
3.14.	Prueba de <i>Google AI</i> escenarios con <i>PPO2</i> ; Rewards = Scoring, Parte 2, extraída de [19] . . . . .	77
3.15.	Prueba de <i>Google AI</i> escenarios con <i>IMPALA</i> ; Rewards = Scoring, Parte 1, extraída de [19] . . . . .	78
3.16.	Prueba de <i>Google AI</i> escenarios con <i>IMPALA</i> ; Rewards = Scoring, Parte 2, extraída de [19] . . . . .	78
3.17.	Juego del 1992: Sensible Soccer . . . . .	86
3.18.	Escudo Manchester City F.C. . . . .	86
4.1.	Partido por defecto de <i>Google Research Football Environment</i>	94
4.2.	Proceso experimental básico de forma visual . . . . .	101
4.3.	Conjunto 1 de escenarios de dificultad progresiva . . . . .	103
4.4.	Conjunto 2 de escenarios de dificultad progresiva . . . . .	104
4.5.	Conjunto 3 de escenarios de dificultad progresiva . . . . .	105
5.1.	Gráfica correspondiente al entrenamiento del agente en academy_counterattack_hard utilizando PPO2 e IMPALA . . . .	118
5.2.	Gráfica correspondiente al entrenamiento del agente en academy_3_vs_1_with_keeper utilizando PPO2 e IMPALA . . . .	118
5.3.	Gráfica correspondiente al entrenamiento del agente en academy_run_pass_and_shoot_with_keeper utilizando PPO2 e CNN119	
5.4.	Gráfica correspondiente al entrenamiento del agente en 1_vs_1_easy utilizando A2C e CNN . . . . .	119
5.5.	Gráfica correspondiente al entrenamiento del agente en 11_vs_11_hard_stochastic utilizando A2C e CNN . . . . .	120

5.6. Gráfica de comparación correspondiente al entrenamiento de forma simple . . . . .	123
5.7. Gráfica de comparación correspondiente al entrenamiento utilizando aprendizaje progresivo sobre el conjunto 1 . . . . .	124
5.8. Gráfica de comparación correspondiente al entrenamiento utilizando aprendizaje progresivo sobre el conjunto 3 . . . . .	124
5.9. Gráfica de comparación correspondiente al entrenamiento utilizando aprendizaje progresivo de forma cíclica sobre el conjunto 1 . . . . .	125



# Índice de tablas

1.1.	Planificación del proyecto esperada . . . . .	9
3.1.	Resultados Prueba multi-entornos . . . . .	92
4.1.	Resultados Prueba de tipo de imputs . . . . .	97
4.2.	Resultados Prueba de tipo de recompensas . . . . .	98
5.1.	“eprewmean” PPO2 utilizando <i>SubprocVecEnv</i> con 8 entornos	110
5.2.	“eprewmean” A2C utilizando <i>SubprocVecEnv</i> con 8 entornos	110
5.3.	“eprewmean” DQN utilizando 1 entorno simple . . . . .	111
5.4.	Mejores resultados de cada combinación algoritmo-red subyacente . . . . .	111
5.5.	Resultados de conjunto 1 de escenarios de dificultad progresiva	112
5.6.	Resultados de conjunto 2 de escenarios de dificultad progresiva	112
5.7.	Resultados obtenidos en el conjunto de escenarios de dificultad progresiva 3 . . . . .	113
5.8.	Resultados obtenidos en el conjunto de escenarios de dificultad progresiva 1 . . . . .	113
5.9.	Resultados obtenidos en el conjunto de escenarios de dificultad progresiva 2 . . . . .	114



# Índice de algoritmos

1.	Algoritmo Montecarlo para mejorar políticas, fuente ccc.inaoep	40
2.	Algoritmo SARSA, fuente sedici.unlp.ar . . . . .	41
3.	Algoritmo Q-Learning, fuente sedici.unlp.edu . . . . .	43
4.	Algoritmo DQN . . . . .	57



# Capítulo 1

# INTRODUCCIÓN

## 1.1. INTRODUCCIÓN AL TEMA

Las necesidades de la sociedad moderna y las empresas han cambiando con el paso de los años, lo que se traduce en que cada vez los robots, máquinas y ordenadores están más capacitados, refiriéndose tanto a la parte hardware como a la parte software, de forma que son capaces de desarrollar funciones que antes no eran ni siquiera imaginables.

Actualmente *la Inteligencia Artificial* está cambiando la forma en que vivimos y trabajamos, incluso más que cualquier tecnología desde el nacimiento de internet, la cuál ya está a nuestro alrededor, por ejemplo escrito en el software en nuestros teléfonos móviles, en nuestros coches, hogares o en el software de negocio que utilizamos diariamente en el trabajo, de tal forma que ayuda a acceder a la información y a tomar mejores decisiones de forma más rápida.

La potencia de *la Inteligencia Artificial* es que sus usos son prácticamente ilimitados. Por ejemplo, se utiliza cuando una tarea requiere analizar rápidamente y derivar la comprensión de los datos o cuando una tarea requiere identificar tendencias y/o anomalías en grandes conjuntos de datos [22].

**La Inteligencia Artificial (IA)** o *Artificial Intelligence (AI)* en inglés, es un subcampo de la ciencia de la computación. Esta permite a las computadoras hacer cosas que normalmente sólo podrían hacer los humanos, concretamente actividades relacionadas con la inteligencia, teniendo el objetivo de resolver todo tipo de problemas como: toma de decisiones, percepción visual o reconocimiento de voz [34].

**El Aprendizaje Automático (AA)** o *Machile Learning (ML)* en inglés, es una rama de la *Inteligencia Artificial* que tiene como objetivo

el desarrollo de técnicas que le permita a los ordenadores “aprender”, es decir, crear algoritmos que sean capaces de generalizar comportamientos y reconocer patrones a partir de información suministrada en forma de ejemplos. Por lo que es un proceso de *inducción del conocimiento* (método que permite generalizar a partir de casos particulares).

Dentro del *Aprendizaje Automático* hay una gran variedad de técnicas diferentes, que se pueden agrupar en los siguientes grupos [31]:

- **Aprendizaje supervisado:** este tipo de aprendizaje se basa en generar una función que establece una correspondencia entre las entradas y las salidas de un sistema, donde la base de conocimientos se forma con ejemplos etiquetados por el experto. Por ejemplo el algoritmo KNN.
- **Aprendizaje no supervisado:** este tipo de aprendizaje se basa en generar una función que relaciona las entradas con las salidas, pero el proceso de modelado de esta función se realiza sobre ejemplos que solo cuentan con entradas, es decir, no están etiquetados por lo que el sistema busca la forma de conseguir reconocer los patrones. Por ejemplo el algoritmo K-medias.
- **Aprendizaje semisupervisado:** este tipo de aprendizaje es una combinación de los dos aprendizajes anteriores, es decir, el aprendizaje supervisado y el aprendizaje no supervisado.
- **Aprendizaje por refuerzo:** este tipo de aprendizaje se basa en el entrenamiento de agentes que aprenden observando el mundo que les rodea y con un continuo flujo de información en las dos direcciones: del mundo (entorno) que lo rodea al agente que realiza las acciones y al revés. Todo esto se realiza mediante un proceso de ensayo-error, reforzando las acciones que conducen a una solución buena y penalizando las acciones que conducen a una solución mala. Por ejemplo el algoritmo PPO2.
- **Transducción:** este tipo de aprendizaje es similar al aprendizaje supervisado pero a diferencia de éste no trata de formar una función para relacionar las entradas con las salidas sino que simplemente trata de predecir las salidas basándose en la similitud con los ejemplos de entrada.
- **Aprendizaje multi-tarea:** este tipo de aprendizaje engloba todos aquellos métodos que usan el conocimiento previamente aprendido por el sistema, de la manera que sea, para otros/múltiples fines.

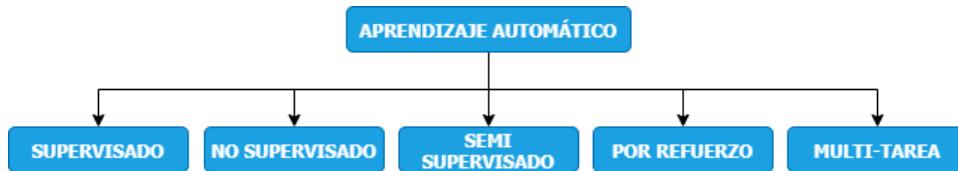


Figura 1.1: Tipos de Aprendizaje Autómatico

Dentro de todas las categorías mencionadas anteriormente, tenemos las técnicas de **Aprendizaje profundo (AP)** o *Deep Learning (DL)* en inglés, que son las técnicas que utilizan las **Redes Neuronales Artificiales (RNA)** o *Neural Network (NN)* en inglés.

Las *Redes Neuronales Artificiales* son técnicas que tratan de emular de forma aproximada los sistemas cognitivos de los seres vivos, entre ellos los del hombre, concretamente el funcionamiento del cerebro, es decir, la interacción entre las diferentes neuronas, que se modulan a través de pesos. Estas técnicas son ideales cuando se quiere resolver un problema con la mejor solución posible pero sin saber como piensa el agente. Por ejemplo, estas ayudan a resolver problemas complejos como el de reconocimiento de rostros [3].

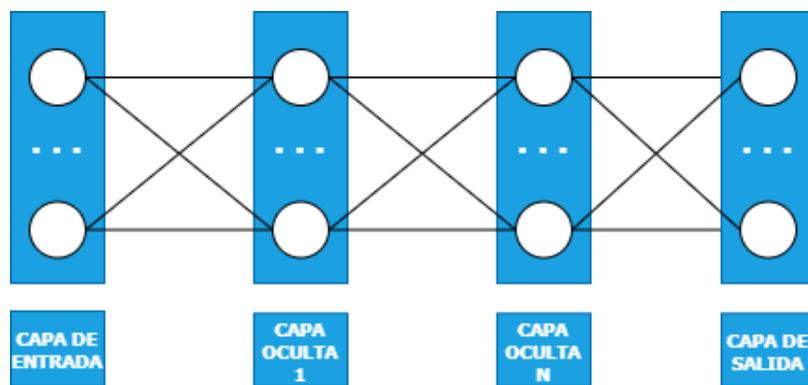


Figura 1.2: Estructura de las Redes Neuronales Artificiales

En este Trabajo Fin de Máster se utiliza la técnica de *Aprendizaje por Refuerzo Profundo*, por lo que se mezcla el *Aprendizaje por Refuerzo* con las *Redes Neuronales Artificiales*, para realizar una investigación y experimentación exhaustiva de la aplicación de este tipo de algoritmos en el entorno de **Google Research Football Environment**, así como también se explica en su máximo detalle el propio entorno, características del mismo y sus diferentes configuraciones.

## 1.2. MOTIVACIÓN PERSONAL

Mi interés en la *Inteligencia Artificial* surgió cuando comencé mis estudios en la universidad, puesto que cuando se entra al mundo de la informática no se tarda en empezar a escuchar hablar de temas relacionados como: robots que son capaces de “aprender” como las personas humanas o coches que son capaces de conducir sin ayuda humana, es decir, solos. ¿Quién no se siente atraído por estos temas?

Con el paso del tiempo he seguido investigando y aprendiendo sobre esta área de la informática, tanto en el grado y el máster en diferentes asignaturas como por mi cuenta leyendo artículos interesantes y buscando información al respecto en internet. Una de las subramas que me llamó la atención nada más empezar a investigar fue la utilizada en el presente trabajo: el *Aprendizaje por Refuerzo* y dentro de este el *Aprendizaje por Refuerzo Profundo*. Uno de los puntos positivos de esta subrama es que sus aplicaciones son prácticamente ilimitadas; algunos ejemplos son: mejora en la gestión de recursos, asistencia de inversión en el sistema financiero, personalización en eCommerce, optimización de tratamientos médicos de largo recorrido, robots, drones, coches autónomos o videojuegos.

La idea de los coches autónomos me parece totalmente revolucionaria, ¿Te imaginas poder hacer un viaje de Córdoba a Almería mientras estás durmiendo y el coche autónomo conduce solo?

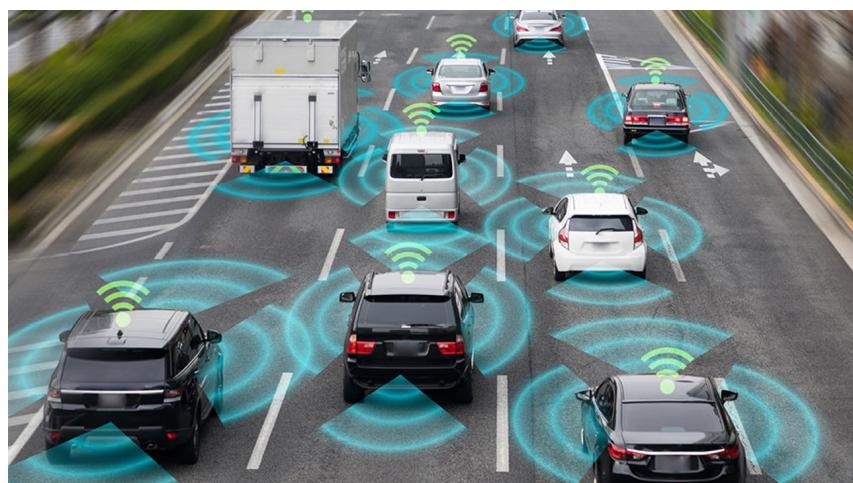


Figura 1.3: Imagen de coches autónomos

En el área de los videojuegos me asombra que se ha conseguido que los agentes entrenados utilizando técnicas de *Aprendizaje por Refuerzo Profundo* consigan jugar a juegos icónicos como juegos de la consola Atari, el antiguo juego de *Go* o videojuegos que se juegan de forma profesional como

es el caso de *Dota2* o *Starcraft2*. Incluso un agente llamado *AlphaGo* se ha convertido en “el mejor jugador de go del mundo”, puesto que en marzo de 2016 este participó en un torneo contra el campeón mundial Lee Sedol, y terminó ganando. Antes de participar en este torneo el agente fue entrenado con 30 millones de partidas [19].

Otra de mis motivaciones para elegir este proyecto fue el entorno en el que se trabajaría, es decir, *Google Research Football Environment*, puesto que soy un gran apasionado del deporte rey y desde pequeño me han gustado los juegos de este deporte. La primera vez que vi la parte gráfica del entorno me llenó de nostalgia puesto que se parecía extremadamente al juego que llevé jugando desde mi infancia *Football Manager*. También la idea de que un equipo de fútbol en el entorno pueda jugar siguiendo una inteligencia previamente entrenada, fue excepcional.

Por lo tanto la fusión tanto del *Aprendizaje por Refuerzo* con el entorno *Google Research Football Environment* significó que no me lo pensara ni un segundo en cuanto a la elección de mi *Trabajo Fin de Máster* se refería.

### 1.3. OBJETIVOS

#### 1.3.1. OBJETIVOS GENERALES

Los objetivos principales de este proyecto son:

- Investigar en profundidad el *Aprendizaje por Refuerzo Profundo*, para entender a la perfección todos sus conceptos, así como para entender la forma en que funcionan los diferentes algoritmos de este tipo de aprendizaje.
- Investigar y entender el entorno con el que se trabaja en este proyecto, es decir, *Google Research Football Environment*. Aquí se incluye su interfaz gráfica, el funcionamiento interno del entorno, la posibilidad de jugar de forma manual, las diferentes acciones que pueden realizar los jugadores, los diferentes espacios de estados con los que pueden aprender los algoritmos de *Aprendizaje por Refuerzo*, los diferentes escenarios disponibles, etc.
- Realizar un proceso experimental utilizando diferentes algoritmos de *Aprendizaje por Refuerzo* sobre *Google Research Football Environment*, variando diferentes parámetros del algoritmo y los diferentes escenarios del entorno. Como último paso de esta experimentación utilizar agentes entrenados sobre escenarios del entorno más sencillos (por ejemplo un entorno de correr y tirar contra un portero) sobre otros escenarios

del entorno más complejos (por ejemplo un entorno de tres atacantes contra un portero).

Una vez explicados los objetivos principales, se definen otra serie de objetivos:

- Comprender los conceptos acerca del funcionamiento interno de las *Redes Neuronales Artificiales*.
- Comprender el funcionamiento interno de los diferentes algoritmos de *Aprendizaje por Refuerzo*.
- Investigar y probar sobre las posibles bibliotecas que proporcionan algoritmos de *Aprendizaje por Refuerzo*, para seleccionar una de ellas de cara a la realización del proyecto.
- Comparar el funcionamiento de los algoritmos de *Aprendizaje por Refuerzo* sobre el entorno *Google Research Football Environment* tanto en mi propio ordenador como en la plataforma *Google Cloud*.
- Realizar un análisis del proceso experimental desarrollado y extraer conclusiones en función de este análisis de los resultados.
- Confeccionar una lista con posibles tareas que podrá hacer otra persona partiendo de este trabajo, es decir, el posible trabajo futuro.

### **1.3.2. OBJETIVOS PERSONALES**

Aparte de los objetivos generales anteriormente enunciados, hay una serie de objetivos a nivel personal que deseo cumplir una vez finalice el proyecto:

- Enfrentarme a una situación real y poner en práctica los conocimientos adquiridos durante el grado y el máster de Ingeniería Informática.
- Adquirir la capacidad de afrontar problemas y buscar soluciones adecuadas por mí mismo, realizando investigaciones, así como consultando las dudas que puedan surgirme.

## **1.4. RESTRICCIONES TÉCNICAS Y DE GESTIÓN**

En este apartado se detallan tanto los factores dato como los factores estratégicos. En primer lugar, se detallan los factores dato, que son aquellas

restricciones que vienen impuestas por la propia naturaleza del problema. En segundo lugar, se describen aquellas restricciones que han sido seleccionadas con el fin de obtener la solución óptima al problema tratado, es decir los factores estratégicos.

#### 1.4.1. FACTORES DATO

- El proyecto deberá de cumplir los objetivos anteriormente detallados.
- Para el desarrollo del proyecto se contará con recursos propios de los miembros del equipo de desarrollo, recursos consumidos por “Google Colab”, así como con recursos disponibles en la Universidad de Granada.
- Como entorno de *Aprendizaje por Refuerzo Profundo* se tendrá que utilizar: *Google Research Football Environment*.

#### 1.4.2. FACTORES ESTRATEGICOS

- La experimentación será desarrollada en el lenguaje de programación *Python*, las razones han sido:
  - Es un lenguaje multiplataforma.
  - Tiene multitud de frameworks de gran utilidad.
  - Su uso es recurrente para tratar con problemas relacionados con la *Inteligencia Artificial*.
  - Cuenta con infinidad de librerías a su disposición que son muy sencillas de instalar, contando entre ellas a librerías que son de utilidad para el proyecto, es decir, aquellas que ofrecen algoritmos de *Aprendizaje por Refuerzo*.
  - Su sintaxis es de gran calidad lo que implica limpieza y estructuración del código.
  - Y por último, por mi grado de conocimiento de este lenguaje, aunque en principio no es muy abundante pero si es el necesario como para tomarlo como punto de partida y aprender más.
- Se utilizará un entorno de programación, donde se realiza el desarrollo, se depurará y se ejecutará el programa durante su creación, concretamente “PyCharm”. Puesto que es un entorno muy sencillo y práctico, que tiene una serie de ventajas como:
  - Es un editor inteligente, que permite completar código con algunos atajos de teclado. También permite navegar a través de

nuestro código, saltando entre las clases y métodos creados, haciendo el flujo de trabajo mucho más dinámico.

- Tiene la posibilidad de refactorizar el código, es decir, modificar el código sin comprometer la ejecución del mismo.
  - Cuenta con una infinidad de temas y plugin por lo que puedes personalizarlo a tu gusto.
- Se utilizará “overleaf.com” para crear la necesaria documentación en LaTex, debido a la facilidad, limpieza y profesionalidad que aporta LaTeX a la hora de generar documentación.
  - Se utilizará *Google Colab* como entorno de ejecución remoto, puesto que permite ejecutar las celdas de código Notebook. Esta decisión esta basada en la velocidad que proporciona el Hardware proporcionado por Google sobretodo si cuentas con la versión Pro de *Google Colab*.
  - Se utilizará el navegador Google Chrome para buscar la información necesaria durante el desarrollo del proyecto, así como para buscar las bibliotecas y herramientas necesarias.
  - Para la realización total de este proyecto se ha decidido utilizar como sistema operativo Ubuntu 19.04.

## 1.5. PROGRAMA DE TAREAS

Se distinguen las siguientes fases a llevar a cabo durante el proceso de desarrollo de este proyecto:

- **Estudio, formación e investigación:** Esta primera fase del proyecto es una de las más importantes del mismo. En ella se realiza un extenso estudio previo de los conceptos teóricos que intervienen en el desarrollo del proyecto, así como de las herramientas y las tecnologías que se utilizan durante el proyecto. Concretamente se realizarán las siguientes tareas: estudio de los conceptos teóricos sobre *Inteligencia Artificial* y subramas de la misma, estudio de diferentes algoritmos de *Aprendizaje por Refuerzo* con la idea de ver las ventajas y desventajas de cada una de ellas, estudio y selección del mejor lenguaje de programación para el desarrollo del proyecto, comprensión del entorno a utilizar durante el proyecto y bibliotecas.
- **Preparación:** Esta fase se prepararan los ordenadores con los software necesarios, así como una cuenta en *Google Colab* que queda lista para poner el código a ejecutar.

- **Puesta en marcha:** Esta fase consiste en empezar a lanzar las primeras pruebas con el entorno y con las librerías de *Aprendizaje por Refuerzo* para afianzar los conceptos teóricos anteriormente comprendidos, así como para comprobar que funciona todo correctamente.
- **Diseño del proceso experimental:** Esta fase consiste en el diseño del proceso experimental a seguir durante el desarrollo del proyecto. Aquí se incluye la selección de qué escenarios del entorno se van a utilizar, qué algoritmos de *Aprendizaje por Refuerzo* se van a utilizar o con qué configuraciones de parámetros se van a realizar las pruebas.
- **Implementación:** Esta fase consiste en la implementación del software y de las pruebas propuestas en el diseño experimental.
- **Documentación:** Esta fase consiste en realizar la documentación de todo lo que se ha ido realizando durante el proyecto, es decir, del desarrollo del proyecto, por tanto se realiza de forma paralela al proyecto, desde el principio hasta el final del mismo. Para dicha documentación se utiliza LaTeX. La documentación se divide en varios formatos, uno a nivel de desarrollador para explicar todos los pasos que se han ido realizando durante el proyecto (memoria del proyecto) y otro a nivel de usuario para que cualquier persona pueda poner a funcionar el código de este proyecto tanto de forma local como de forma remota utilizando Google Colab. Esta parte llamada anexo de informática se encuentra al final de la memoria del proyecto.

## 1.6. PLANIFICACIÓN DEL PROYECTO

Fase de desarrollo	06	07	08	09	10	11	12	Suma
Estudio, formación e investigación	25	15	12	2	2	2	2	60
Preparación	2	3						5
Puesta en marcha			20	10				30
Diseño del proceso experimental			10	25	20			55
Implementación				20	50	25		95
Documentación		10	5	5	5	15	15	45
<b>Total</b>	<b>27</b>	<b>28</b>	<b>47</b>	<b>62</b>	<b>77</b>	<b>42</b>	<b>17</b>	<b>300</b>

Tabla 1.1: Planificación del proyecto esperada

Según la normativa, cada crédito corresponde a 10 horas de clase, y 15 horas de estudio, por tanto cada crédito del proyecto al no ser presencial, se

corresponde con 25 horas de trabajo.

El *Trabajo Fin de Máster* está formado por 12 créditos correspondientes a 300 horas en total. La planificación temporal de cada una de las fases de desarrollo, en lo que a número de horas se refiere, se refleja en la tabla 1.1, cada una de las filas de este cuadro representa cada una de las fases de desarrollo, mientras que cada una de las columnas de este cuadro representa un mes del año 2020 (en formato numérico, por ejemplo Julio es “07”), en los cuales se ha estado trabajando.

Aunque la ejecución de cada una de las pruebas propuestas en el proceso experimental no se incluya como fase de desarrollo ni por consiguiente se incluya en esta planificación temporal, cabe destacar que cada una de las pruebas por separado puede durar incluso días. Me parecía importante destacarlo puesto que limita mucho la realización del proyecto, teniendo en muchos casos que realizar tiempos muy grandes de espera para después poder continuar, como anécdota indicar que muchas de las pruebas las he dejado puestas antes de dormir para que se ejecuten durante la noche.

## **1.7. RECURSOS DEL PROYECTO**

Los recursos humanos que se utilizarán para la realización de este proyecto serán: recursos software, recursos hardware y recursos humanos.

### **1.7.1. RECURSOS HARDWARE**

Los recursos hardware disponibles son dos ordenadores, un ordenador de sobremesa que será el que use normalmente y un ordenador portátil que se use en situaciones excepcionales cuando no tenga acceso al ordenador de sobremesa.

- Ordenador de sobremesa donde se ha realizado la mayor parte del proyecto es un MSI MS-7693, sus características son:
  - Procesador: AMD FX-8350 8 núcleos (4 GHz).
  - Memoria RAM: 16GB 1600MHZ DDR3 (2x8GB).
  - Tarjeta Gráfica: AMD R9 290 Tri-X 4GB GDDR5 Sapphire.
  - Disco Duro: Disco duro 1TB (7200 rpm S-ATA).
- Ordenador portátil utilizado en situaciones excepcionales es un MSI GL62 6QF-1229XES, y sus características son:
  - Procesador: Intel® Core i7-6700HQ (2.6 GHz, 6 MB).

- Memoria RAM: 8GB DDR4 SODIMM (1x8GB).
- Tarjeta Gráfica: Nvidia GeForce GTX 960M 2GB GDDR5.
- Disco Duro: Disco duro 1TB HDD (7200 rpm S-ATA) + 256GB SSD (256GB \*1 M.2 SATA).

### 1.7.2. RECURSOS SOFTWARE

El proyecto ha sido realizado bajo un sistema operativo Ubuntu 19.04.

Para el desarrollo y experimentación se ha utilizado:

- El navegador Google Chrome en su versión más actualizada, para poder acceder a las distintas fuentes de información necesarias durante el desarrollo del proyecto, así como para acceder a *Google Colab*, lugar donde ejecutaremos nuestros algoritmos de forma online con los recursos externos de Google. También necesario para acceder a páginas web como “draw.io” o “overleaf.com”.
- Para las pruebas de forma local (en mi ordenador) el entorno de programación elegido para la creación de nuestro programa ha sido “PyCharm”, en este se codifica y se prueba, por tanto se puede decir que es tanto el entorno de desarrollo como el entorno de pruebas, de forma local.
- Python en su versión más reciente a ser posible, puesto que es el lenguaje de programación elegido por el estudiante.

### 1.7.3. RECURSOS HUMANOS

Las personas implicadas en este proyecto son:

- **Director del proyecto:** Prof D. Juan Gómez Romero  
Profesor del departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada. Su labor en este proyecto consiste en guiar los pasos a seguir por parte del alumnado y proporcionarle la información y recursos necesarios para que el resultado del proyecto sea exitoso. También realiza la labor de revisar periódicamente el trabajo realizado por el autor del proyecto.
- **Autor del proyecto:** Ángel Murcia Díaz  
Estudiante del Máster de Informática de la Universidad de Granada. Su tarea es la de realizar el proyecto en su totalidad.

## 1.8. ESTRUCTURA DEL DOCUMENTO

En esta sección se expone una guía aproximada del contenido de cada uno de los capítulos de esta memoria:

- **Capítulo 1: Introducción**

Este es el capítulo actual, una introducción para el lector del proyecto, de tal forma que este después de leer el capítulo sepa de forma global el tema del proyecto así como el trabajo que se va a realizar durante el desarrollo del mismo.

- **Capítulo 2: Marco teórico.**

Este capítulo es fundamental en cualquier proyecto (académico o científico) puesto que en él se explican las bases donde se sustenta cualquier análisis, experimento o propuesta de desarrollo. En concreto se van a explicar conceptos teóricos referentes a la *Inteligencia Artificial*, al *Aprendizaje Automático* y de forma más extensa al *Aprendizaje por Refuerzo* y al *Aprendizaje por Refuerzo Profundo*, así como diferentes algoritmos de estos tipos de aprendizaje.

- **Capítulo 3: Herramientas.**

Este capítulo explica en profundidad las herramientas utilizadas para el proyecto: el entorno con el que se trabaja durante todo el proyecto: *Google Research Football Environment*, las librerías a utilizar en el mismo, así como el uso de *Google Colab* y el sistema de multi-entorno.

- **Capítulo 4: Metodología del proceso experimental.**

Este capítulo explica el proceso experimental que se propone utilizando algoritmos de *Aprendizaje por Refuerzo* sobre el entorno *Google Research Football Environment*, así como los pasos que se van dando durante el desarrollo, hasta llegar a lanzar las pruebas que componen la experimentación.

- **Capítulo 5: Experimentación y discusión de los resultados.**

Este capítulo se centra en el análisis de los resultados extraídos durante el proceso experimental realizado. Para ayudar a este análisis se utilizan algunas gráficas de diferentes tipos, puesto que siempre el apartado visual es más fácil de comprender para el lector.

- **Capítulo 6: Conclusiones y futuras mejoras.**

Este capítulo expone las conclusiones extraídas del desarrollo del proyecto y propone futuras mejoras a realizar en este proyecto partiendo desde él mismo, está pensando para el futuro lector o estudiante que esté interesado en continuar con este proyecto.

# Capítulo 2

# MARCO TEÓRICO

En este capítulo se describe el marco teórico en el cual se encuadra este proyecto. Como se comentó durante la sección de introducción este capítulo es fundamental, puesto que en él se explican las bases donde se sustenta cualquier análisis o propuesta de desarrollo.

En primer lugar, se hace una breve introducción de la *Inteligencia Artificial*, que es el gran campo donde se encuadra este proyecto. Posteriormente, se describe el *Aprendizaje Automático*. Más adelante, se describe en profundidad el *Aprendizaje por Refuerzo*. Después, se describe las *Redes Neuronales Artificiales*. Por último, se detalla en qué consiste el *Aprendizaje por Refuerzo Profundo*, haciendo hincapié en un conjunto de algoritmos de este tipo que se utilizan durante el proceso experimental del proyecto.

En definitiva, en este capítulo se parte de una visión global del marco teórico en el que se encuentran el conjunto de técnicas que se utilizan, hasta llegar a la descripción detallada de las técnicas en particular.

## 2.1. INTELIGENCIA ARTIFICIAL

Según su definición: La **Inteligencia Artificial (IA)** o *Artificial Intelligence (AI)* en inglés, «es el arte de desarrollar máquinas con la capacidad de realizar funciones que cuando son realizadas por humanos requieren inteligencia» (Rich y Night 1991) [33].

En primer lugar, hablaremos del contexto histórico. El término *Inteligencia Artificial* empieza a ser utilizado en 1956, fue acuñado por *John McCarthy* en la célebre reunión de Dartmouth (reunión sobre tecnología y ciencias de la información). En esta se lanzaron predicciones demasiado optimistas que elevaban a la inteligencia artificial como el futuro a corto plazo [17] aunque es actualmente cuando se ha hecho más popular este campo

seguramente por el interés en manejar grandes cantidades de datos, en tener algoritmos muy complejos o en aumentar el poder de cómputo o almacenaje. Según *John McCarthy* la *Inteligencia Artificial* es: “la ciencia y la ingeniería de crear máquinas inteligentes, especialmente programadas por computación inteligente” [31].

La investigación en este campo empieza en la década de los años 40 pero no es hasta la época de los 50 cuando el concepto experimenta un estallido de popularidad, en esa época se estudiaba la resolución de problemas y los métodos simbólicos, el científico y matemático Alan Turing es considerado el padre de la *Inteligencia Artificial* puesto que él escribe en esta década un trabajo que se considera la base de esta disciplina: *Computing Machinery and Intelligence*. En la década de los años 60, una gran potencia como EEUU, concretamente su departamento de defensa mostró interés por este campo y comenzó a invertir dinero en entrenar ordenadores para imitar el comportamiento de los seres humanos. En la década de los años 70 *DARPA (Defense Advanced Research Projects Agency)* realiza proyectos de planimetría y en el año 2003 crea asistentes personales inteligentes, que fueron en los que se inspiraron para realizar *Siri* o *Cortana* [17].

Este trabajo del primer asistente personal inteligente abrió el camino para la automatización y el razonamiento formal que vemos hoy en los ordenadores, incluyendo sistemas de soporte a decisiones y sistemas de búsqueda inteligentes.

Aunque el cine Americano y las novelas de ciencia ficción representan la *Inteligencia Artificial* como robots malignos que acaban eliminando a los humanos y apoderándose del mundo (a esto se le conoce como singularidad tecnológica), la realidad es que la *Inteligencia Artificial* ha evolucionado y sigue evolucionando para beneficiar y ayudar a todos los humanos [35].

Las aplicaciones reales en las que hoy día se utiliza la *Inteligencia Artificial* son prácticamente ilimitadas, por ejemplo: en la salud, en las finanzas, en el comercio, en el transporte, en la música, en los juegos, en los juguetes, en la industria pesada, etc.

Por lo que se puede decir realmente que la *Inteligencia Artificial* ha cambiado nuestra forma de vivir y de trabajar en todos los sentidos, siendo más revolucionaria que cualquier tecnología desde el nacimiento de internet.

Para adaptarse a un rango tan amplio de tareas, la *Inteligencia Artificial* ha tenido que separarse para evolucionar más rápidamente, generando diferentes tipos de *Inteligencia Artificial*, estos son [17]:

- **Máquinas reactivas:** este tipo de *Inteligencia Artificial* se basa en decisiones sobre el momento presente, pero no puede analizar datos pasados ni aprender en base a esos datos, es decir, no evoluciona. Un

ejemplo sería la computadora *Deep Blue*, esta era capaz de reconocer las figuras en el tablero y procesar 200 millones de movimientos en un segundo. Pero su único objetivo era procesar cuál era el mejor movimiento en tiempo real en base a la acción que realizaba su oponente, la parte mala es que ignora cualquier dato pasado o futuro externo a la partida o al procesamiento del siguiente movimiento.

- **Máquinas con memoria limitada:** este tipo de *Inteligencia Artificial* hace que las máquinas sean capaces de mirar al pasado aunque de forma limitada. De forma que pueden almacenar la información durante cierto tiempo, y añadirla a su programación para crear nuevos patrones de comportamiento. Dentro de este tipo, las máquinas utilizan el *Aprendizaje Automático* (se profundizará en él en la próxima subsección) para, en base a unos conocimientos base, añadir nueva información a su base de datos para mejorar su funcionamiento.
- **Máquinas con teoría de la mente:** este tipo de *Inteligencia Artificial* se basa los sistemas o máquinas capaces de entender cómo funcionan las personas, objetos o sistemas que los rodean. Estas inteligencias artificiales aprenden en base a nuestros comportamientos y, en base a ellos, saben cuáles son nuestros gustos, necesidades, deseos o cómo esperamos ser tratados. Este tipo de *Inteligencia Artificial* es capaz de comprender el entorno que le rodea, y supone un paso importante para poder realizar una interacción social más cercana a lo que se esperaría de un humano.
- **Máquinas con autoconciencia:** este tipo de *Inteligencia Artificial* se basa en que la máquina tenga conciencia de sí misma, se reconozca como un ente independiente y sea capaz de tomar decisiones, diferenciando entre sí mismo y los objetos, personas o sistemas que le rodean. Todavía no existen máquinas o sistemas que tengan conciencia de sí mismos, más allá de las inteligencias malvadas que conocemos de películas.

Entre las ventajas que tiene la *Inteligencia Artificial* se destacan las siguientes [17]:

- Aumenta la eficacia y rapidez de los procesos que realiza.
- Permite que las personas no tengan que realizar tareas repetitivas puesto que se pueden hacer de forma totalmente automática.
- Puede desarrollar tareas que son peligrosas para los seres humanos, como las tareas realizadas en el espacio.

- Evita errores que pueden tener los seres humanos, puesto que sus tareas se basan en procesos computacionales, además no sufren algunos problemas que si sufren los humanos, como: cansancio tanto físico como emocional.
- Facilita la vida de las personas, por ejemplo siendo utilizada en asistentes personales virtuales.
- Realizan tareas “mecánicas” con alta precisión, gracias al análisis de gran cantidad de datos y variables.

## 2.2. APRENDIZAJE AUTOMÁTICO

Como se ha comentado en la sección de introducción y en la subsección anterior de este capítulo, el *Aprendizaje Automático* es un campo de la *Inteligencia Artificial*.

Según su definición científica: El **Aprendizaje Automático (AA)** o *Machine Learning (ML)* en inglés, es una disciplina del campo de la Inteligencia Artificial que, a través de algoritmos, dota a los ordenadores de la capacidad de identificar patrones en datos masivos para hacer predicciones. Este aprendizaje permite a los computadores realizar tareas específicas de forma autónoma, es decir, sin necesidad de ser programados. Los algoritmos de *Aprendizaje Automático* permiten a los ordenadores “entrenarse” con datos de entrada, además de utilizar modelos estadísticos, con el fin de llegar a una solución dentro de un rango específico.

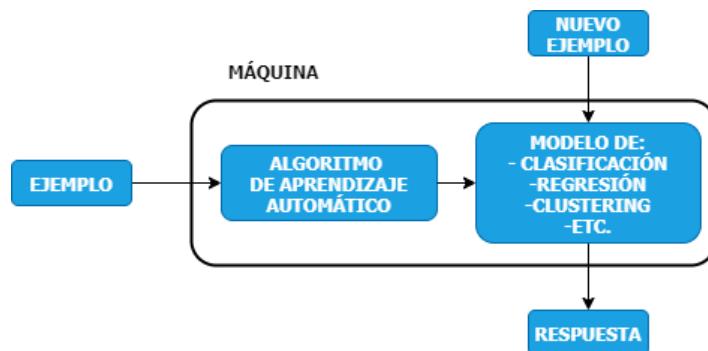


Figura 2.1: Proceso de *Aprendizaje Autómatico*

El flujo de trabajo en el *Aprendizaje Automático* es el que se puede observar en la figura 2.1. A partir de un conjunto de datos de ejemplo, se genera un modelo (que puede ser de clasificación, predicción, etc.). Una vez entrenado este modelo, dado un nuevo ejemplo, se puede utilizar el modelo para generar una respuesta.

Por ejemplo, primero se entrena un modelo de clasificación que es capaz de distinguir si una imagen corresponde a un perro o si corresponde a un gato utilizando para el entrenamiento de este modelo una serie de ejemplos con imágenes de perros y gatos previamente etiquetados por el experto. Después dado un nuevo ejemplo que no ha sido utilizado previamente durante el entrenamiento, el modelo será capaz de predecir si ese ejemplo corresponde con un perro o con un gato. Los algoritmos de *Aprendizaje Automático* son capaces de identificar patrones de comportamiento en los datos [21].

El *Aprendizaje Automático* está en continuo desarrollo, lo que produce una gran satisfacción para la tecnología, optimizándola, y creciendo exponencialmente a lo largo de los años. El interés en este campo es debido a que es aplicado en infinidad de tecnologías, incluso en acciones cotidianas de la vida, como en el desbloqueo facial del móvil.

El *Aprendizaje Automático* se divide en varios subcampos que fueron explicados con detalle durante la sección de introducción, los cuales se podían ver en la figura 1.1.

### 2.2.1. APLICACIONES REALES

Para un mayor acercamiento por parte del lector a este tipo de ciencia de la computación, durante esta subsección se explican aplicaciones reales del *Aprendizaje Automático*. Destacar que existe gran variedad entre ellas, y la gran mayoría serán cotidianas en nuestras vidas en un futuro muy cercano, puesto que la tecnología esta avanzando muy rápido durante los últimos años.

#### ASISTENTES PERSONALES VIRTUALES

Como se comentó anteriormente los asistentes virtuales abrió el camino para la automatización y diseño formal que vemos hoy día en los ordenadores, por lo que su importancia es notable.

*Siri, Alexa o Google Now* son algunos de los ejemplos más populares de asistentes personales virtuales. Estos ayudan a encontrar información cuando se les pregunta por voz. Para responder estos agentes: buscan la información, recuerdan sus consultas relacionadas o envían un comando a otros recursos (como al teléfono) para recopilar información. Incluso pueden realizar ciertas tareas como establecer alarmas o poner recordatorios [2].

El aprendizaje automático es una parte importante de estos asistentes personales, puesto que recopilan y refinan la información sobre la base de las anteriores interacciones con el agente. Posteriormente, este conjunto de datos se utiliza para generar resultados que se adaptan a sus preferencias,

por lo que se puede decir que cada agente personal es “personal” valga la redundancia.

Los asistentes en realidad están integrados en muchas plataformas. Por ejemplo:

- *Altavoces inteligentes:* Amazon Echo y Google Home
- *Teléfonos inteligentes:* Samsung Bixby en Samsung S8
- *Aplicaciones móviles:* Google Allo



Figura 2.2: Imagen de Alexa, extraída de powerplanetaonline

## PREDICCIONES DE TRÁFICO

¿Quién no ha utilizado en alguna ocasión los servicios de navegación GPS, para calcular la ruta más rápida para ir a algún lugar en coche? La respuesta es: todos hemos estado utilizando en alguna ocasión de nuestra vida los servicios de navegación GPS. Durante este proceso, nuestra ubicación actual y velocidad actual se guardan en un servidor central para administrar el tráfico. Después, estos datos se utilizan para construir un mapa del tráfico actual.

Este sistema también lo utilizan otros servicios como Uber, de forma que puede anticipar la demanda de los pasajeros.

El *Aprendizaje Automático* es la solución en el sentido de que ayuda a estimar las regiones donde se puede encontrar la congestión sobre la base de las experiencias diarias.



Figura 2.3: Icono semáforo, extraído de freepik

## RECONOCIMIENTO DE IMAGEN

Para el ojo humano resulta una tarea muy sencilla el reconocer cualquier imagen o los diferentes objetos que existen en una imagen. En contraposición, para una computadora, esta es una tarea muy complicada porque para la computadora una imagen no es más que una serie de valores numéricos, por esto se utilizan algoritmos de procesamiento de imágenes para buscar patrones en imágenes digitales. Mediante el uso de estos algoritmos, las computadoras trabajan en el reconocimiento de patrones y los algoritmos de *Aprendizaje Automático* pueden reconocer cualquier forma de elementos visuales.

Por ejemplo, utilizando *Aprendizaje Automático* se puede desbloquear un teléfono móvil a través del reconocimiento facial, la cámara de alta gama del teléfono reconoce 80 puntos nodales en un rostro humano y usa tecnologías de *Aprendizaje Automático* para medir la variable del rostro de una persona y desbloquear el teléfono. Esto se encuentra ahora entre las aplicaciones comunes de aprendizaje automático [26]. En un futuro no muy lejano se acabará utilizando el reconocimiento facial para realizar pagos bancarios.

Otros usos del reconocimiento de imágenes son:

- Drones
- Fabricación
- Coches autónomos
- Actividades forestales

## VÍDEOVIGILANCIA

Los sistemas de videovigilancia actuales de cualquier sitio funciona con *Inteligencia Artificial*, que permite detectar delitos antes de que ocurran. Imaginas una sola persona monitoreando múltiples cámaras de video, esto es un trabajo difícil a la vez que aburrido, por lo que tiene sentido que los sistemas de videovigilancia actuales funcionen de forma automática.

La forma en que funcionan estos sistemas es realizando un seguimiento del comportamiento inusual de las personas, como permanecer inmóviles durante mucho tiempo, tropezar o tomar una siesta en los bancos, etc. El sistema una vez que detecta alguno de estos comportamientos inusuales puede dar una alerta a los asistentes humanos, lo que en última instancia puede ayudar a evitar contratiempos [26].

La videovigilancia se utiliza para diferentes propósitos como:

- Prevención del robo de cobre
- Detección de eventos anormales
- Protecciones de instalaciones
- Monitoreo de operación
- Estacionamientos
- Monitoreo de tráfico
- Patrones de compra

## ANÁLISIS DE SENTIMIENTOS

El análisis de sentimientos es una aplicación de *Aprendizaje Automático* de primer nivel que se refiere a la clasificación de sentimientos, la extracción de opiniones y el análisis de emociones. Normalmente se busca analizar los 7 sentimientos básicos: ira, desprecio, asco, miedo, felicidad, neutral y tristeza.

Para este cometido las máquinas se preparan para analizar sentimientos basados en las palabras, de forma que pueden identificar si las palabras se dicen en una noción positiva, negativa o neutral. Además, pueden definir la magnitud de estas palabras.

Con la ayuda del proceso llamado Procesamiento del lenguaje natural (NLP), los mineros de datos extraen y concluyen automáticamente la opinión analizando ambos tipos de algoritmos de aprendizaje automático: datos supervisados y no supervisados. Las empresas que tratan con clientes utilizan este modelo para mejorar la experiencia del cliente en función de los comentarios [26].

Un ejemplo es el de aplicaciones como *Jiosaavn*, esta aplicación sugiere música basada en los sentimientos de los usuarios al analizar el historial de las canciones reproducidas, las listas de reproducción favoritas e incluso el tiempo de las listas de música.

Los beneficios del análisis de sentimientos son:

- Monitoreo efectivo de marcas y redes sociales
- Soporte al cliente mejorado
- Análisis competitivo
- Mejor seguimiento de los comentarios de los empleados y UGC (contenido generado por el usuario como reseñas)

## SERVICIOS EN REDES SOCIALES

Los servicios en los que está presente la *Inteligencia Artificial* en las redes sociales abarca desde la personalización de sus noticias hasta una mejor orientación de los anuncios. Las plataformas de redes sociales utilizan el *Aprendizaje Automático* para sus propios beneficios y los del usuario.

Esto afecta a la gran mayoría de las personas puesto en la actualidad casi todo el mundo tiene redes sociales. En estas los ejemplos de características que utilizan *Aprendizaje Automático* son fáciles de notar e incluso es fácil estar utilizando estas maravillosas características sin saber que no son más que aplicaciones de *Aprendizaje Automático*. Algunas de estas características son [26]:

- *Personas que quizás conozcas*: esta característica se basa en que el *Aprendizaje Automático* funciona con un concepto simple: comprensión con experiencias. Las redes sociales comprueba continuamente a los amigos con los que te conectas, los perfiles que visitas con mucha frecuencia, tus intereses, lugar de trabajo o un grupo que compartes con alguien, etc. Sobre la base de la información que ha ido almacenando con el paso del tiempo propone nuevas amistades que seguramente le gusten al usuario.
- *Reconocimiento facial*: esta característica se basa en que una persona sube una foto tuya con un amigo a la red social y esta reconoce instantáneamente a ese amigo. Puesto que previamente se verifican las poses y proyecciones en la imagen, observa las características únicas y luego las empareja con las personas en tu lista de amigos.

- *Pines similares:* esta característica es una técnica para extraer información útil de imágenes y videos que está basada en que el *Aprendizaje Automático* es el elemento central de la visión por computador. Por ejemplo, Pinterest utiliza la visión por computadora para identificar los objetos (o pines) en las imágenes y recomienda pines similares en consecuencia.



Figura 2.4: Icono de Redes sociales, extraído de concepto.de

## RECOMENDACIÓN DE PRODUCTOS

Un sistema de recomendación de productos es algo esencial hoy día, puesto que a las personas con los mismos gustos les suelen gustar las mismas cosas o productos, en esta premisa se basan este tipo de sistemas. Uno de los ejemplos más conocidos es Netflix.

Si has comprado recientemente en alguna tienda online, lo más seguro es que te llegue al correo una sugerencia de compra de otro producto, si no es así, es posible que haya notado que el sitio web de compras o la aplicación le recomiende algunos artículos que de alguna manera coinciden con su gusto [26] [1].

Es el *Aprendizaje Automático* quién se encarga de la selección de estos productos sugeridos, en función de su comportamiento con el sitio web, compras anteriores, artículos que me gustaron o que se agregaron al carrito, preferencias de marca, etc.

## FILTRADO DE CORREO Y MALWARE

Hay una serie de filtros de correo no deseado que son utilizados por los clientes de correo electrónico. Estos filtros se actualizan constantemente utilizando *Aprendizaje Automático*. Funciona mejor que el filtrado de spam basado en reglas puesto que este no rastrea los últimos trucos que adoptan por los spammers. Perceptrón multicapa, o la inducción del árbol de decisiones son algunas de las técnicas de filtrado de spam que funcionan con *Aprendizaje Automático*.

Cada día se detectan aproximadamente 325.000 malwares que son similares en un 90–98 % a sus versiones anteriores. Los programas de seguridad del sistema, es decir, los antivirus o antimalware entre otros, funcionan con *Aprendizaje Automático* para comprender el patrón de codificación de los malwares. Por lo tanto, detectan fácilmente un malware nuevo con una variación del 2 al 10 % y ofrece protección contra este [2].

### **ATENCIÓN AL CLIENTE EN LINEA**

Actualmente, la mayoría de comercios online ofrecen la opción de chatear con el representante de atención al cliente mientras el cliente navega por el sitio web. Pero no todos los sitios web tienen una persona contratada para el cometido, es decir, responder en vivo a las consultas realizadas por el cliente, en la mayoría de los casos, hablas con un “chatbot”.

Estos bots tienden a extraer información del sitio web y presentarla a los clientes. Con el paso del tiempo, los “chatbots” tienden a comprender mejor las consultas de los usuarios y brindarles mejores respuestas, lo cual es posible gracias a sus algoritmos de *Aprendizaje Automático* [26] [1].

### **REFINAMIENTO DE RESULTADOS DE MOTORES DE BÚSQUEDA**

Google y otros motores de búsqueda utilizan el *Aprendizaje Automático* para mejorar los resultados de la búsqueda. De forma que cada vez que el usuario ejecuta una búsqueda, los algoritmos vigilan cómo el usuario responde a los resultados, de tal forma que si este abre los resultados principales y permanece en la página web por mucho tiempo, el motor de búsqueda asume que los resultados que mostró coincidieron con la consulta, pero si este llega a la segunda o tercera página de los resultados de la búsqueda pero no abre ninguno de los resultados, el motor de búsqueda estima que los resultados mostrados no cumplieron con los requisitos. De esta manera, estos algoritmos de *Aprendizaje Automático* mejora ostensiblemente los resultados de búsqueda [2].

### **DETECCIÓN DE FRAUDE**

El *Aprendizaje Automático* está demostrando su potencial para hacer del ciberespacio un lugar seguro y el seguimiento de los fraudes monetarios en línea es uno de sus ejemplos [2].

Por ejemplo: Paypal usa *Aprendizaje Automático* para protegerse contra el lavado de dinero. La empresa utiliza un conjunto de herramientas que les ayuda a comparar millones de transacciones que tienen lugar y a distinguir

entre transacciones legítimas o ilegítimas que tienen lugar entre compradores y vendedores.



Figura 2.5: Icono de Detección de Fraude, extraído de ittravelservices

## SEGURIDAD FINANCIERA

Una de las prioridades de cualquier entidad financiera es la ciberseguridad, por lo que es uno de los fines más importantes del *Aprendizaje Automático*. Los bancos utilizan métodos de análisis discriminante lineal para detectar transacciones fraudulentas y alertar a sus clientes sobre la posibilidad de tales actividades. Esto se hace analizando las actividades financieras y asignando puntuaciones de riesgo para hacer predicciones sobre acciones fraudulentas [1].

También los minoristas en línea y los servicios de pago deben implementar una detección de fraude porque recuperarse de los ciberataques puede ser mucho más difícil que prevenirlos.

## RECONOCIMIENTO DE VOZ

El reconocimiento automático de voz consiste en la transformación de la palabra hablada en texto y hace que la entrada de datos sea mucho más sencilla, ya que la palabra hablada es significativamente más rápida que la velocidad de escritura. Esta tarea la pueden realizar ordenadores, teléfonos móviles o asistentes personales entre otros dispositivos.

Las máquinas ahora pueden comprender fácilmente una señal de voz, un hecho que los consumidores aprovechan todos los días cuando usan la búsqueda por voz en un motor de búsqueda o le indican a Alexa que elija su próxima película. El reconocimiento de voz puede incluso permitir que los usuarios hablen directamente con los “chatbots”, ya que las técnicas de *Aprendizaje Automático* pueden “traducir” las ondas sonoras en bits, aunque

todavía existen algunos desafíos, como la velocidad del habla, que pueden dificultar la traducción. Sin embargo, esta tecnología ya se ha introducido en la vida diaria de muchas personas [1].



Figura 2.6: Reconocimiento de voz, extraído de linuxadictos

## SERVICIOS MÉDICOS

El cuidado de la salud es uno de los usos más nobles del aprendizaje automático, y las aplicaciones pueden variar desde escanear rápidamente los resultados de las pruebas para detectar el cáncer de mama hasta usar redes neuronales artificiales para imitar los procesos de pensamiento del cerebro humano y obtener información más profunda sobre nuestras propias mentes [1].

Las máquinas incluso se pueden utilizar para hacer un diagnóstico médico en determinadas circunstancias en las que los pacientes superan en número a los médicos o cuando la fatiga podría provocar errores dañinos. Por ejemplo, la IA de Infervison se está utilizando en China para detectar casos de cáncer de pulmón temprano. Sin esta máquina, la falta de radiólogos calificados haría casi imposible salvar muchas vidas.

Por supuesto, no todas las aplicaciones son de vida o muerte. El aprendizaje automático se puede utilizar para facilitar aspectos simples de las visitas al médico, como la planificación de la terapia para su próxima sesión.

## COCHES AUTÓNOMOS

La tarea de conducir un automóvil de forma autónoma [40] por una pista de carreras fue abordada previamente desde la perspectiva de la neuroevolución por Koutnik para controlar un automóvil en el entorno de simulación de carreras TORCS utilizando la visión desde la perspectiva del conductor. Actualmente en lugar de usar la neuroevolución, se usa una red neuronal para la estimación del estado, junto con un algoritmo de *Aprendizaje por*

Refuerzo, en este tema se profundiza durante las próximas subsecciones.

Esta tarea es de los inventos más impactantes puesto que el transporte es algo necesario para la humanidad. La conducción autónoma es una de las aplicaciones por las que se investiga y se desarrolla el *Aprendizaje por Refuerzo*, ya que las pruebas corroboran el éxito de estas técnicas en los automóviles.

Se prevé que en un futuro esto sea algo habitual y la gran mayoría de los coches conduzcan autonómicamente, aunque para esto aún queda recorrido tanto por el lado tecnológico como por el lado de concienciar a las personas de que estos coches son seguros y se pueden utilizar, eso último quizá sea lo más complicado.



Figura 2.7: Coche autónomo, extraído de motorpasión

### 2.3. APRENDIZAJE POR REFUERZO

Después de ver la utilidad del *Aprendizaje Automático* desde la perspectiva de aplicaciones reales, durante esta subsección se explica con máximo detalle, tanto de forma más científica como de forma más cercana, en qué consiste la rama del *Aprendizaje Automático* que se utiliza de forma práctica en este proyecto: el *Aprendizaje por Refuerzo* [24] [37] [41].

Según su definición científica, el **Aprendizaje por Refuerzo** es un área del *Aprendizaje Automático* inspirado en la psicología conductista, cuya ocupación es determinar qué acciones debe escoger un agente software en un entorno con el fin de maximizar alguna noción de “recompensa” o premio acumulado. Posteriormente se explican conceptos del *Aprendizaje*

por Refuerzo, los componentes que lo forman y finalmente algunas técnicas comúnmente utilizadas.

Cabe destacar las consecuencias de las decisiones tomadas se demoran en el tiempo y no se aprecian de forma inmediata, por lo que hace complicado medir como de buena o como de mala es una decisión que se toma. Esto es uno de los principales retos del *Aprendizaje por Refuerzo*.

También se ha de destacar la *falta de información* exhaustiva y la incapacidad de recolectarla en su totalidad, puesto que a veces solo se puede recolectar parte de la misma. Esta incertidumbre que se crea, provoca que se tengan que intercalar dos técnicas en busca de la solución óptima, estas son:

- **Exploración:** este método busca encontrar la mejor solución alejándose de las regiones prometedoras, normalmente de forma aleatoria.
- **Explotación:** este método persigue encontrar la mejor solución en regiones prometedoras.

Posteriormente se explican los conceptos y componentes que forman este tipo de problemas a través de un marco matemático de trabajo que se llama **Procesos de Decisión de Markov (MDPs)**, estos marcos permiten modelar un problema de *Aprendizaje por Refuerzo* [14].

### 2.3.1. COMPONENTES

El proceso ilustrado del *Aprendizaje por Refuerzo* con todos sus componentes señalados se puede ver en la figura 2.8.

Los componentes principales [30] [14] de un proceso de *Aprendizaje por Refuerzo* son el **agente** y el **entorno**. El agente es un tomador de decisiones en el entorno, por tanto el agente *interactúa* tratando de influir en el entorno realizando *acciones*, mientras el entorno *reacciona* ante ellas y genera nuevos *estados*.

- **El espacio de estados (S)** es el conjuntos de todos los posibles estados que existen. Puede ser finito o infinito, pero cada estado esta formado por un conjunto *finito* de variables.
- **El espacio de acciones (A)** es el conjunto de acciones que puede utilizar el agente, este puede variar dependiendo del estado en que se encuentre el entorno. Al subconjunto de acciones que puede realizar el agente condicionado por el estado en el que se encuentra el entorno se le llama  $A(s)$ . Esto ocurre en problemas en los cuales el agente en

ciertos momentos no puede realizar ciertas acciones, por ejemplo en el fútbol no puedes tirar a puerta en una falta indirecta, por lo que el supuesto agente no tendría la opción de “disparo” entre las acciones de su subconjunto de acciones disponibles.

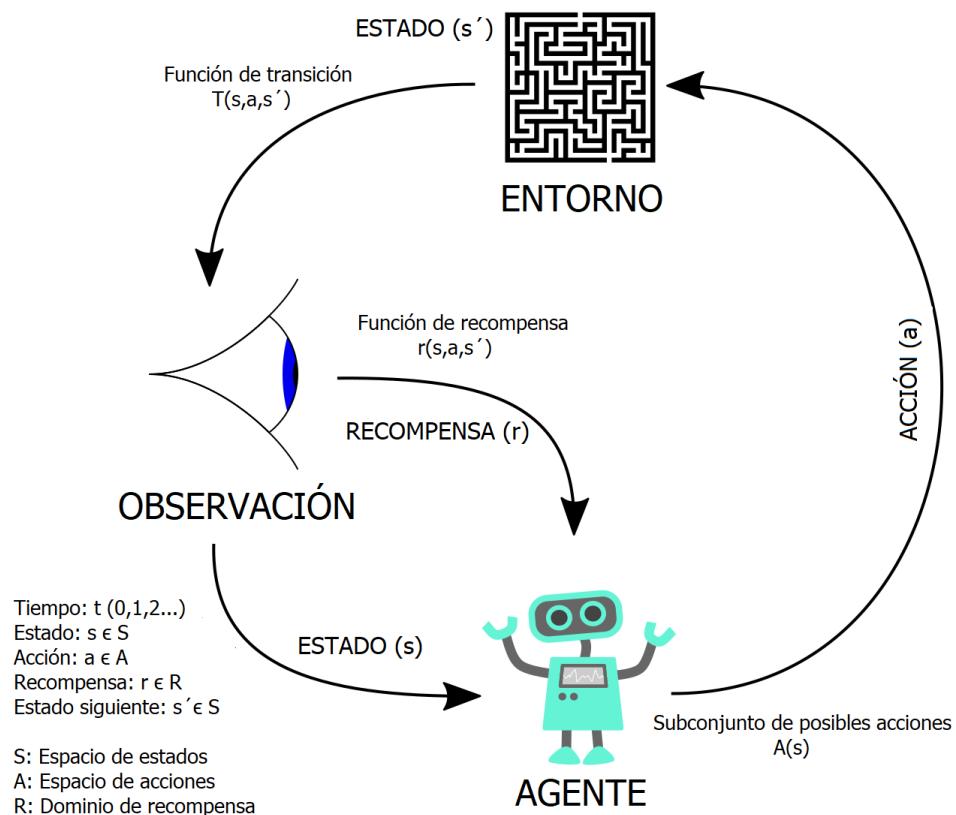


Figura 2.8: Esquema de componentes del proceso de *Aprendizaje por Refuerzo*, extraido de wikipedia

Cuando se produce un cambio de estado en el entorno quiere decir que el entorno está reaccionando, este cambio repercute en las observaciones del agente en el futuro. La **recompensa** es una parte fundamental puesto que de la misma depende el aprendizaje (cuanto mejor se aproxime la cuantificación de la recompensa a la solución del problema, mejor podrá realizarse el entrenamiento del agente).

La observación y el estado del entorno pueden coincidir o no. Cuando no coinciden, se dice que la **observación** es una interpretación del estado. En algunos problemas los agentes solo son capaces de observar parte del estado mientras que en otros son capaces de observar el estado al completo.

Después de que el agente realiza una acción de su subconjunto de acciones

a realizar, el entorno cambia su estado entre uno de los disponibles en el espacio de estados. La función que hace este cometido se llama **función de transición**.

Una vez ocurrida la transición, el entorno ofrece un nuevo estado, lo que implica una nueva observación por parte del agente. El entorno también otorga una **recompensa** (también llamada *penalización* por la comunidad) al agente. La función que hace este cometido se llama **función de recompensa**.

Las funciones previamente comentadas, es decir, la función de transacción y la función de recompensa, establecen el modelo del entorno.

La interacción entre el agente y el entorno se produce en una serie de ciclos secuenciales, como se comentó anteriormente y se puede ver en la figura 2.8. Cada ciclo se le conoce como **step o paso**, esto es, una unidad de tiempo en la que se completa un ciclo de interacción, dependiendo del problema.

En cada paso el agente observa el entorno, realiza una acción y recibe del entorno una recompensa y una nueva observación, la experiencia recolecta estos datos en forma de tupla (**tupla de experiencia**).

Un **episodio** lo forman un conjunto de pasos, el último paso puede significar alcanzar el objetivo del entorno. Por lo que de forma más formal un episodio es una secuencia de estados del entorno, acciones realizadas y recompensas obtenidas que finalizan en un **estado terminal**, este estado puede ser éxito o fracaso. Un agente puede necesitar varios pasos para alcanzar un estado terminal o incluso puede ser que en el problema no tenga este tipo de estados (estos problemas se llaman **problemas continuos**). Para este caso se debe de establecer un **horizonte** o número máximo de pasos para que cuando se llegue a ese número de pasos concluya el entrenamiento.

La suma de todas las recompensas que se obtienen desde el principio del episodio hasta el final del episodio se llama **recompensa acumulada**. Con esta de forma implícita se puede saber si se ha conseguido un estado terminal favorable o desfavorable para el agente.

### 2.3.2. PROCESOS DE DECISIÓN DE MARKOV (MDPs)

El **Proceso de Decisión de Markov (MDPs)** es un marco teórico probabilístico que se utiliza para describir un problema de *Aprendizaje por Refuerzo*.

La amplia mayoría de los problemas de *Aprendizaje por Refuerzo* pueden ser formalizados en este tipo de marco teórico o en una de sus extensiones de tal forma que en el se definen: el formato de cada uno de sus componentes y funciones que se han explicado en subsecciones anteriores.

Los MDPs parten de una suposición muy importante: las probabilidades de alcanzar el siguiente estado, dado el estado actual y la acción, es totalmente independiente de las interacciones previas. Esta es una propiedad no contextual de los efectos de una acción en el entorno y es conocida como la propiedad de Markov: La probabilidad de moverse de un estado ( $s$ ) a otro estado ( $s'$ ), dada la misma acción y estado, es la misma independientemente de todos los estados previos y desarrollo del episodio encontrados hasta ese momento del proceso [39] [11].

$$P(s'|s, a) = P(s'|s, a, S_{t-1}, A_{t-1}, S_{t-2}, A_{t-2}, \dots) \quad (2.1)$$

$S$  = Espacio de estados

$A$  = Espacio de acciones

$t$  = El paso actual

$$s' = S_{t+1}$$

$$s = S_t$$

$$a = A_{t+1}$$

## FUNCIÓN DE TRANSICIÓN

Esta función define las consecuencias de una acción tomada por el agente en el entorno. En este punto se aplica la mencionada anteriormente propiedad de Markov, para que devuelva un conjunto de probabilidades a estados de transición utilizando solamente el estado actual y la acción actual. Se define como se puede ver a continuación (2.2):

$$p(s'|s, a) = P(S_t = s' | S_{t-1} = s, A_{t-1} = a) \quad (2.2)$$

La suma de todas las probabilidades de los posibles estados siguientes del entorno debe de sumar 1, por lo tanto:

$$\sum p(s'|s, a) = 1, \forall s \in S, \forall a \in A(s), s' \in S \quad (2.3)$$

La complejidad de la función de transición cambia dependiendo del problema que se trata. Si este es totalmente **determinístico** solo existirá un estado con probabilidad 1 y el resto con probabilidad 0, mientras que si el problema es **estocástico** existe un factor de cierta aleatoriedad en las reacciones del entorno que deben calcularse para todos los estados que pueden sucederse a partir de una acción en un estado concreto.

## FUNCIÓN DE RECOMPENSA

Esta función devuelve un valor numérico que se corresponde con la magnitud de bondad de las acciones y consecuentes transiciones que se realizan. En otras áreas como la robótica es muy común agregarle a este valor un coste adicional de tiempo para reducir la bondad en función de este, de esta manera se penalizan los valores altos de tiempos de ejecución mientras que los tiempos bajos de ejecución premian. El objetivo es no ser solo eficaz, sino eficiente en las tareas que hay que realizar.

Esta función al igual que ocurría con la función de transacción explicada en la subsección anterior puede ser más o menos compleja, puede calcularse de diversas formas y al igual que sucedía antes depende del problema que se trata. Dependiendo de este se utilizan unos datos para calcular esta recompensa u otros, aunque normalmente se utiliza estado actual, acción actual y estado siguiente, por lo que la función se suele definir como se ve a continuación (2.4):

$$\begin{aligned} r(s, a) &= \mathbb{E}[R_t | S_{t-1} = s, A_{t-1} = a] \\ r(s, a, s') &= \mathbb{E}[R_t | S_{t-1} = s, A_{t-1} = a, S_t = s'] \\ R_t &\in R \end{aligned} \quad (2.4)$$

Con  $R$  nos referimos al dominio que tiene las señales de recompensa.  $\mathbb{E}$  es la esperanza matemática.

En algunos problemas es importante calcular la **recompensa acumulada** (*return*), esta es la suma de las recompensas obtenidas desde el principio hasta el final del episodio.

$$t = R_{t+1} + R_{t+2} + R_{t+3} + \dots + R_T \quad (2.5)$$

$T$  es el número total de pasos del episodio.

## DESCUENTO

Como ya es sabido la unidad de medida de tiempo es el paso, cada unidad está relacionada con un ciclo de interacción del agente con el entorno.

Un episodio son todos los pasos que suceden hasta llegar a un estado terminal (*done*) del problema, por lo que en principio los problemas son finitos, aunque como también se comentó anteriormente existen los problemas continuos que no tienen estado terminal por lo que en estos hay que definir un número de pasos límite o más conocido como horizonte para que se paren de ejecutar una vez se alcance ese número de pasos. También tenemos los

problemas híbridos que tienen tanto uno o varios estados terminales como un número predefinido de pasos límite para que se pare la ejecución en caso de que el nunca se llegue a uno de los estados terminales. En la comunidad esto es conocido como definir el tipo de horizonte a utilizar.

Al existir la opción de que se prolongue el entrenamiento del agente en el tiempo hasta encontrar un estado terminal, es necesario descontar de alguna manera el momento en que se encuentra, de forma que se premian las recompensas tempranas y se perjudican a las tardías. Este descuento se añade a la función de recompensa acumulada.

Normalmente se utiliza un número real positivo que este comprendido entre 0 y 1 para ir descontando de forma exponencial ese valor a las futuras recompensas. Este valor se le llama **factor de descuento** o gamma ( $\gamma$ ).

También puede ser considerado un reductor de incertidumbre en las estimaciones. Porque el futuro es incierto, y cuánto más pasos se mira hacia adelante, mayor es el factor estocástico y la variabilidad que tendrán las estimaciones. El factor de descuento ayuda a ir reduciendo el grado en el que las recompensas afectan y estabilizan el aprendizaje. En la figura 2.9, se puede ver como un valor de recompensa que fuera originalmente +1 se ve reducido en función a los pasos que se hayan ejecutado para conseguirlo.

**EFFECTO DEL FACTOR DE DESCUENTO Y TIEMPO SOBRE EL VALOR DE LAS RECOMPENSAS**

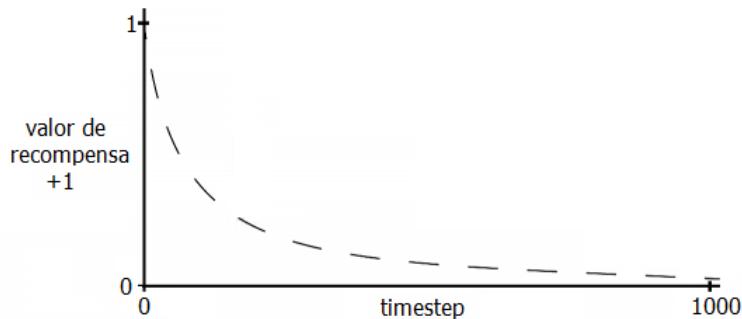


Figura 2.9: Gráfica de recompensa en relación con el tiempo influenciado por el factor de descuento, extraída de [24] y modificada

Como se puede ver conforme se aumenta el número de pasos se reduce la recompensa.

De forma matemática, la recompensa acumulada quedaría de la siguiente forma incluyendo el descuento (2.6):

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots + \gamma^{T-1} R_T \quad (2.6)$$

Si esta ecuación se simplifica para conseguir una más general (2.7):

$$G_t = \sum_{k=0}^T \gamma^k R_{t+k+1} \quad (2.7)$$

Hasta aquí, los componentes y conceptos del *Aprendizaje por refuerzo*. Destacar que hay muchas variables que se desarrollan para adaptarlo a conjuntos de problemas que tienen características concretas. Por ejemplo para problemas con observaciones parciales del entorno (POMDP) o para problemas con acciones, estados y recompensas continuos. No se van detallar puesto que no es objetivo de este *Trabajo Fin De Máster*.

### 2.3.3. POLÍTICA

En esta subsección, se habla de cómo solventar un problema en un MDPs. De forma general, hay que conseguir un sistema capaz de realizar una secuencia de acciones que consiga la mejor recompensa acumulada posible. Esto es lo que se conoce como política y se denota como  $\pi$ .

La política realmente es una función que devuelve la acción que utilizará el agente dado un estado del entorno, la intención de esta función es darle al agente la mejor acción posible en cada momento para maximizar la recompensa. Una política puede ser cualquier función, no tiene por qué ser necesariamente “inteligente”; de hecho, puede usarse una función aleatoria como política —si bien es cierto que es una mala política, es una política al fin y al cabo. En contraposición se pueden usar políticas muy complejas como es el caso una *Red Neuronal Artificial*.

Las políticas pueden ser tanto estocásticas como deterministas. En las políticas estocásticas dada una observación  $s$  tendremos como salida con distribución de probabilidades de todas las salidas posibles que corresponderá con las diferentes acciones que puede realizar el agente, mientras que en las políticas deterministas tendremos una única salida que será la acción a realizar por el agente.

Las siguientes funciones junto con la política, permiten describir, evaluar y mejorar el comportamiento de la misma política:

- *Función de acción-valor*
- *Función de estado-valor*
- *Función de acción-ventaja*

Se describirán posteriormente cada una de estas funciones con detalle.

## FUNCIÓN ACCIÓN-VALOR

La función **acción-valor** trata de valorar la realización por parte del agente de una acción  $a$  en un estado  $s$  (y continuar después aplicando la política  $\pi$ ). Si esta función es buena es una herramienta de gran utilidad a la hora de decidir entre que acciones elegir y así poder mejorar las políticas.

Por lo que dicho de otra forma esta función trata de estimar la recompensa acumulada de una política  $\pi$  después de haber realizado una acción  $a$  en un estado  $s$  y continuar hasta el estado terminal según la política  $\pi$  utilizada (2.8):

$$\begin{aligned} q_\pi(s, a) &= \mathbb{E}_\pi[G_t | S_t = s, A_t = a] \\ q_\pi(s, a) &= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a\right] \end{aligned} \quad (2.8)$$

Destacar que la acción  $a$  no tiene que ser obligatoriamente elegida por la política  $\pi$  sino que puede ser seleccionada de forma manual.

## FUNCIÓN ESTADO-VALOR

La función **estado-valor** sirve para comparar diferentes políticas, siendo de mucha utilidad puesto que una de las metas es poder comparar políticas para seleccionar la que mejor funcione en un problema tratado.

$$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s] \quad (2.9)$$

El funcionamiento de esta función es calcular la recompensa acumulada a partir de un estado o estados bajo una política concreta ( $\pi$ ). Dicho de otra forma se calcula como se va a comportar una política a partir de un estado del entorno. Si se reformula la función anterior (2.9) se puede llegar a la siguiente función (2.10).

$$v_\pi(s) = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s\right] \quad (2.10)$$

Con esta función es posible comparar políticas diferentes en un mismo problema, partiendo desde el mismo estado en ambos casos:

$$\pi > \pi' \Leftrightarrow v_\pi(s) \geq v_{\pi'}(s) \quad (2.11)$$

Por definición, se supone que existe una política óptima ( $\pi^*$ ), que es la que mejor funciona en el problema. Resolver el problema consistirá en encontrar una política que obtenga un valor de función estado-valor para cada estado lo más cercano posible a la política óptima.

$$\pi_* \geq \pi' \forall \pi' \quad (2.12)$$

La función estado-valor se puede expresar recursivamente, lo que se conoce como Ecuación de Bellman:

$$\begin{aligned} v_s &= \mathbb{E}_\pi[G_t | S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s] \end{aligned} \quad (2.13)$$

La recompensa acumulada es la suma de la recompensas hasta llegar al final del episodio, teniendo en cuenta la ponderación del valor de descuento exponencial ( $\gamma$ ). Si expresamos la acumulación de recompensa como la primera recompensa obtenida más la acumulación de recompensas siguientes multiplicadas por  $\gamma$  (elevado a 1 puesto que calcularse a partir del estado siguiente), podemos volver a obtener una función estado-valor y, por tanto, una función recursiva.

Se puede extraer una conclusión a partir de este concepto. En una política óptima  $\pi$ , la función estado-valor siempre será igual a la función de acción-valor para la acción a que estime el mejor valor en  $s$  (la mejor acción posible), ya que partimos del supuesto de que siempre elige la mejor acción al ser la mejor política (2.14):

$$\begin{aligned} v_{\pi_*} &= \max(q_{\pi_*}(s, a)) \\ a &\in A(s) \end{aligned} \quad (2.14)$$

## FUNCIÓN ACCIÓN-VENTAJA

La función **acción-ventaja** deriva de las anteriores, es decir, la función acción-valor y estado-valor. A esta también se le llama función ventaja puesto es la diferencia en términos de recompensas acumuladas estimadas entre la función acción-valor de una acción  $a$  en un estado  $s$  y la función estado-valor del estado  $s$  bajo la política  $\pi$ .

$$a_\pi(s, a) = q_\pi(s, a) - v_\pi(s) \quad (2.15)$$

Dicho de otra forma con esta función se determina cuánto de bueno es coger la acción  $a$  en el lugar de seguir la política  $\pi$ .

### EVALUACIÓN ITERATIVA DE UNA POLÍTICA

Con las 3 funciones anteriormente explicadas se puede evaluar el nivel de bondad de una política definida en un MDPs.

Con la función estado-valor es posible decidir cuando una política es mejor que otra. En esta subsección se va a explicar cómo aproximar la función estado-valor recorriendo el espacio de acciones ( $a(s)$ ) y mejorando dichas estimaciones de forma iterativa (2.16), este algoritmo es conocido como evaluación iterativa de la política que se adapta a cualquier problema definido en MDPs.

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) [r + \gamma v_\pi(s')] \quad (2.16)$$

A continuación se define a qué se refiere algunas de las variables de la función anterior:

- $k$  se refiere a la iteración que está ejecutando.
- $v_0(s)$  se refiere a la iteración inicial de estimaciones por parte de la función estado-valor para cada estado  $s \in S$  con la política que estamos evaluando, serán valores arbitrarios y 0 en caso de que el estado sea terminal para cada estado  $s \in S$ .
- $\pi(a|s)$  se refiere a la probabilidad de ejecutar una acción  $a$  en el estado  $s$  por parte de la política. Para simplificar, se asume que la política es determinística, por lo que el valor será directamente 1 para la acción seleccionada y 0 para el resto.
- $s'$  y  $r$  se refiere al siguiente estado y recompensa obtenida tras realizar dicha acción por parte de la política. Como ya se ha mencionado anteriormente, una acción  $a$  puede desencadenar más de un  $s'$  diferente, dependiendo de si el problema es estocástico o no, por eso se habla de probabilidad de  $s'$  y  $r$  sabiendo el estado  $s$  y acción  $a$  realizada.
- El segundo sumatorio calcula el valor del estado  $s$  como la suma de la recompensa  $r$  y el valor estimado actualmente para el siguiente estado  $s'$ , con su correspondiente descuento en caso de tenerlo.

Esta función se puede ejecutar iterativamente de tal forma que en cada iteración se está estimando el valor de  $v_\pi(s)$  a su valor real para cada estado

del problema. Este valor converge con  $k \rightarrow \infty$ . En la práctica, deberemos establecer un umbral ( $\theta$ ) en el que si el valor nuevo no tiene una diferencia absoluta mayor que  $\theta$ , se considere una aproximación correcta y finalice.

Al considerar el factor de aleatoriedad en reacciones del entorno ante las acciones, el valor que vamos a ir convergiendo para cada estado será la recompensa acumulada esperada que obtendremos si ejecutamos muchos episodios con esa política desde ese estado. Por lo que en el momento que hagamos más de una iteración, se está estimando las recompensas a partir de estimaciones de recompensas. Esto se le conoce como **bootstrapping** y es muy utilizado como técnica dentro del *Aprendizaje por Refuerzo*.

## MEJORAR POLÍTICA

Una vez analizadas las políticas como se explica en la subsección anterior, el siguiente paso es intentar mejorar esas políticas

Una forma muy sencilla sería generar políticas aleatoriamente para posteriormente evaluarlas y seleccionar la mejor, pero en los problemas más complejos este tipo de técnicas no son eficientes ni eficaces, por lo que se descarta el uso de estas técnicas sencillas en problemas complejos.

Existen multitud de formas más complejas que son mucho mejores para el objetivo de mejorar políticas, en este caso se va a destacar el uso de la función acción-valor. Gracias a esta función se obtienen las estimaciones de recompensas para cada acción a posible en cada estado  $s$  del entorno, por lo que se puede utilizar esa información para establecer una guía en la modificación de la política, con el objetivo de lograr una política mejor.

Al igual que en la sección anterior se aplica un algoritmo iterativo para la causa, en este caso se calcula la función estado-valor a partir de un algoritmo iterativo, a este se le denomina algoritmo de mejora de política (2.17). Destacar que es necesario que el problema se defina en MDPs.

$$\pi'(s) = \operatorname{argmax}_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')] \quad (2.17)$$

$\gamma$  corresponde al estado siguiente de  $s$  tras realizar la acción  $a$ . Básicamente consiste en buscar aquellas acciones que nos lleven a estados siguientes con la estimación de estado-valor más alta existente. Al seleccionar estas acciones, en lugar de las que seleccionaría la política  $\pi$  por sí misma, se está creando una nueva política que se denota como  $\pi'$ .

Como se dijo en subsecciones anteriores la función estado-valor calcula una estimación sumando todas las recompensas estimadas de todos los siguientes estados posibles (ecuación de Bellman), esto es lo que se utiliza

para cada acción en un estado concreto y se selecciona el máximo existente. Como consecuencia directa, esta función también es una aproximación que tratamos que sea lo más leal a la realidad posible.

Por lo tanto, se puede decir que existe un ciclo que se pueden ir repitiendo para que una política siga mejorando progresivamente, los pasos de este ciclo son:

1. Calcular la función estado-valor con el algoritmo iterativo de evaluación para una política  $\pi$ .
2. Calcular la función acción-valor con el algoritmo de mejora de política, el cuál parte de la función de estado-valor previamente calculada en el paso 1.
3. Utilizando la función acción-valor, se pude generar una nueva política.
4. Recalcular la función estado-valor con el algoritmo iterativo de evaluación para esta nueva política generada en el paso 3 y repetimos el proceso siempre que haya un cambio significativo positivo en las estimaciones de la misma (mayor que  $\theta$ ).

Al final la política converge, puesto que se llega a un punto que la política no cambia utilizando este proceso.

## **EXPLORACIÓN & EXPLOTACIÓN**

Si en los métodos para mejorar una política el número de iteraciones es infinito, el método siempre converge puesto que es algo equivalente a una búsqueda exhaustiva. El problema aparece cuando el problema que se trata es demasiado complejo y cada iteración tarda mucho tiempo en ejecutarse entonces llegar al número necesario de iteraciones para que converja es prácticamente inviable puesto que es demasiado tiempo.

Esto es el mayor problema que se encuentra al mejorar la política del agente, cuando intenta explotar su conocimiento previo estimando las recompensas (**explotación**) como se ha comentado anteriormente. Por lo que surge como solución la **exploración**, es decir, alejarse de la idea de explotar zonas prometedoras y explorar nuevos posibles desenlaces del entorno realizando acciones que no se consideran óptimas en la política actual, para que de esta forma se generen nuevos recorridos sobre los que en el futuro se pueda conseguir la convergencia con bastantes menos iteraciones que la si solo se hubiera utilizado con la estrategia de explotación. El secreto está en buscar un equilibrio entre la exploración y la explotación.

Para incorporar la técnica de la exploración a los problemas se puede hacer de varias formas:

- **Estrategias de exploración aleatoria:** esta es la estrategia más popular y que se usa con mayor frecuencia, consiste en que el agente explota su política la mayoría de veces pero en algunas ocasiones explora utilizando una acción aleatoria. Es mayormente conocido como  $\epsilon$ -Greedy, siendo  $\epsilon$  un valor muy pequeño que representa la probabilidad de utilizar la técnica de explotación en lugar de la técnica de exploración.
- **Estrategias de exploración optimistas:** esta técnica consiste en incrementar la preferencia de estados en los que hay un mayor factor de incertidumbre, es decir, los estados desconocidos por el agente, para de esta forma provocar que con mayor probabilidad las acciones del agente transiten a estos estados de mayor incertidumbre, en lugar de utilizar la acción óptima para su política dada.
- **Estrategias de exploración de información de espacio de estado:** esta técnica consiste en enriquecer la información aportada por los estados del entorno, para esto se codifica un grado de incertidumbre como parte de cada uno de los estados estados. Así el agente puede diferenciar fácilmente cuales son los estados que ha transitado y cuales son los que no ha transitado.

#### 2.3.4. MÉTODOS PARA EVALUAR Y MEJORAR LAS POLÍTICAS

El objetivo al final, es conseguir estimar las recompensas futuras de la mejor forma posible, a pesar de la incertidumbre de los problemas, de tal forma que cuanto mejor sean las estimaciones de recompensa mejor capacidad tendrá el agente por medio del *feedback* que recibe de su propio desempeño con el entorno, puesto que como es sabido la forma de aprendizaje utilizada es un aprendizaje de ensayo-error.

Por lo que el objetivo se encuentra en optimizar las estimaciones de la función acción-valor. Tener unas medias altas de estimaciones para una acción dada ayuda a generalizar ese conocimiento dejando a un lado el propio ruido del entorno.

Existen diferentes métodos o algoritmos para la realización de esta tarea de evaluar y mejorar políticas, a continuación se explican algunos de ellos, concretamente los más conocidos y utilizados. Destacar que de forma general las técnicas utilizadas en estos métodos son las descritas en subsecciones anteriores de la documentación.

**MÉTODO DE MONTECARLO PARA MEJORAR POLÍTICAS**


---

**Algoritmo 1:** Algoritmo Montecarlo para mejorar políticas, fuente ccc.inaoep

---

```

for por cada episodio do
    Determinar estado inicial de s;
    while no finalice el episodio do
        | Seleccionar una acción a desde el estado s usando  $\pi$  con
        | exploración;
    end
    for por cada par  $(s,a)$  de ese episodio do
        |  $r$  = recompensa  $r$  obtenida en la ocurrencia  $(s,a)$ ;
        | Añade  $r$  a recompensa $(s,a)$ ;
        |  $Q(s,a) = \text{promedio}(\text{recompensa}(s,a))$ ;
    end
    for para cada  $s$  en el episodio do
        |  $\pi(s) = \text{argmax}_a Q(s,a)$ ;
    end
end

```

---

El método de **Montecarlo para mejorar políticas** [24] [41] [14] consiste en ejecutar una cantidad considerable de episodios para conseguir un considerable número de trayectorias, para calcular un promedio de las recompensas acumuladas para cada estado.

La *trayectoria* se forma por el historial que se queda almacenado de cada uno de los pasos del episodio, para cada paso se almacena el estado, la acción, la recompensa y el estado siguiente obtenido, todo esto se almacena como una *tupla de experiencia*, por lo que se puede decir que una trayectoria es un conjunto de las tuplas de experiencia de cada uno de los pasos de un episodio.

Cuando se obtiene la trayectoria de un episodio, se calcula el valor de recompensa acumulada que hay desde cada estado hasta su estado final, es decir, hasta que termina el episodio. Por lo que se consigue una recompensa acumulada diferente por cada paso del episodio.

Posteriormente, se estima la función acción-valor ( $Q(s,a)$ ) calculando el promedio de recompensa acumulada obtenida en cada estado y acción de la trayectoria. Entonces, la función simplemente consiste en almacenar el valor promedio por cada estado y acción diferente del entorno, por lo que realmente es una función tabla que asocia según el estado y acción entrante una recompensa.

Para calcular el promedio, se puede hacer obteniendo la media de los valores de recompensa para cada estado y acción, o simplemente quedándose

con el primer valor de recompensa para cada estado y acción.

La limitación de este método es que solo se puede actualizar la función valor cuando el agente finaliza cada episodio. La idea de esto es que después de cada episodio, las recompensas observadas se puedan usar para evaluar la política y que esta se mejore para todos los estados visitados en el episodio.

## MÉTODO DE DIFERENCIA TEMPORAL SARSA

---

### **Algoritmo 2:** Algoritmo SARSA, fuente sedici.unlp.ar

---

```

Iniciar cada entrada  $Q(s,a)$  de forma arbitraria, para todo  $s \in S$ 
y para todo  $a \in A(s)$ ;
for por cada episodio do
    Determinar estado inicial de  $s$ ;
    while no finalice el episodio do
        Seleccionar una acción  $a$  desde el estado  $s$  usando una
        política derivada de  $Q$ ;
        while no se llegue a un estado terminal do
            Ejecutar acción  $a$ . Observar la recompensa  $r$  y el nuevo
            estado  $s'$ ;
            Seleccionar acción  $a'$  desde  $s'$  usando una política
            derivada de  $Q$ ;
             $Q(s,a)=Q(s,a)+\alpha[r+\gamma Q(s',a')-Q(s,a)]$ ;
             $s=s'$ ;
             $a=a'$ ;
        end
    end
end

```

---

El método de *Diferencia Temporal (SARSA)* [24] [41] [14] es un método “*on-policy*”, esto quiere decir que tiene una política inicial y la actualiza al final de cada episodio.

Este método propone sustituir la recompensa total del episodio por la suma de la recompensa actual que recibe el agente del entorno  $R_{t+1}$  tras realizar su acción  $A_t$  con una estimación del valor acumulado a partir del estado siguiente y ponderada con el factor de descuento  $\gamma$ . Se utilizar tanto para calcular la función estado-valor, aprendiendo a evaluar y mejorar políticas utilizada para evaluar políticas, como para calcular la función acción-valor, utilizada generalmente para mejorar dichas políticas (2.18):

$$Q(s, a) = Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)] \quad (2.18)$$

La sustitución comentada anteriormente consiste en sumarle a la estimación actual la diferencia entre su valor actual y la nueva estimación.

$\alpha$  corresponde con la importancia relativa de la nueva estimación frente a la actual, de forma que si  $\alpha$  es igual a 1 no se tiene en cuenta el valor previo de la estimación y si  $\alpha$  es igual a 0 se mantiene el valor previo y no experimenta una actualización de la estimación en este paso.

Un punto fuerte del método es que supera la limitación que presentaba el anteriormente comentado método de Montecarlo, esta limitación consistía en que la función valor solo se puede actualizar al finalizar del episodio provocando que una acción quede diluida entre todas las de su episodio o entre las acciones de todos los episodios realizados (puesto que se consideran los valores de todos los episodios ejecutados), la consecuencia indirecta es que el método puede lidiar con problemas muy complejos en términos de estados y acciones con alta incertidumbre de la transiciones de los mismos, también con problemas con *sparse rewards*, es decir, con recompensas escasas ofrecidas por el entorno al agente durante el proceso de aprendizaje (como en el conocido problema: *MountainCar*, donde solo se obtiene recompensa positiva si finaliza).

Se puede decir que los métodos de diferencia temporal convergen más rápido que el método de Montecarlo, en términos generales, puesto que el proceso de bootstrapping que implementan es efectivo, aunque el sesgo debido al uso de estimaciones sea mayor.

Aunque también es posible establecer puntos intermedios entre los métodos de Montecarlo y SARSA, la forma de establecer estos puntos medios es haciendo que no se actualicen las estimaciones de la función valor ni cuando finaliza un episodio, ni de forma inmediata cada vez que se produce un paso, en lugar de esto se actualizan cada  $n$  pasos.

## MÉTODO Q-LEARNING

El método de *Q-Learning* es un método “*off-policy*” o más conocido como un método de libre de política, lo que significa que la estimación de la política y la política de comportamiento pueden ser diferentes.

Durante las trayectorias de episodios con experiencia, en las estimaciones por paso se utiliza la siguiente función (2.19):

$$Q(s, a) = Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)] \quad (2.19)$$

En este caso para estimar las recompensas futuras en cada uno de los pasos de una trayectoria, se hace una operación muy similar a SARSA, puesto que también realiza un aprendizaje paso a paso y no episodio a episodio como en el método de Montecarlo. La diferencia viene dada en la actualización de la política, en el caso de SARSA se utiliza la diferencia de esa estimación con la suma de recompensa actual y estimación en el estado

siguiente, en este caso la estimación siguiente en realidad es otra política que elige la mejor acción que considera  $Q$  (siendo esta  $a \in A(s)$ ), en lugar de elegir la acción con la que realmente se está produciendo esa experiencia.

---

**Algoritmo 3:** Algoritmo Q-Learning, fuente sedici.unlp.edu
 

---

```

Iniciarizar cada entrada  $Q(s,a)$  de forma arbitraria, para todo  $s \in S$ 
y para todo  $a \in A(s)$ ;
for por cada episodio do
  Determinar estado inicial de  $s$ ;
  while no finalice el episodio do
    Seleccionar una acción  $a$  desde el estado  $s$  usando una
    política derivada de  $Q$ ;
    Ejecutar acción  $a$ . Observar la recompensa  $r$  y el nuevo
    estado  $s'$ ;
     $Q(s,a)=Q(s,a)+\alpha[r+\gamma \max_{a'} Q(s',a')-Q(s,a)]$ ;
     $s=s'$ ;
  end
end
  
```

---

## 2.4. REDES NEURONALES

En esta subsección se explican las *Redes Neuronales Artificiales*, puesto que en posteriores subsecciones veremos como las técnicas de *Aprendizaje por Refuerzo Profundo* se apoyan en estas técnicas.

Aunque las *Redes Neuronales Artificiales* existen desde hace años, es en la actualidad cuando están volviendo a ser utilizadas y valoradas por los logros que están consiguiendo. Por ejemplo, Google ha logrado derrotar a su propio reCAPTCHA (sistema de detección de bots) con redes neuronales, en Stanford han conseguido generar pies de fotos automáticamente, etc.

A pesar de que la idea de imitar el funcionamiento de las redes neuronales de los organismos vivos consiste en un enfoque biológico, este no ha sido especialmente útil: las Redes neuronales Artificiales han ido moviéndose para tener un enfoque matemático y estadístico. Estos enfoques se basan en una idea sencilla: dados unos parámetros hay una forma de combinarlos para predecir un cierto resultado. El problema, claro está, es que no sabemos cómo combinarlos.

Las Redes neuronales Artificiales son un modelo para encontrar esa combinación de parámetros y aplicarla al mismo tiempo. En el lenguaje propio, encontrar la combinación que mejor se ajusta es “entrenar” la red neuronal. Una red ya entrenada se puede usar luego para hacer predicciones o clasificaciones.

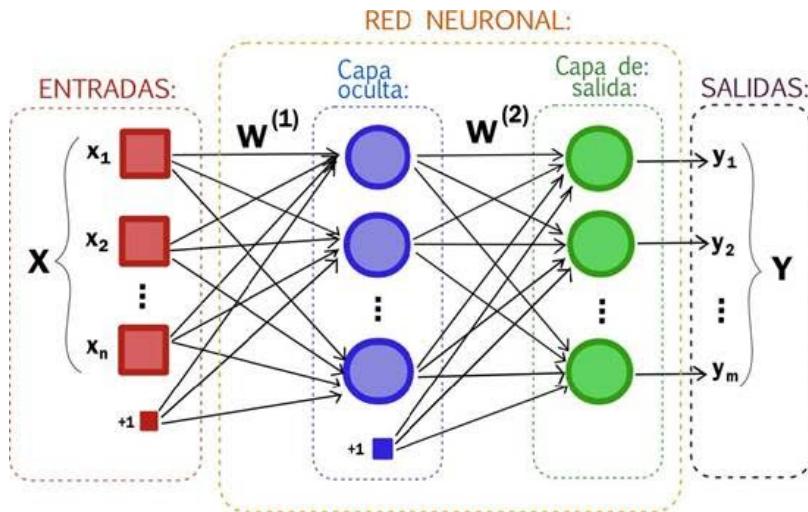


Figura 2.10: Estructura *Red Neuronal Artificial*

Las *Redes Neuronales Artificiales* se forman por un conjunto de **neuronas** o nodos interconectadas entre sí, las neuronas están organizadas por **capas**, a través de la primera capa se introducen las entradas del problema, por lo que se le denomina **capa de entrada**, mientras la **última capa** devuelve una salida interpretable, el resto de capas se les denomina **capas ocultas o intermedias**. Esta estructura se puede ver en la figura 2.10

La entrada de la *Red Neuronal Artificial* es un **vector de características** que corresponde al problema que se quiere resolver, por ejemplo si consideramos el juego del 4 en raya, el vector de características estaría formado por elementos que indicasen la pieza que hay en cada casilla respectivamente (o indicar que no hay ninguna). Por lo que se deduce que la capa de entrada tendrá tantas neuronas como elementos contenga el vector de características.

Una **Red Neuronal Artificial** [5] puede tener una o más capas ocultas, estas capas no tienen una conexión directa con el entorno como hemos visto anteriormente en la figura 2.10, tan solo tienen conexión con las neuronas de la capa anterior, siendo la primera capa o capa de entrada la única que tiene contacto con el entorno (vector de características). En el caso de solo existir una capa de entrada y una de salida, es decir, de no existir capas intermedias, las entradas serían demasiado independientes entre sí a la hora de influir en la salida, y no servirían para utilizar en los problemas del mundo real porque son mucho más complejos que eso, por lo que es necesario que la red sea capaz de detectar patrones e interdependencias entre los distintos valores de las entradas, por lo que las capas ocultas son necesarias para ayudar a representar esa complejidad del problema.

La capa de salida recoge los datos suministrados por la última capa

oculta y los procesa para conseguir una respuesta definitiva de la red. La forma en que se da esa salida puede variar dependiendo del problema en sí. La conexiones entre las neuronas de una capa con las neuronas de otra capa se les denomina **pesos** ( $W$ ), estos son utilizados por la **Red Neuronal Artificial** para calcular la salida de la siguiente capa.

#### 2.4.1. FUNCIONES DE ACTIVACIÓN

Cuando las neuronas de una capa tienen que trasmisir la información entrante no lo hacen de forma directa, sino que antes se procesa esa información entrante.

El procesamiento de esta información de entrada se hace mediante la función de activación, la cuál determina si la neurona se excita o no se excita. Destacar la existencia de multitud de funciones de activación, algunas se destacan a continuación:

- **Sigmoide:** las neuronas que tienen como función de activación: sigmoide, se comportan sumando todas las entradas ponderadas, es decir, aplicándoles sus respectivos pesos, para luego utilizar ese único valor ( $z$ ) siguiendo la siguiente fórmula (2.20):

$$\sigma(z) = \frac{1}{1 + e^{-z}} \quad (2.20)$$

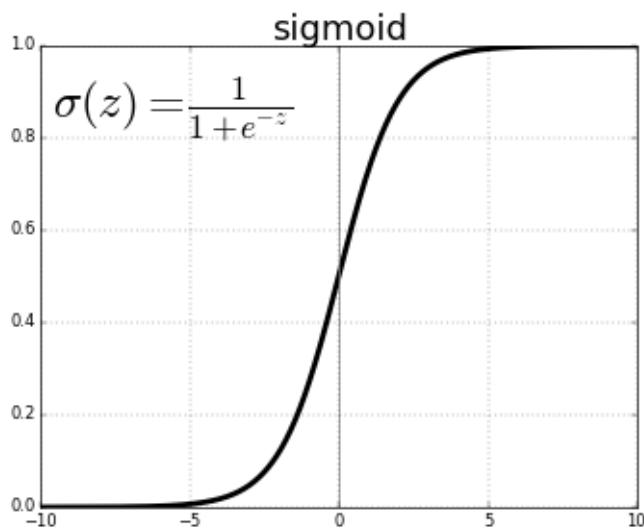


Figura 2.11: Función de activación Sísmoide

En la figura 2.11 se puede observar que la salida se acota entre los valores 0 y 1 independientemente de las entradas que pueda recibir. El valor neutro de la entrada  $z$  (valor 0) corresponde con un valor intermedio de la salida (valor 0,5). A medida que la entrada es positiva y más alta, la salida se acerca al valor 1, mientras que cuando la entrada es negativa y menor, la salida se acerca al valor 0.

Esto provoca que no haya tanta diferencia entre un conjunto de entradas muy altas a otras que no lo son tan altas.

- **Relu:** las neuronas que tienen como función de activación: **Rectified linear unit** o más conocida como relu [7], tienen una mayor sencillez para calcularla porque tiene una derivada más simple, por lo que es mejor cuando hay muchas capas en la red. Los pesos de las neuronas sigmoide son más difíciles de actualizar debido al problema de desaparición del gradiente. La función es (2.21):

$$R(z) = \max(0, z) \quad (2.21)$$

En la ecuación 2.21 se observa que si el valor de la sumas ponderadas de las entradas es menor que 0, se devuelve el valor 0 y si la suma ponderada de entrada es positiva, se devuelve tal cual.

Destacar que esta función de activación es actualmente más usada que la sigmoide. La razón es que se utilizan estos tipos de funciones debido a que permiten que las redes detecten esas relaciones no lineales entre entradas y salidas.

#### 2.4.2. FUNCIONES DE SALIDA

Las funciones de salida son funciones que se utilizan en la última capa de la *Red Neuronal Artificial*. Estas utilizan los valores que han llegado a este punto de la red con la intención de transformarlos en una respuesta final al problema que está tratando de resolver.

La función que se utiliza con mayor frecuencia es la función exponencial normalizada o más comúnmente llamada función **Softmax** (2.24). Esta se suele utilizar en problemas de clasificación con salidas discretas, ya que con esta se puede obtener una pseudo-distribución de probabilidades entre las posibles salidas:

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}} \quad (2.22)$$

Como se ha comentado antes utilizando esta fórmula se puede obtener una pseudo-distribución de probabilidades, la forma de obtener una proba-

bilidad por cada una de las clases que pertenecen al problema es utilizar una neurona de la capa de salida para cada una de las posibles clases, por lo que el valor de una neurona de la capa de salida corresponde a la probabilidad de que el ejemplo analizado pertenezca a una clase y así con todas las clases [12].

#### 2.4.3. FUNCIONES DE ERROR

El cometido de las **funciones de error** [13] es evaluar la salida producida por la red, es decir, comparar la salida obtenida por la *Red Neuronal Artificial* con la salida que debería de haber dado.

Estas funciones tratarán de cuantificar el error que ha tenido la *Red Neuronal Artificial*, algunas de estas funciones se explican a continuación:

- **Error Cuadrático:** este tipo de error mide el promedio de los errores al cuadrado, es decir, la diferencia entre el valor que debería estimar y el valor que se estima realmente:

$$MSE = \frac{1}{N} \sum_n^{j=1} (d_j - y_j)^2 \quad (2.23)$$

Siendo n el número de salidas que tiene el problema, y las salidas correctas y d las predicciones. Esta función tiene el inconveniente de que los errores más altos afectan mucho al promedio.

- **Entropía Cruzada:** : este tipo de error es un cálculo que consiste en la suma negativa del producto del logaritmo de cada componente de la salida predicha y el componente de las salida real que debería dar:

$$H(p, q) = - \sum_x p(x) \log(q(x)) \quad (2.24)$$

El objetivo que persiguen las *Redes Neuronales Artificiales* es el de minimizar la función de pérdida que utilicen puesto que esto significará un mayor rango de acierto.

Decidir una función de pérdida es muy importante a la hora de entrenar la red en un futuro, ya que dependiendo de su diseño puede acabar consiguiendo mejores o peores resultados. La elección depende de la naturaleza del problema a resolver.

#### 2.4.4. ALGORITMOS DE OPTIMIZACIÓN

En esta subsección se abordará el tema de los algoritmos de optimización utilizados, esto influye en el aprendizaje de las **Redes Neuronales Artificiales**.

ficiales [9], puesto que se ocupan en la actualización de los pesos existentes en las conexiones entre las capas de la red.

Los **algoritmos de optimización** tienen como objetivo usar los valores que devuelven las funciones de error para determinar como se van a actualizar los pesos, con la finalidad de que la *Red Neuronal Artificial* mejore los cálculos en las futuras salidas.

Existen diferentes algoritmos para realizar esta tarea, se van a explicar algunos de ellos, los más utilizados:

- **Gradiente descendiente estocástico:** esta técnica de optimización sirve para optimizar cualquier función utilizando derivadas para encontrar mínimos.

Puesto que normalmente en las *Redes Neuronales Artificiales* se trabaja con miles de parámetros, en lugar de utilizar todo el conjunto de datos de entradas para optimizar, como se utiliza en el gradiente descendente tradicional, en este caso se utiliza una muestra aleatoria del conjunto de datos de entrada en cada iteración, de tal forma que va propagando el error de la función de pérdida, de esta forma se introduce aleatoriedad lo que evita el sobreajuste y permite una mejor generalización.

También puedes configurar para que las muestras se organicen por **mini lotes** (mini batches), lo que hace que el entrenamiento sea más rápido y tenga la opción de usar el paralelismo. Esta opción consigue mejor resultado en algunos problemas, aunque existe el riesgo de caer en un mínimo local más fácilmente.

Si observamos la figura 2.4, vemos que es un ejemplo muy sencillo. Debemos tener en cuenta que sólo tiene un parámetro ( $x$ ). El caso es que en redes neuronales tenemos una función que puede tener cientos de miles de parámetros, dependiendo de la red, y que esto ni siquiera es mostrable en una gráfica ya que espacialmente no entendemos más allá de tres dimensiones (2 parámetros más el valor de salida).

- **Adam:** este algoritmo de optimización es uno de los más usados [18]. Esta basado en gradientes de primer orden, lo que implica:

- Es computacionalmente eficiente
- Tiene pocos requisitos de memoria.
- Es muy adecuado para problemas de redes con un gran número de parámetros y datos.

Adam es la abreviatura de *Momentum Adaptable*.

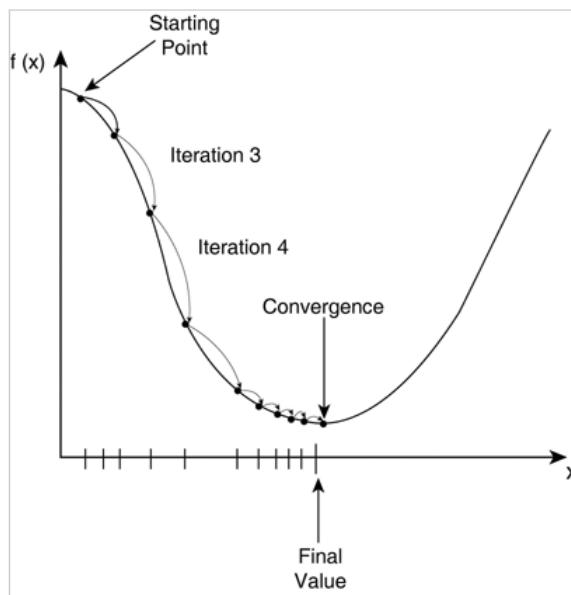


Figura 2.12: Proceso del descenso del gradiente estocástico, imagen extraída de [cs.us.es](http://cs.us.es)

## 2.5. REDES NEURONALES CONVOLUCIONALES

Las **Redes Neuronales Artificiales Convolucionales** (CNN) [6] son parecidas a las *Redes Neuronales Artificiales* convencionales, es más se puede decir que son una extensión de estas, puesto que tienen una parte densa (refiriéndose a la *Red Neuronal Artificial* convencional tal y como se han explicado hasta el momento en la subsección anterior) y una parte convolucional que está antes de la parte densa y cuyo principal objetivo es el procesamiento de imágenes, donde se extraen las características más importantes de las imágenes.

Se utilizan normalmente para trabajar con imágenes, pues es muy interesante realizar la tarea de procesamiento de imágenes previamente comentada, las imágenes se utilizan como entrada de la red, utilizando por norma general el valor de cada uno de sus píxeles (puede ser que solo sea un valor si el píxel se encuentra en escala de grises o por el contrario una tripleta de valores si está a color (RGB)). Puesto que lo normal es utilizar estas redes con imágenes, los casos reales más usuales donde se aplican este tipo de redes son: categorización de objetos, clasificación de escenas y clasificación de imágenes en general.

En las siguientes subsecciones se van a explicar las diferentes capas que forman la parte convolucional de la red y el proceso que se realiza en estas

capas desde que entran como entrada los valores de los píxeles de la imagen hasta que la imagen esta preprocesada y sirve como entrada a la parte densa de la red.

### 2.5.1. CAPA CONVOLUCIONAL

Las **Capas Convolucionales** [27] se llaman así puesto que realizan un proceso llamado convolución. Este proceso trata de transformar una imagen de entrada aplicando sobre la misma **un filtro o kernel** para conseguir una matriz de datos, como la imagen, solo que con una características concretas destacadas. Se pueden identificar infinidad de características con solo cambiar el filtro utilizado por ejemplo, se puede realizar la tarea de detección de bordes de una imagen.

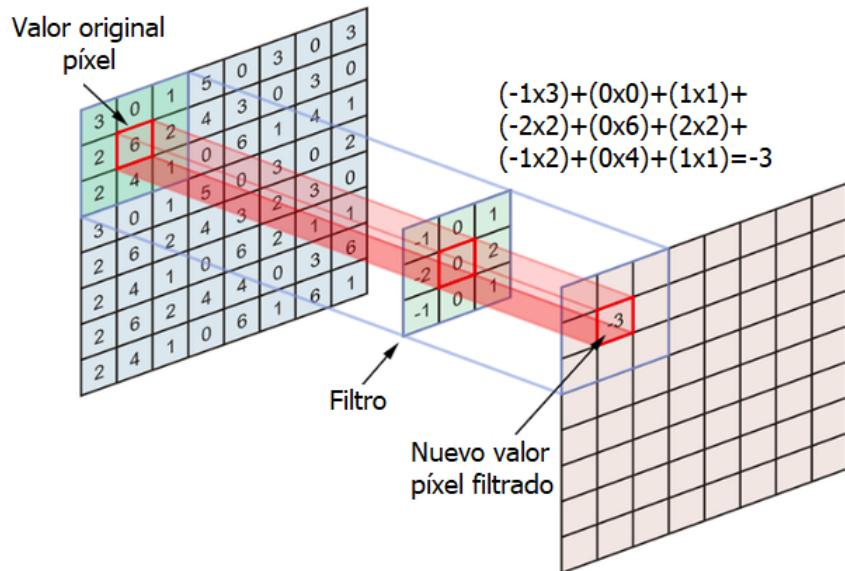


Figura 2.13: Convolución: aplicación de filtro o kernel a una matriz de píxeles (imagen)

Una capa convolucional, a su vez, puede tener más de un filtro, lo que se conoce como canales (channels) del mapa de características. La idea es que cada filtro detecte características específicas de la imagen (bordes, patrones, etc.).

Normalmente en este tipo de capas no se aplica un solo filtro o kernel sino que se aplican una serie de ellos, lo que se conoce como canales (channels) del mapa de características, utilizando cada uno de los filtros para detectar una característica concreta (bordes, patrones, etc.). Por ejemplo, si se aplican 32 filtros de 5x5 diferentes, se obtendrían 32 matrices de 24x24 a partir de

una única imagen de 28x28.

### 2.5.2. CAPA POOLING

La **Capa de Pooling** sirve para reducir la dimensionalidad de los mapas de características producidos por la *Capa Convolucional*. Por lo que esta capa se coloca a continuación de la *Capa Convolucional* aunque también puede colocarse detrás de otra *Capa de Pooling* puesto que es común colocar más de una seguidas.

Para reducir la dimensionalidad se utiliza una operación de reducción bastante simple, esta consiste en: dividir la imagen en submarcos o un conjunto de rectángulos. En cada una de estas subzonas, utiliza la operación de Max Pooling, esta operación lo que hace es seleccionar el valor del píxel más alto para representar toda esa subzona, este proceso se puede ver en la figura 2.14:

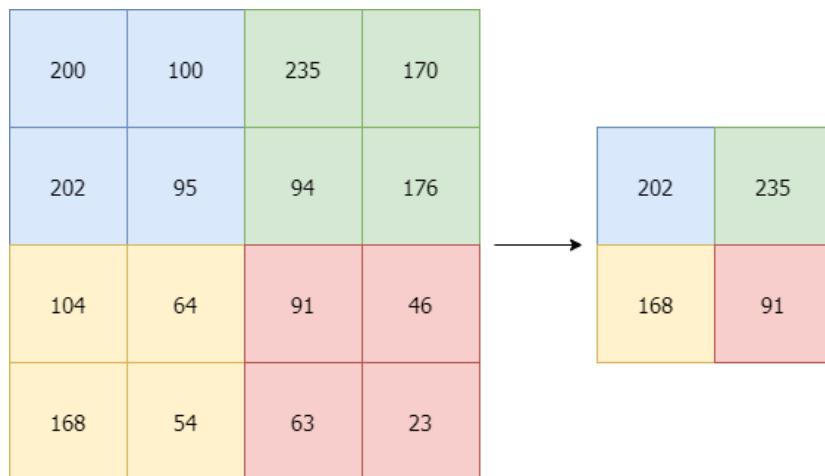


Figura 2.14: Aplicación de capa Max Pooling a mapa de características

Evidentemente después de realizar este proceso se pierde información aunque se puede pensar que esto es negativo en realidad es casi en la totalidad de los casos positivo. Porque la disminución en el tamaño o dimensión de los mapas de características produce una reducción de sobrecarga en el cálculo en las capas posteriores, haciendo que el procesamiento de imágenes en un tiempo menor. Otra ventaja es que es una representación más genérica del problema que se trata de resolver, por lo que de forma implícita se reduce el sobreajuste favoreciendo a la generalización.

### 2.5.3. CAPA FLATTEN

La **Capa Flatten** [28] coge todas las características y las concatena en un único vector de características, de tal forma que la capa de entrada de la parte densa tendrá una neurona por cada una de estas características

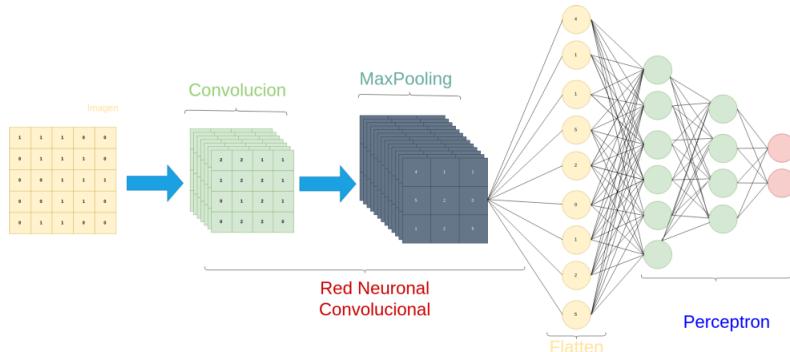


Figura 2.15: Capa Flatten en CNN, imagen extraída de mc.ai

Se suele utilizar después de una o varias *Capas Convolucionales* y *Capas de Pooling*, puesto que después de este proceso la imagen habrá sido tratada en sucesivas capas por lo que contaremos con cada vez tendremos más canales de menor dimensión con características más aisladas y específicas del problema. Por lo que para conectarlas a la parte densa se utiliza un vector de características que es conseguido por la *Capa Flatten* como se ha explicado anteriormente.

## 2.6. APRENDIZAJE POR REFUERZO PROFUNDO

En anteriores subsecciones se han explicado los conceptos y fundamentos del *Aprendizaje por Refuerzo*, por lo que los conceptos y fundamentos del *Aprendizaje por Refuerzo Profundo* se dan por sabidos al ser iguales, puesto que este tipo de aprendizaje se puede ver simplemente como una extensión del *Aprendizaje por Refuerzo*. La peculiaridad del **Aprendizaje por Refuerzo Profundo** es que como su propio nombre indica utiliza *Aprendizaje Profundo* basado en el uso de *Redes Neuronales* en la política del problema tratado. Como se comentó anteriormente, una política puede ser cualquier función por lo que si se utiliza como función un conjunto grande de pesos y una red, estamos utilizando este método.

Estas técnicas son muy parecidas a las explicadas hasta ahora, pero se utiliza una nueva tecnología para estimar los valores de función que probable-

mente sea más potente y obtenga mejores resultados. Cuando más interesa apostar por estas técnicas es en problemas complejos, no solo por su espacio de estados y acciones, sino por la cantidad de variables que pueden tener cada uno de ellos. Aunque cuando prácticamente es obligatorio apostar por este tipo de métodos es cuando nos encontramos ante un problema con un espacio de acciones continuo, puesto que este tipo de problemas no pueden ser abordados por una función tabla.

A continuación se explican algunos de los métodos o algoritmos más importantes de *Aprendizaje por Refuerzo Profundo*.

### 2.6.1. MÉTODOS BASADOS EN VALOR: DEEP Q-NETWORK (DQN)

Esta técnica fue la primera que entreno un agente de *Aprendizaje por Refuerzo* en *ATARI*, este funcionaba nutriéndose de los píxeles crudos de la propia imagen.

DQN [25] [10] es uno de los algoritmos más populares dentro del *Aprendizaje por Refuerzo Profundo*, siendo el inicio de una serie de investigaciones e innovaciones que marcaron un antes y un después en el mundo del *Aprendizaje por Refuerzo*.

Estos agentes normalmente utilizan como entrada los píxeles de las imágenes (aunque pueden utilizar una entrada de cualquier otro tipo), en el caso concreto de utilizar imágenes funciona exactamente igual que los humanos, es decir, vemos imágenes a través de la vista, posteriormente construimos una nueva experiencia a partir de estas imágenes de entrada, y finalmente tomamos una acción y observamos los resultados. Destacar que se utilizan funciones que usan *Redes Neuronales Artificiales Convolucionales*, puesto que el uso de las mismas permiten la detección de características en las imágenes (explicado con detalle en la subsección de la capa convolucional) lo que es un punto positivo, aunque también se pueden usar redes no convolucionales.

Por lo tanto se puede decir que **DQN** es una estrategia **basada en valor (value-based)** y es una evolución del ya explicado Q-Learning, utilizando como funciones de estimación de recompensa las *Redes Neuronales Artificiales*. El cambio se encuentra en la función Q, que se modela a través de una función no lineal, concretamente una *Red Neuronal Artificial* en lugar de utilizar una función-tabla.

*Las Redes Neuronales Artificiales* tienen tantas neuronas (inputs) en la capa de entrada como variables una observación del estado s o igual al vector de características procesadas si ha habido un procesamiento convolutivo previo o igual al vector de entrada del entorno en caso de que se utilice este

tipo de entrada en lugar de imágenes. Mientras que la capa de salida tiene tantas neuronas (outputs) como acciones posibles del problema.

Para elegir una acción, podríamos coger la que tuviera un valor estimado más alto, aplicar una distribución de probabilidad para favorecer la explotación, o hacer  $\epsilon$ -greedy, etc. La red tiene la forma que se puede ver en la figura 2.16.

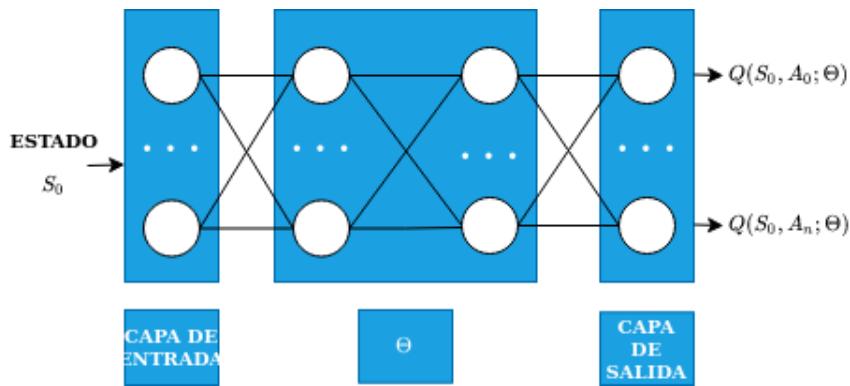


Figura 2.16: Red Neuronal Artificial para DQN en un problema con valores discretos

Indicar que  $\Theta$  se refiere al conjunto de pesos de las capas densas de la red.

En el caso de los problemas continuos, es decir, que tengan acciones continuas lo que implica problemas con un espacio infinito de valores, DQN se puede adaptar incluyendo la acción como entrada de la red. La salida en este caso es  $Q(s, a; \Theta)$  para esa acción concreta. Si queremos  $Q$  para otra acción, la parte de entrada de  $s$  sería la misma y habría que cambiar  $a$ . Como estamos operando con acciones continuas, no podemos calcular todas las posibilidades a partir de la entrada  $s$ , como se podía hacer anteriormente. Por lo que se concluye que DQN no es idóneo para resolver este tipo de problemas continuos. En este caso la red tiene la forma que se puede ver en la figura 2.17.

Aunque esta adaptación a DQN se puede combinar con otras técnicas para complementarse y que pueda funcionar bien para los problemas continuos, como en DDPG el cuál se explicará de forma breve en subsecciones posteriores. Otra posibilidad es discretizar el espacio de acciones pero esto simplifica demasiado el problema por lo que no es adecuado.

Al igual que ocurre con los métodos tradicionales de *Aprendizaje por Refuerzo* en estos métodos, también se utiliza la memoria de experiencias, para de este modo tener almacenados todas las experiencias y así poder utilizarlas para el beneficio en cualquier orden y más de una vez. De esta

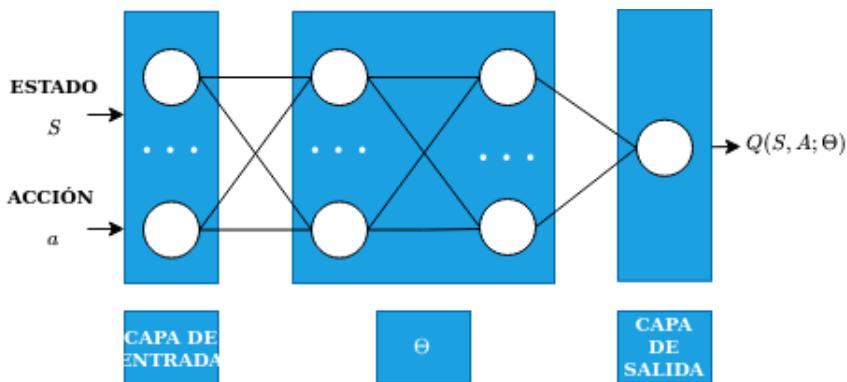


Figura 2.17: Red Neuronal Artificial para DQN en un problema con acciones continuas

forma se elimine el orden secuencial lo que repercute en que se mejore la exploración de dependencias en zonas. Con este método a diferencia de los modelos tradicionales es posible utilizar varias veces un paso o pasos de la experiencia que fuesen interesantes para aumentar la bondad del modelo.

Respecto a la función de perdida óptima y a la función de optimización que utilizan estas *Redes Neuronales Artificiales* en este contexto son desconocidas y es parte del paradigma del *Aprendizaje por Refuerzo* el seleccionar la mejor función de perdida posible y la mejor función de optimización posible. Cuando los entornos son encarecidamente complejos en ocasiones es imposible saber la respuesta óptima, y la exploración (aprendizaje ensayo-error) es la única forma de poder acercarnos o aproximarnos a la mejor respuesta posible en cada una de las situaciones planteadas por el entorno.

Por lo tanto, es necesario estimar la respuesta óptima para poder tener una función de error para la *Red Neuronal Artificial*, concretamente se estima la recompensa acumulada real que se espera del entorno una vez concluye el episodio, a partir de un estado, acción, siguiente estado y recompensa.

La función de pérdida podría ser el error cuadrático (MSE) de la siguiente forma (2.25):

$$MSE = \mathbb{E}_\pi[(Q(S, A) - Q(S, A; \theta))^2] \quad (2.25)$$

La finalidad es medir la diferencia entre el valor real de  $Q$  y el obtenido por la red que se está intentando mejorar y finalmente aplicar gradiente descendente y optimizar así la función de error. Se obtendrá una actualización de los pesos en la red ( $\Delta\theta$ ) (2.26):

$$\begin{aligned}
 \Delta\Theta &= \Delta w = -\eta \frac{\delta E}{\delta w} \\
 \frac{\delta E}{\delta w} &= 2(Q(S, A) - Q(S, A; \Theta)) \frac{\delta Q(S, A; \Theta)}{\delta w} \\
 \frac{\delta Q(S, A; \Theta)}{\delta w} &= \nabla_{\Theta} Q(S, A; \Theta) \\
 \Delta\theta &= -2\eta(Q(S, A) - Q(S, A; \Theta)) \nabla_{\Theta} Q(S, A; \Theta)
 \end{aligned} \tag{2.26}$$

Se aplica la expresión de Q-Learning, puesto que  $Q(S, A)$  no es conocido (2.27):

$$\begin{aligned}
 Q(S_0, A_0) &\leftarrow Q(S_0, A_0) + \alpha[R_1 + \gamma \max_{a \in A} Q(S_1, a) - Q(S_0, A_0)] \\
 \Delta\Theta &= \alpha[R + \gamma \max_{a \in A} Q(S', a; \Theta) - Q(S, A; \Theta)] \nabla_{\Theta} Q(S, A; \Theta)
 \end{aligned} \tag{2.27}$$

Esta formulación matemática describe la forma en que se van a modificar los pesos a partir de cada tupla de experiencia.

Se puede aplicar la técnica de mini lotes (mini-batches) extraído de la memoria de experiencias, esto significa que se seleccionan subconjuntos para realizar los entrenamientos de los pesos. Como se está actualizando una aproximación a partir de una aproximación se puede decir que se está utilizando bootstrapping.

Por esto se suelen utilizar dos redes Q. Estas redes tienen exactamente la misma arquitectura pero tienen conjuntos de pesos diferentes:

- *Red Q*: Es la red que define la función  $Q(S, A, \Theta)$ .
- *Red objetivo ( $\bar{Q}$ )*: Es un estimador imparcial del error medio cuadrático de Bellman utilizado en entrenar la red Q. Esta red objetivo se sincroniza con la red Q después de un predefinido periodo de interacciones, produciéndose un acoplamiento entre las dos redes.

De forma resumida los pesos de la red objetivo ( $\bar{Q}$ ) se actualizan con menor frecuencia a partir de los pesos de la red Q realizando copias de estos parámetros. La estimación de la recompensa acumulada se aproxima por parte de la red Q, es la red que predice las acciones y estados siguientes del entorno (2.28):

$$\Delta\Theta = \alpha(R + \gamma \max_{a \in A} \bar{Q}(S', a; \bar{\Theta}) - Q(S, A, \Theta)) \nabla_{\Theta} Q(S, A; \Theta) \tag{2.28}$$

---

**Algoritmo 4:** Algoritmo DQN

---

```

for por cada episodio do
    Inicializar contador de experiencias a 0;
    Inicializar número de iteraciones para actualizar a 0;
    while no finalice el episodio do
        Seleccionar una acción a partir de la política generadora de
        experiencia ( $Q$ );
        Ejecutar acción a. Observar la recompensa r y el nuevo
        estado  $s'$ ;
        Almacenar estado  $s$ , acción a, recompensa r y estado
        siguiente  $s$  en memoria D (de experiencias);
        Incrementar el contador de experiencias en 1;
        if contador de experiencias es igual a  $n\_batch$  then
            Muestrear el mini-batch de experiencia de las últimas
             $n\_batch$  experiencias;
            Obtener salida de predicción de cada experiencia usando
            política objetivo ( $\bar{Q}$ );
            Optimizar red generadora de experiencia ( $Q$ ) con
            gradiente descendiente utilizando la predicción obtenida;
            Fijar el contador de experiencias a 0;
        end
        Incrementar el número de iteraciones para actualizar en 1;
        if Número de iteraciones para actualizar es igual a  $C$  then
            Actualizar la red objetivo ( $\bar{Q}$ ) a partir de la política
            generadora de experiencia ( $Q$ );
            Fijar número de iteraciones para actualizar a 0;
        end
    end
end

```

---

Al igual que existe el Double Q-Learning, destacar que también existe el Double DQN. Con este método se intenta solucionar el problema de sobreajuste muy común en este tipo de métodos, puesto que la política siempre selecciona el máximo.

### 2.6.2. MÉTODOS BASADOS EN POLÍTICA

Los métodos **basados en política** [23] [14] [24] [41] o también conocidos como los métodos de política de gradiente no almacenan una política explícitamente sino que la política se encuentra codificada en la red neuronal. Contrastan con los vistos en la subsección anterior, puesto que los métodos basados en valor construyen una función acción-valor  $Q(S, A)$  que estimará la recompensa a través del estado y la acción.

En estos métodos la construcción de la política se encuentra representada de forma explícita y es almacenada en memoria durante el entrenamiento ( $\pi : s \rightarrow a$ ) sin necesidad de una función  $Q$ .

La finalidad de estos métodos no es estimar valores de recompensa a partir de estados y acciones como en los métodos basados en valor, sino que la finalidad es la de recolectar experiencia y después derivar los parámetros a una política óptima, utilizando para ello el gradiente.

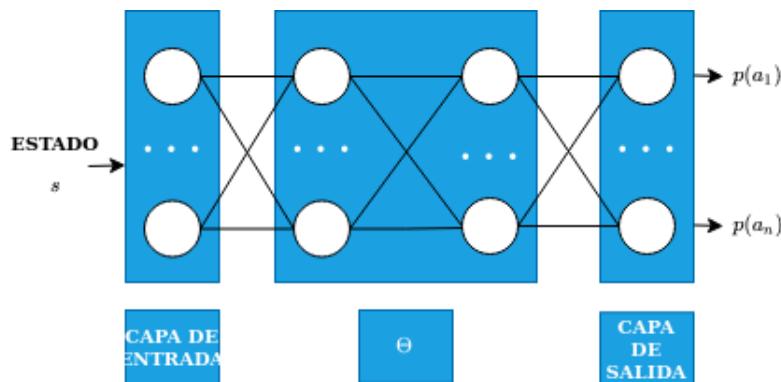


Figura 2.18: Red Neuronal Artificial para métodos basados en políticas

Por lo que se intenta calcular directamente cuál es la mejor acción dado un estado. La salida corresponde con una distribución de probabilidad de cada una de las acciones. Se puede utilizar esta distribución de probabilidades directamente como una técnica de exploración puesto que no tiene que seleccionar siempre la acción con el porcentaje más alto.

Los métodos basados en política son idóneos para utilizar en problemas con un espacio de acciones continuo, a diferencia de los métodos basados en valor, puesto que pueden ofrecer como salida directamente un valor continuo que determina la acción a realizar.

El punto negativo de los métodos basados en políticas en comparación con los métodos basados en valor es que ofrecen menos información porque no se sabe la recompensa que espera al tomar decisiones en cada momento, solamente se sabe la acción que se tiene que realizar.

Uno de estos métodos se le conoce como **REINFORCE**, este de forma resumida se basa en una red neuronal que es entrenada al final de cada episodio con valores que devuelve Montecarlo, calculando su valor de perdida y optimización utilizando el optimizador Adam, no se detalla en profundidad el funcionamiento de este algoritmo puesto que al no ser utilizado en la experimentación no es objetivo de este *Trabajo Fin de Máster*.

### 2.6.3. MÉTODOS ACTOR-CRÍTICO

Los métodos de **Actor-Crítico** (**Actor-Critic**) [36] se basa en una mezcla de los dos métodos comentados anteriormente con el objetivo de mezclar sus ventajas y de minimizar sus inconvenientes:

- *Actor*: esta parte se encarga de recoger la mejor de los métodos basados en políticas, puesto que estos de forma resumida se basan en actuar (de ahí el nombre de *actor*) puesto que lo único que les interesa a estos métodos es conseguir las acciones que obtengan los mejores resultados. Para este cometido trata de seleccionar la mejor acción posible para el estado en que se encuentra.
- *Critic*: esta parte se encarga de recoger lo mejor de los métodos basados en valor, puesto que de forma resumida se basan en estimar (de ahí el nombre de *critic*) los valores de recompensa que el agente conseguirá en un tiempo futuro para cada una de las acciones disponibles. Para este cometido indica la estimación de la recompensa acumulada a partir del estado actual ( $V(S; \Theta_v)$ ).

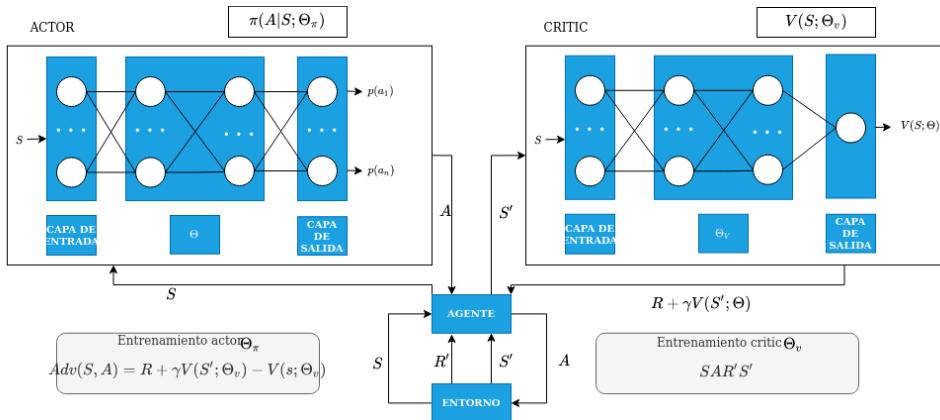


Figura 2.19: Funcionamiento método Actor-Critic

Como se puede ver en la figura anterior, es decir, en la figura 2.19, el actor recibe el estado  $s$  y devuelve la acción con la mayor distribución de probabilidad conseguida, esa acción provoca un estado siguiente y que será el que reciba la parte critic para que devuelva la estimación de recompensa acumulada que le espera, este valor vuelve a pasar por el agente.

El actor aprende teniendo en cuenta las estimaciones de critic, el cuál aprende de forma directa utilizando las ecuaciones de Bellman.

Gracias a este método podemos reducir los inconvenientes de los métodos por separado:

- Con este método se acelera el aprendizaje y se evita el estancamiento del mismo actualizando pesos durante el tránscurso del episodio, inconveniente de los métodos basados en políticas.
- Con este método se evita el sesgo de las estimaciones, inconveniente de los métodos basados en valor que se genera al calcular las estimaciones a través de estimaciones.

### A3C & A2C

A3C es un método de la familia de actor-critic [23] que es utilizado tanto para resolver problemas discretos como para resolver problemas continuos, nacido con la intención de mejorar el sesgo de las estimaciones y para explotar el procesamiento en paralelo.

El funcionamiento de **A3C** es el siguiente: tiene  $n$  agentes en diferentes entornos del problema (estos entornos son exactamente iguales), pero cada agente utiliza una política y función valor diferente, cada uno de estos agentes se les conoce como trabajadores, estos funcionan en paralelo sin tener ninguna relación entre ellos, de tal forma que cuando uno de ellos termina su cometido actualiza la política global y la función valor global a partir de la que tiene este y su experiencia.

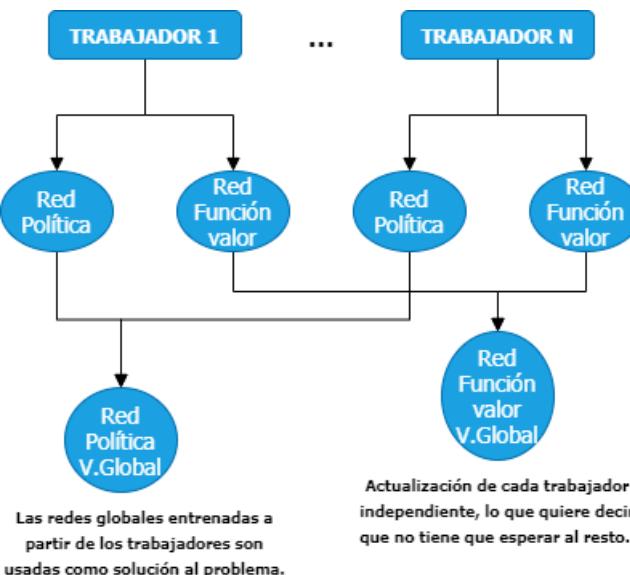


Figura 2.20: Método A3C

La actualización de la política depende de la función de optimización que tenga la *Red Neuronal Artificial*, por lo que se denota de la siguiente forma (2.29):

$$\Theta = \Theta_{old} + \alpha * \Theta_{trabajador_n} \quad (2.29)$$

$\Theta$  es el conjunto de pesos global,  $\Theta_{old}$  es el valor antiguo de estos pesos globales y  $trabajadores_n$  es el valor del conjunto de pesos de cada uno de los trabajadores.

Respecto a la función valor usa los valores de ventaja visto en el apartado de métodos de actor crítico.

Los trabajadores no se esperan entre ellos, cada uno cuando termina de entrenar n pasos hace su actualización en las redes globales. Por tanto, es muy importante mencionar que A3C tiene diseñado un mecanismo de bloqueo para que la concurrencia entre los trabajadores no provoque que sus actualizaciones se sobrescriban entre ellas en la red global.

**A2C** [25] es un método creado a partir del método A3C, la diferencia con este es que es asíncrono por lo que en este caso tan solo se utiliza una red neuronal y varios conjuntos de pesos para cada trabajador.

Los pesos de las diferentes capas de la red es compartido por los diferentes trabajadores para el actor y también para el crítico, ofreciendo las salidas al mismo tiempo. Esto se acaba utilizando en una red global al igual que ocurría con A3C.

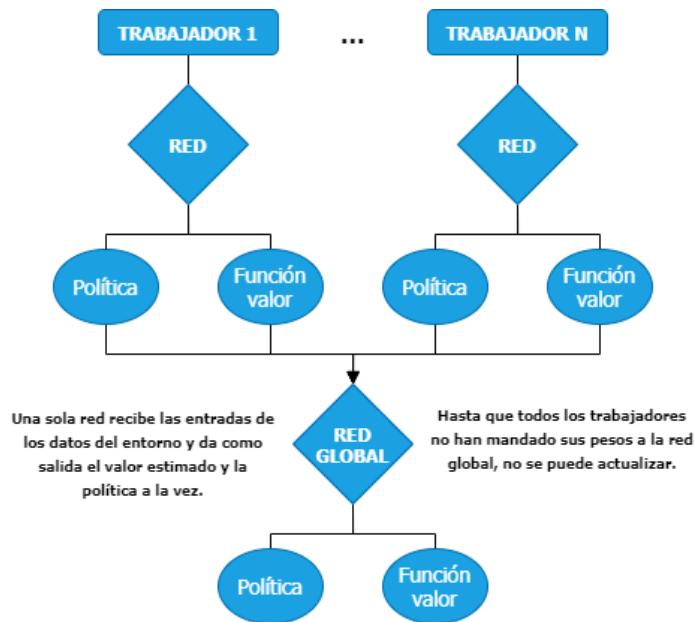


Figura 2.21: Método A2C

En A2C también existe un conjunto de trabajadores, estos se ejecutan en el mismo entorno, la diferencia se encuentra en que hasta que todos los

trabajadores no terminan no se realiza la actualización en la red global a partir de todos estos.

Cada uno de los trabajadores calcula su conjunto de pesos ( $\Theta_{trabajador_n}$ ) para posteriormente enviarlo a la red global, la cuál espera a que todos los trabajadores les manden sus conjuntos de pesos, cuando esto ocurre hace un promedio para actualizarlos y por último se inicializan los pesos de todos los trabajadores con estos valores, todo este proceso se repetirá de forma continua hasta un numero predefinido de repeticiones.

Este procedimiento obtiene mejores resultados que el A3C, no es seguro si es por la asincronía o por la exploración. También se demuestra que A2C funciona mejor con mini-lotes.

#### 2.6.4. PROXIMAL POLICY OPTIMIZATION (PPO2)

Existen una familia de métodos de **gradiente de políticas** [36] [25], los cuales utilizan los valores de ventaja producidos por la función ventaja-valor explicada con detalle en subsecciones anteriores. El funcionamiento de estos métodos se basa en que la política antigua se ejecuta durante un periodo predefinido (periodo T) (menos de un episodio), durante este periodo se estiman los valores de ventaja. Cuando este procesa termina se optimizan los pesos de la política en épocas organizadas en mini-lotes y por último se sustituye la política nueva por la política antigua.

En primer lugar, es necesario definir el ratio de probabilidad (2.30).

$$r_t(\Theta) = \frac{\pi_\Theta(A_t|S_t)}{\pi_{\Theta_{old}}(A_t|S_t)} \quad (2.30)$$

Este define cómo de grande es la actualización en el conjunto de pesos de la política. Cuanto más alejado se encuentre de 1 significa que más grande será el cambio en los parámetros, puesto que si introducimos en ese ratio la propia política antigua (sin actualizar) se obtiene (2.31):

$$r_t(\Theta_{old}) = \frac{\pi_{\Theta_{old}}(A_t|S_t)}{\pi_{\Theta_{old}}(A_t|S_t)} = 1 \quad (2.31)$$

La función de pérdida (función objetivo) es fundamental a la hora de poder optimizar una *Red Neuronal Artificial*, en este caso(2.32):

$$L^{CPI}(\Theta) = \hat{\mathbb{E}}_t \left[ \frac{\pi_\Theta(A_t|S_t)}{\pi_{\Theta_{old}}(A_t|S_t)} \hat{A}_t \right] = \hat{\mathbb{E}}_t[r_t(\Theta)\hat{A}_t] \quad (2.32)$$

$L^{CPI}(\Theta)$  es la función objetivo,  $\hat{\mathbb{E}}_t$  denota la ventaja en ese paso  $t$  y  $\hat{\mathbb{E}}$  indica la esperanza matemática empírica con un mini-lote finito de pasos.

$r_t(\Theta)$  se utiliza para penalizar en casi de que haya un cambio grande en la política.

Esto se usa en Trust Region Policy Optimization (TRPO). El principal inconveniente es que la optimización  $L^{CPI}(\Theta)$  provoca actualizaciones de política grandes en exceso si no se establecen restricciones en el proceso.

Con la intención de superar los inconvenientes nombrados anteriormente se plantea una función novedosa dentro de esta familia de algoritmos, que es una evolución de la explicada anteriormente (2.33):

$$L^{CPI}(\Theta) = \hat{\mathbb{E}}_t[\min(r_t(\Theta)\hat{A}_t, \text{clip}(r_t(\Theta), 1 - \epsilon, 1 + \epsilon)\hat{A}_t)] \quad (2.33)$$

$\epsilon$  en este caso es un hiperparámetro. Esto es conocido como objetivo sustituto recortado.

El primer término dentro del mínimo es el  $L^{CPI}$  del TRPO y el segundo término ( $\text{clip}(r_t(\Theta), 1 - \epsilon, 1 + \epsilon)\hat{A}_t$ ) se utiliza para “recortar” el ratio de probabilidad de la función objetivo.

La explicación de por qué se escoge el mínimo de estos dos valores es que se puede ignorar el cambio de ratio de probabilidad cuando hace que el objetivo mejore y se incluye el cambio de ratio cuando hace que el hace que el objetivo empeore. El valor de desventaja juega un papel fundamental en este proceso puesto que determina si está mejorando o empeorando.

En cuanto al cálculo de la ventaja en un mini-lote de tamaño T, se realiza de la siguiente forma (2.34):

$$\begin{aligned} \hat{A}_t &= \delta_t + (\gamma\lambda)\delta_{t+1} + \dots + (\gamma\lambda)^{T-t}\delta_{T-1} \\ \delta_t &= r_t + \gamma V(S_{t+1}) - V(s_t) \end{aligned} \quad (2.34)$$

Destacar que también existen más variantes de PPO, que utilizan otras funciones objetivo.

Estos métodos son muy recientes (2017-2018) y son fundamentales para los avances actuales en el uso del *Aprendizaje por Refuerzo Profundo*. Aunque presentan un desafío a la elección del tamaño de esos periodos T y tamaños de los mini-lotes anteriormente mencionados.



# Capítulo 3

## HERRAMIENTAS

### 3.1. ENTORNO: GOOGLE RESEARCH FOOTBALL ENVIRONMENT

#### 3.1.1. INTRODUCCIÓN AL ENTORNO

El entorno: **Google Research Football Environment**, es un entorno creado en 2019 por *Google AI*, una división de Google dedicada exclusivamente a la inteligencia artificial.

Este es un entorno de código abierto que tiene el propósito de investigar el *Aprendizaje por Refuerzo* [19]. *Google AI* ha proporcionado la base del código en un repositorio público de *Github* (link al repositorio) y también su trabajo de investigación al respecto.

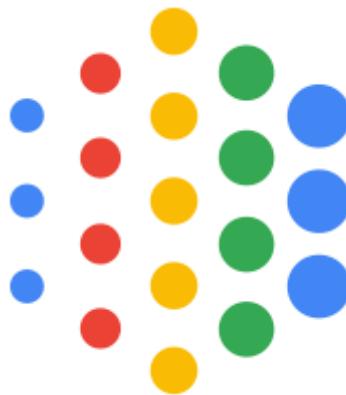


Figura 3.1: Logo Google AI

La novedad de este entorno de *Aprendizaje por Refuerzo* es que los agentes aspiran a dominar el deporte más popular del mundo: el fútbol.

Se sigue el modelo de los videojuegos de fútbol populares como es el caso de *Football Manager* o de *Soccer Manager*, el entorno proporciona una simulación de fútbol en 3D basada en la física en la que los agentes controlan a uno o todos los jugadores de fútbol del equipo, estos agentes “aprenden” a jugar para conseguir marcar goles o a impedir que los marque el equipo rival, así que depende de la situación el agente “aprende” a pasar la pelota a sus compañeros, a regatear para así poder eliminar contrarios y llegar al objetivo del gol o por el contrario el agente “aprende” a hacer entradas o presionar la salida de balón con el objetivo de evitar el gol del equipo contrario.

A continuación en la figura 3.2, se muestra una captura del entorno en pleno funcionamiento para que se pueda observar la simulación 3D proporcionada por el entorno de la que se ha hablado anteriormente, concretamente pertenece al partido por defecto que se puede iniciar con el objetivo de comprobar si ha funcionado la instalación del entorno, donde se enfrenta un equipo que actúa en consecuencia a unas normas aleatorias contra un equipo que actúa en consecuencia con un serie de reglas predefinidas.



Figura 3.2: Captura del entorno en funcionamiento

*Google Research Fotball Environment* proporciona varios componentes cruciales como son: **Google Engine**, **Google Benchmarks** y **Google Academy**.

A continuación, se explican de forma breve cada uno de los componentes mencionados [32]:

- **Football Engine:** un motor de juego altamente optimizado que simula el fútbol.

- **Football Benchmarks:** un conjunto versátil de tareas de benchmark de diferentes dificultades que se pueden usar para comparar diferentes algoritmos.
- **Football Academy:** un conjunto de escenarios de *Aprendizaje por Refuerzo* progresivamente más difíciles y diversos.

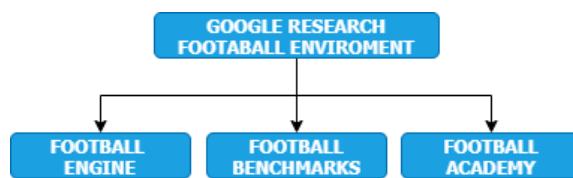


Figura 3.3: Componentes de *Google Research Football Environment*

En posteriores subsecciones se explica cada uno de estos componentes de forma detallada.

Como se ha comentado en la capítulo de introducción, los juegos proporcionan un gran entorno para los agentes de *Aprendizaje por Refuerzo*, ya que con estos se pueden probar nuevas ideas de una manera reproducible. Aplicar esos principios al fútbol parece sencillo, pero la realidad es que crear un entorno de aprendizaje reforzado para el fútbol está lejos de ser trivial y conlleva una serie de desafíos:

- *Complejidad:* si lo comparamos con la mayoría de los entornos de *Aprendizaje por Refuerzo* existentes, el fútbol no se centra en resolver una secuencia de tareas simples, sino en un grupo coordinado de tareas complejas.
- *Estocasticidad:* la mayoría de los entornos de *Aprendizaje por Refuerzo* son fundamentalmente deterministas y no manejan bien la aleatoriedad, sin embargo, el fútbol está sometido a distintas fuentes de estocasticidad, lo que aumenta drásticamente la complejidad de los algoritmos, al igual que ocurre con otros escenarios como los coches autónomos [4].
- *Multijugador:* el fútbol es lo que se conoce como un escenario de aprendizaje cooperativo y de múltiples agentes que esencialmente describe el entorno en el que los agentes necesitan colaborar y competir entre sí para lograr una serie de objetivos. Desde el punto de vista del *Aprendizaje por Refuerzo*, este tipo de entorno ofrece el mayor grado de complejidad para los agentes.
- *Totalmente observable pero continuo:* un partido de fútbol es lo que se conoce en la teoría de la *Inteligencia Artificial* como un entorno

totalmente observable que puede visualizarse en su totalidad. Esto contrasta con entornos de otros juegos como es el caso de *Dota2*, puesto que estos los agentes desconocen el entorno completo del juego. Sin embargo, la observabilidad en el fútbol se ve desafiada por la naturaleza continua del juego en el que hay movimientos casi infinitos para cualquier estado dado con el fin de lograr un objetivo específico [4].

- *Caro*: la simulación de entornos complejos como el fútbol requiere arquitecturas de GPU costosas que resultan en costos muy elevados e imposible de asumir para la mayoría de los laboratorios de investigación.

Por los motivos listados anteriormente el fútbol ha logrado eludir la gran mayoría de algoritmos de *Inteligencia Artificial*, por lo que el equipo de *Google AI* equilibró esos desafíos con modelos de *Aprendizaje por Refuerzo* de última generación que ayudan a dominar el fútbol de una manera única [32].

#### 3.1.2. FOOTBALL ENGINE

*Football Engine* es el núcleo de *Google Research Football Environment*, este componente es una versión modificada de *Gameplay Football*.

Se basa en acciones de entrada para ambos equipos y simula un partido de fútbol que incluye todos los elementos que incluye un partido de fútbol real, como son: goles, faltas, tarjetas, saques de esquina, penaltis e incluso fuentes de juego, o en contraposición una acción concreta del juego de forma independiente.

El lenguaje de programación en que esta escrito este componente es *C++* altamente optimizado, lo que permite que se pueda ejecutar tanto en máquinas que utilicen CPU como en máquinas que utilicen GPU, en ambos casos se puede activar el renderizado o no activarlo en función de si quiere o no quiere el usuario. Se puede llegar a alcanzar un rendimiento de aproximadamente 25 millones de pasos por día en una sola máquina de núcleo hexa.

*Football Engine* tiene características adicionales dirigidas específicamente al *Aprendizaje por Refuerzo* [19]:

- El entorno permite al agente “aprender” utilizando diferentes representaciones de estado que se detallan en posteriores subsecciones.
- Para investigar el impacto de la aleatoriedad, se puede ejecutar en modo estocástico donde hay aleatoriedad tanto en el entorno (esta aleatoriedad se puede ver cuando por ejemplo un disparo pega en el

poste cuando en una situación normal sería gol) como en las decisiones que toma el oponente o en el modo determinista donde no existe aleatoriedad. Es el modo estocástico el que viene por defecto y el que se suele utilizar en la mayoría de los casos.

- El entorno es compatible con la API de *OpenAI Gym* ampliamente utilizada, que agiliza su adopción en otros entornos de investigación.
- El entorno permite que los investigadores puedan hacerse una idea de como funciona el entorno, jugando entre ellos o contra sus agentes, utilizando teclados o mandos.

### 3.1.3. FOOTBALL BENCHMARK

En primer lugar, para entender esta subsección es necesario saber que es un *benchmark*. Un *benchmark* es una prueba de rendimiento realizada a conciencia en un sistema o uno de sus componentes para poder medir diferentes capacidades generando resultados.

*Football Benchmark* es un componente que lo forman un conjunto de *benchmark* para la investigación de problemas de *Aprendizaje por Refuerzo* basados en *Football Engine*. El objetivo que tienen estos *benchmarks* es jugar un juego de fútbol "estándar" contra un oponente basado en reglas fijas que fue diseñado de forma manual para este propósito. Ofrecemos tres versiones: *Football Easy Benchmark*, *Football Medium Benchmark* y *Football Hard Benchmark*, la diferencia entre estos, como su propio nombre en inglés indica, es el nivel de dificultad de cada uno de ellos.

*Google AI* proporciona como referencia los resultados obtenidos lanzando el conjunto de *benchmarks* con tres algoritmos de *Aprendizaje por Refuerzo* de última generación: *DQN*, *PPO* e *IMPALA*. *PPO* se ejecuta en múltiples procesos en una sola máquina, mientras que *IMPALA* y *DQN* se ejecutan en un clúster distribuido con 500 y 150 actores respectivamente. Se utiliza la representación del Super Mini Mapa apilada y la misma arquitectura de red en todos los algoritmos. Se utilizan ambos tipos de recompensa ("scoring" y "checkpoints"). Los resultados se obtienen después de calcular la media de 5 ejecuciones (con semillas diferentes) [19] [20].

Los resultados se pueden ver en las figuras 3.4 y 3.5.

En la figura 3.4 se representan en forma de gráfico de barras las pruebas teniendo en cuenta la recompensa "Scoring" (los goles de diferencia (goles marcados menos goles que te marcan)), mientras en la figura 3.5 se representan en forma de gráfico de barras las pruebas teniendo en cuenta la recompensa "Checkpoint" (en lugar de la diferencia de goles tiene en cuenta las ocasiones de gol). En la parte de abajo de las gráficas se puede ver el algoritmo de *Aprendizaje por Refuerzo* utilizado, concretamente *IMPALA*,

### 3.1. ENTORNO: GOOGLE RESEARCH FOOTBALL ENVIRONMENT



Figura 3.4: Gráfica de *Google AI* utilizando *Google Benchmark*: Reward = Scoring, Parte 1, extraída de [19]

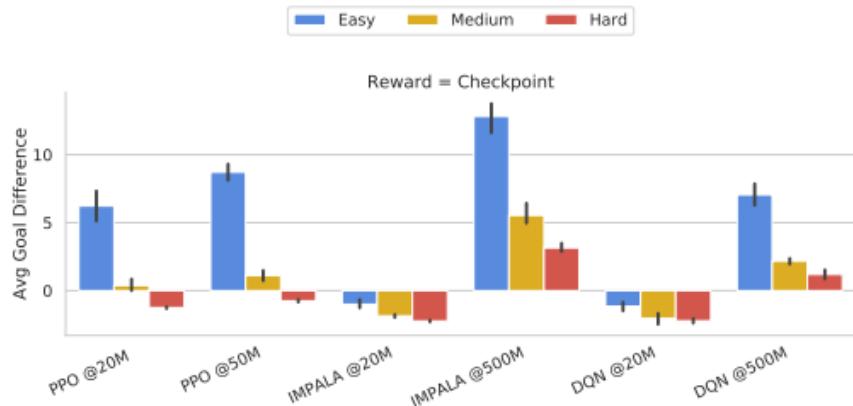


Figura 3.5: Gráfica de *Google AI* utilizando *Google Benchmark*: Reward = Checkpoint, Parte 2, extraída de [19]

*DQN* o *PPO2*, así como también se puede ver los millones de pasos que se han seguido para el entrenamiento, en este caso o 20 millones de pasos o 50 millones de pasos o 500 millones de pasos. Para cada una de estas configuraciones (algoritmo - millones de pasos) se pueden ver 3 barras, como la propia leyenda indica: la de color azul es para *Football Easy Benchmark* (nivel fácil), la de color amarillo *Football Medium Benchmark* (nivel medio) y la de color rojo *Football Hard Benchmark* (nivel difícil), las barras indican la diferencia de goles que ha conseguido el equipo después del respectivo entrenamiento.

De estos resultados proporcionados por *Google AI* se pueden obtener

algunas conclusiones como:

- La diferencia de goles se reduce con la dificultad, consiguiendo que el oponente sencillo (*Football Easy Benchmarck*) sea muy fácil de batir, consiguiendo diferencias de goles positivas muy abultadas mientras que el oponente de nivel medio (*Football Medium Benchmarck*) y de nivel difícil (*Football Hard Benchmarck*) sea bastante complicado de batir puesto que se pierden contra ellos la mayoría de los partidos de prueba, siendo el algoritmo *IMPALA* el único que consigue buenos resultados a partir de los 100 millones de pasos contra el oponente de nivel medio y contra el oponente de nivel difícil en 200 millones de pasos.
- Se ha demostrado que *Football Easy Benchmarck* es idóneo para la investigación en algoritmos que se ejecutan en una única máquina mientras que (*Football Medium Benchmarck*) y (*Football Hard Benchmarck*) son idóneos para la investigación en algoritmos que se ejecutan de forma simultánea en varias máquinas.

Se puede decir que los *benchmarks* son muy útiles, no solo en este caso sino para cualquier sistema o componente informático puesto que permite comprobar si el rendimiento es el esperado y permite comparar rendimientos. Por lo tanto, el hecho de que se incluya *Football Benchmark* en el entorno es una idea genial y de gran ayuda para los investigadores.

### **3.1.4. FOOTBALL ACADEMY**

*Football Academy* es un componente del entorno que ofrece un conjunto diverso de escenarios de diferente dificultad, lo que permite a los investigadores la posibilidad de poner en marcha nuevas ideas de investigación puesto que pueden comenzar rápidamente con ideas y repetirlas, así como les permite probar conceptos de alto nivel y proporcionar ideas de investigación como es la que se explora en este *Trabajo Fin de Máster*: hacer que los agentes “aprendan” de escenarios progresivamente más difíciles, esta idea se explica con el máximo detalle en capítulos posteriores. La ventaja de este componente es que permite a los investigadores trabajar a través de una API simple, mediante la cual pueden definir sus propias pruebas indicando para cada una de estas que escenario del entorno desea utilizar.

### **ESCENARIOS DISPONIBLES**

Para utilizar uno de los escenarios disponibles es suficiente con definirlo a la hora de crear el entorno.

### 3.1. ENTORNO: GOOGLE RESEARCH FOOTBALL ENVIRONMENT

72

Los escenarios que ofrece el entorno se listan y se describen a continuación [19] [38]:

- **11\_vs\_11\_easy\_stochastic:** este escenario del entorno propone un partido de fútbol “normal” por así decirlo, es decir, un equipo de 11 jugadores contra otro equipo de 11 jugadores, durante 90 minutos. Se define el escenario como estocástico y con una dificultad fácil.
- **11\_vs\_11\_hard\_stochastic:** este escenario es igual que el escenario anterior: *11\_vs\_11\_easy\_stochastic*, con la diferencia de que la dificultad es alta.
- **11\_vs\_11\_stochastic:** este escenario es igual que los escenarios anteriores: *11\_vs\_11\_easy\_stochastic* y *11\_vs\_11\_hard\_stochastic*, con la diferencia de que la dificultad es media.
- **academy\_3\_vs\_1\_with\_keeper:** este escenario del entorno propone una situación de 3 jugadores contra un jugador del equipo rival, nuestros jugadores parten en el borde del área: uno en el centro, otro en la izquierda y otro en la derecha, mientras que el rival estará en frente del jugador central (el cuál es quién empieza con la pelota), destacar que hay portero en el equipo rival.



Figura 3.6: Escenario: academy\_3\_vs\_1\_with\_keeper

- **academy\_corner:** este escenario del entorno propone una situación de saque de esquina.
- **academy\_counterattack\_easy:** este escenario del entorno propone una situación de contragolpe, es decir, una situación en la que el equipo

atacante acaba de robar la pelota y tiene que ser rápido para atacar al equipo defensor puesto que está descolocado. Se define la dificultad como fácil.



Figura 3.7: Escenario: academy\_corner



Figura 3.8: Escenario: academy\_counterattack\_easy

- **academy\_counterattack\_hard:** este escenario es igual que el escenario anterior: *academy\_counterattack\_easy*, con la diferencia de que la dificultad es alta.
- **academy\_empty\_goal\_close:** este escenario del entorno propone una situación de ataque en la que el jugador está en el área de penalti y

### 3.1. ENTORNO: GOOGLE RESEARCH FOOTBALL ENVIRONMENT

---

74

tiene que marcar, la portería está vacía, por lo tanto el atacante solo tiene que tirar para marcar gol.

- **academy\_empty\_goal:** este escenario del entorno propone una situación de ataque en la que el jugador está en el centro del campo y tiene que marcar, destacar que está vacía la portería, por lo tanto el atacante solo tiene que tirar para marcar gol.
- **academy\_pass\_and\_shoot\_with\_keeper:** este escenario del entorno propone una situación de ataque en la que el atacante tiene que realizar un pase y tirar a portería para batir al portero rival.
- **academy\_run\_pass\_and\_shoot\_with\_keeper:** este escenario del entorno propone una situación de ataque en la que el atacante tiene que correr, después realizar un pase y finalmente tirar a portería para batir al portero rival.
- **academy\_run\_to\_score:** este escenario del entorno propone una situación en que el jugador comienza en el centro del campo y 5 jugadores rivales lo persiguen por detrás, por lo que el atacante tendrá que correr para que no lo pillen y tirar, destacar que está la portería vacía.
- **academy\_run\_to\_score\_with\_keeper:** este escenario del entorno propone una situación en que el jugador comienza en el centro del campo y 5 jugadores rivales lo persiguen por detrás, por lo que el atacante tendrá que correr para que no lo pillen y tirar, destacar que en este caso el rival cuenta con la ayuda del portero.
- **academy\_single\_goal\_versus\_lazy:** este escenario del entorno propone un partido de 11 contra 11, donde los oponentes no pueden moverse, tan solo pueden interceptar la pelota, en el caso de que la pelota este lo suficientemente cerca de ellos. Nuestro defensa tiene el balón al principio.
- **1\_vs\_1\_easy:** este escenario consiste en un uno contra uno, es decir, un jugador de tu equipo y otro del equipo contrario, destacar que su dificultad es fácil.
- **5\_vs\_5:** este escenario consiste en un cinco jugadores de tu equipo contra cinco jugadores del equipo contrario.

Destacar la existencia de escenarios donde han de controlarse a varios jugadores de forma simultánea, como en el *Aprendizaje por Refuerzo* de múltiples agentes, por lo que se puede decir que con este *Trabajo Fin de Máster* también se incluye esta subdisciplina. Por ejemplo en el escenario “clásico” por así llamarlo, el cuál es el de un partido a 90 minutos de 11 contra 11.

*Google AI* hizo una serie de pruebas parecidas a la realizadas con los *benchmarks*, pero en este caso se probó en diferentes escenarios, con diferentes número de pasos de entrenamiento y con diferentes algoritmos [19].

En primer lugar, se muestran las figuras 3.9 y 3.10 que representan los resultados de las pruebas realizadas con el algoritmo de *Aprendizaje por Refuerzo PPO2*, utilizando como sistema de recompensa “checkpoint”.

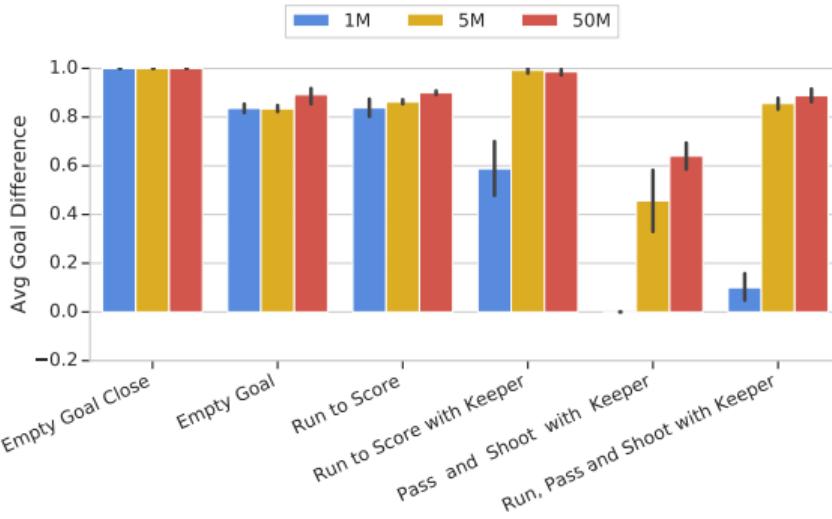


Figura 3.9: Prueba de *Google AI* escenarios con *PPO2*; Rewards = Checkpoint, Parte 1, extraída de [19]

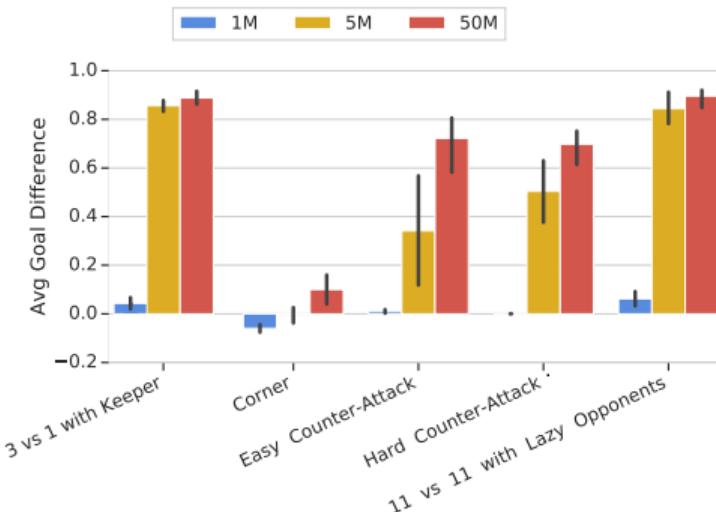


Figura 3.10: Prueba de *Google AI* escenarios con *PPO2*; Rewards = Checkpoint, Parte 2, extraída de [19]

### 3.1. ENTORNO: GOOGLE RESEARCH FOOTBALL ENVIRONMENT

**76**

---

En segundo lugar, se muestran las figuras 3.11 y 3.12 que representan los resultados de las pruebas realizadas con el algoritmo de *Aprendizaje por Refuerzo IMPALA*, utilizando como sistema de recompensa “checkpoint”.

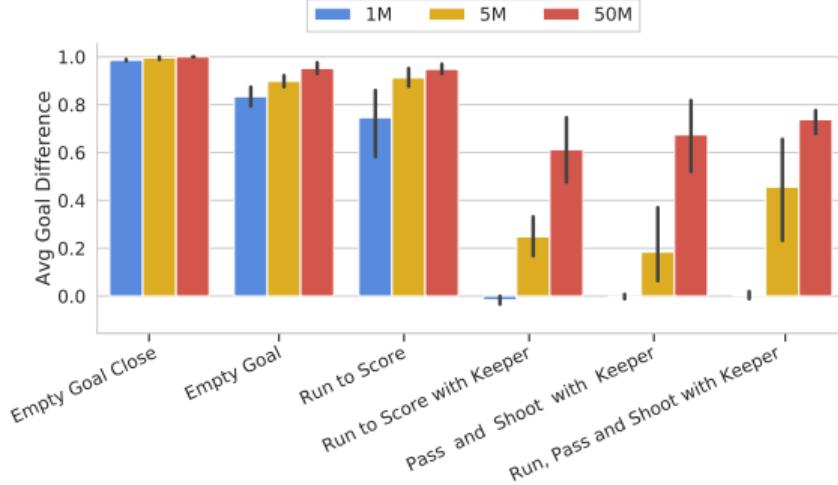


Figura 3.11: Prueba de *Google AI* escenarios con *IMPALA*; Rewards = Checkpoint, Parte 1, extraída de [19]

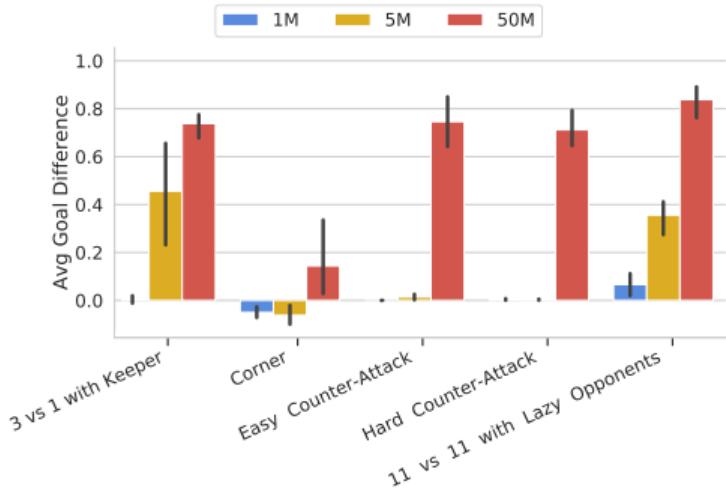


Figura 3.12: Prueba de *Google AI* escenarios con *IMPALA*; Rewards = Checkpoint, Parte 2, extraída de [19]

*Google AI* también realiza estas pruebas utilizando “Scoring” como tipo de recompensa para poder comprobar si hay diferencia de rendimiento entre usar un tipo de recompensa u otro en cada uno de los escenarios que se prueban.

En primer lugar, en las figuras 3.13 y 3.14 se pueden ver los resultados correspondientes a la ejecución del algoritmo *PPO2* con este tipo de recompensa.

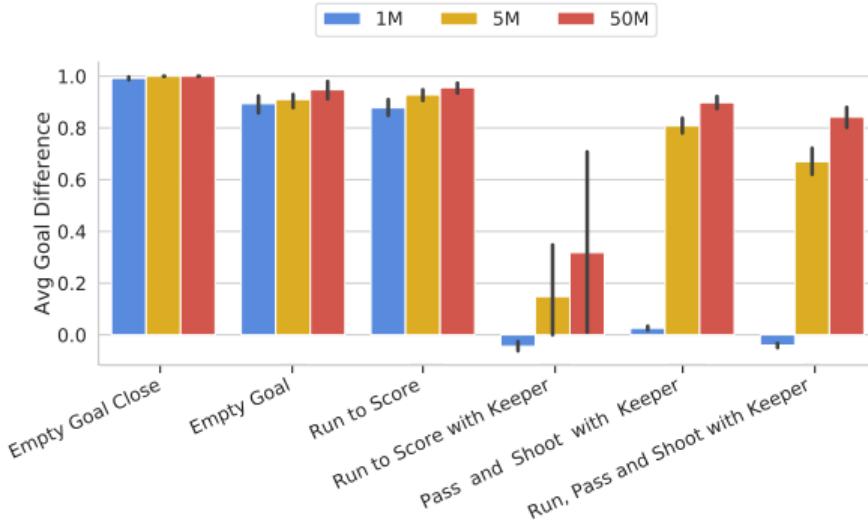


Figura 3.13: Prueba de *Google AI* escenarios con *PPO2*; Rewards = Scoring, Parte 1, extraída de [19]

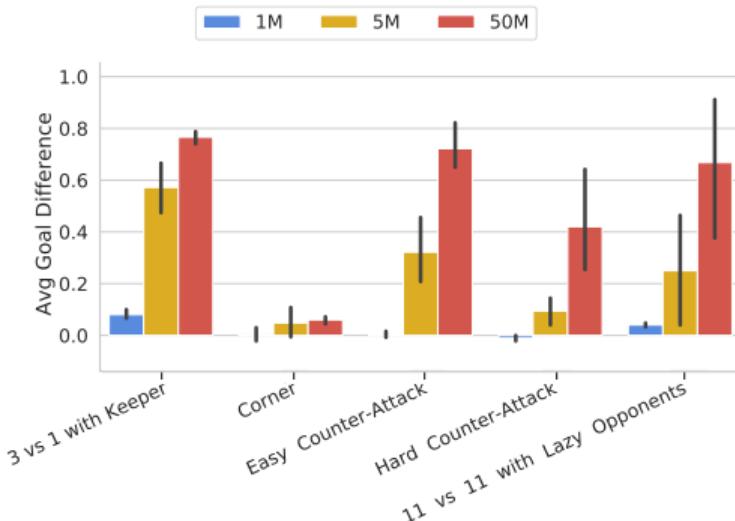


Figura 3.14: Prueba de *Google AI* escenarios con *PPO2*; Rewards = Scoring, Parte 2, extraída de [19]

En segundo lugar, en las figuras 3.15 y 3.16 se pueden ver los resultados correspondientes a la ejecución del algoritmo *IMPALA* utilizando “Scoring” como tipo de recompensa.

### 3.1. ENTORNO: GOOGLE RESEARCH FOOTBALL ENVIRONMENT

78

---

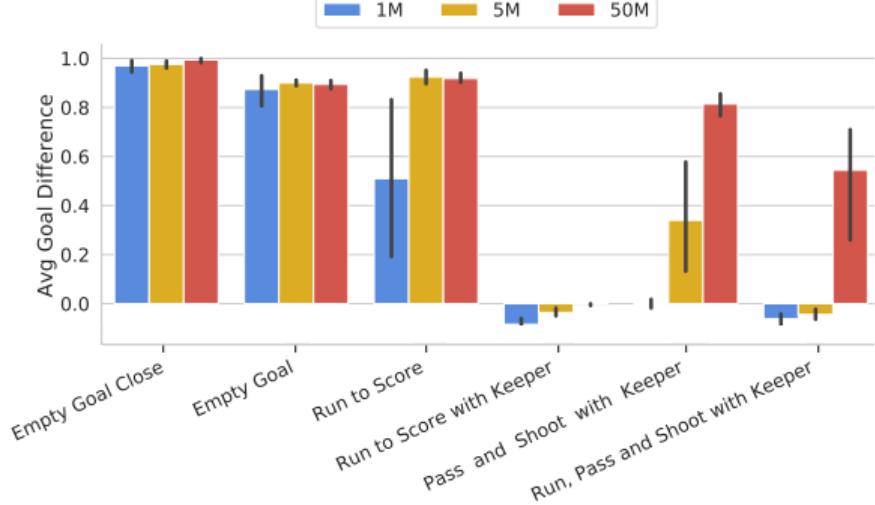


Figura 3.15: Prueba de *Google AI* escenarios con *IMPALA*; Rewards = Scoring, Parte 1, extraída de [19]

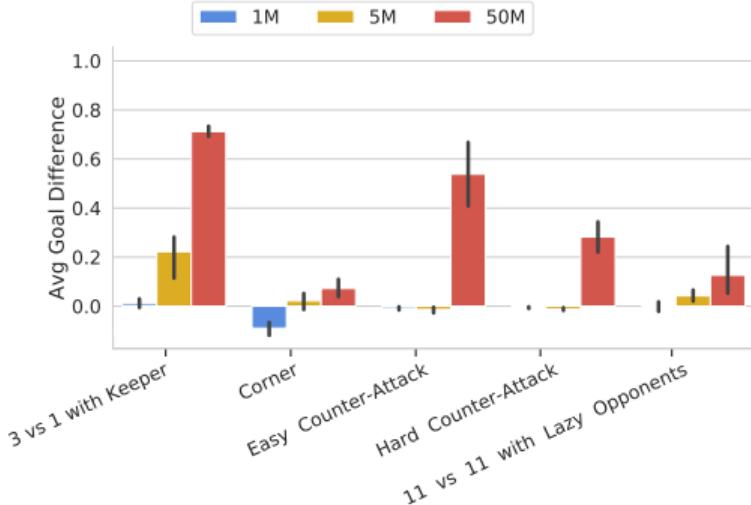


Figura 3.16: Prueba de *Google AI* escenarios con *IMPALA*; Rewards = Scoring, Parte 2, extraída de [19]

En las gráficas anteriormente mostradas, las barras representan la diferencia de goles conseguida en cada uno de los escenarios del entorno, respecto al color de las barras tiene que ver con el número de pasos de entrenamiento a que se someten los agentes, en este caso concreto: el color azul corresponde a 1 millón de pasos, el color amarillo corresponde a 5 millones de pasos y el color rojo corresponde a 50 millones de pasos.

Destacar que estos experimentos utilizan la misma configuración experimental que para los *Football Benchmarks* y destacar que las recompensas varían en función de la diferencia de goles conseguida por lo que si el equipo rival marca un gol se resta un punto a esa recompensa mientras que si el equipo dirigido por el agente marca un gol se suma un punto a la recompensa. Las recompensas intermedias se consiguen si ocurre una ocasión del gol (solo si se utiliza “checkpoints” como sistema de recompensa).

De estos resultados de las pruebas realizadas por *Google AI* se pueden obtener algunas conclusiones como:

- La diferencia de goles aumenta conforme suben los pasos de entrenamiento del agente, evidentemente puesto que está “aprendiendo” durante más tiempo, aunque sería interesante investigar si no cae en sobreentrenamiento.
- En los escenarios con niveles de dificultad, como por ejemplo “Easy Counter-Attack” y “Hard Counter-Attack”, se consiguen mejores resultados en las dificultades más fáciles.
- En los entornos más sencillos como “Empty Goal Close” y “Empty Goal” se consigue ganar el partido, es decir, conseguir una diferencia de goles positiva con tan solo 1 millón de pasos de entrenamiento.
- *IMPALA* consigue iguales o mejores resultados que *PPO2* en todos los escenarios probados.
- Se consiguen mejores resultados en todos los casos con “checkpoints” como sistema de recompensa puesto que el sistema de recompensa de “scoring” al solo tener en cuenta los goles puede dejar estancado su “aprendizaje”, estos sistemas de recompensas se detallan en posteriores subsecciones.
- El escenario “Corner” parece ser el escenario más difícil, se intuye que la explicación es que en este escenario el equipo tiene que enfrentarse a un equipo completo (11 jugadores), y estos jugadores oponentes también pueden anotar si roban la pelota después del corner.
- Se consigue ver a simple vista cuales son los entornos más sencillos y los más difíciles, por lo que se podría realizar una clasificación de los escenarios (aquellos que aparecen en las pruebas) según su dificultad, la clasificación se puede ver a continuación:
  - *Entornos Sencillos*: “Empty Goal Close”, “Empty Goal”, “Run to Score”, “Run to Score with Keeper” y “Run, Pass and Shoot with Keeper”.

### 3.1. ENTORNO: GOOGLE RESEARCH FOOTBALL ENVIRONMENT

---

- *Entornos de mayor dificultad:* “3 vs 1 with Keeper”, “corner”, “Easy Counter-Attack”, “Hard Counter-Attack” y “11 vs 11 with Lazy Opponents”.

Por último, destacar tal cuál dice el artículo de *Google AI* [19] que el tiempo de ejecución de estas pruebas es horas, días e incluso semanas, como se comentó en el capítulo de introducción el tiempo de ejecución de estos algoritmos de *Aprendizaje por Refuerzo* para conseguir ejecutar muchas épocas o iteraciones es muy alto.

#### 3.1.5. MEDIO AMBIENTE DEL ENTORNO

En esta subsección se analiza este entorno complejo para comprender su representación en el espacio de estados y en el espacio de acciones.

#### ESPACIO DE ESTADOS

El entorno *Google Research Football Environment* permite varios tipos de representaciones [8] [16], las cuales son las entradas que toman los algoritmos de *Aprendizaje por Refuerzo* para así poder “aprender”, las diferentes representaciones se pueden ver a continuación:

- **Números flotantes (simple115):** esta representación se realiza utilizando números flotantes por lo que proporciona una codificación más compacta. Consiste en un vector de 115 dimensiones que resume la gran mayoría de los aspectos del juego, estos aspectos son:
  - Información del balón:
    - Posición del balón, utilizando las coordenadas x, y e z ([x,y,z]).
    - Vector de dirección del balón, utilizando las coordenadas x, y e z ([x,y,z]).
    - Ángulos de rotación del balón, utilizando las coordenadas x, y e z ([x,y,z]).
    - Posesión del balón, número que indica a qué equipo pertenece el balón:
      - ◊ 1: equipo local.
      - ◊ 0: no pertenece a nadie.
      - ◊ -1: equipo visitante.
  - Información del equipo local:
    - Vector de N elementos, con las posiciones de los jugadores del equipo local, utilizando las coordenadas x e y ([x,y]). Normalmente N es 11 pero se puede ver reducido en algunos escenarios que no se cuente con 11 jugadores.

- Vector de N elementos con los vectores de dirección de los jugadores del equipo local, utilizando las coordenadas x e y ([x,y]).
- Vector de N elementos con números flotantes indicando el cansancio de cada jugador del equipo local, en un rango de [0-1], significando 1 que el jugador conserva la energía al completo y significando 0 que el jugador está agotado.
- Vector de N elementos que indica si un jugador tiene tarjeta amarilla o no la tiene, con un 1 se indica que el jugador tiene tarjeta amarilla o con un 0 que no tiene tarjeta amarilla.
- Vector de N elementos con valores de tipo “boolean” (Verdadero o Falso) para indicar si un jugador esta jugando el partido o no está jugando el partido (el motivo es una tarjeta roja previa).
- Vector de N elementos que indica los roles de cada uno de los jugadores del equipo local:
  - ◊ GK: portero.
  - ◊ CB: defensa central.
  - ◊ LB: lateral izquierdo.
  - ◊ RB: lateral derecho.
  - ◊ DM: mediocentro defensivo.
  - ◊ CM: mediocentro.
  - ◊ LM: interior izquierdo.
  - ◊ RM: interior derecho.
  - ◊ AM: mediapunta.
  - ◊ CF: delantero centro.
- Información del equipo visitante: esta información es exactamente igual que la del equipo local pero para el equipo visitante.
- Información del jugador controlado:
  - Número entero que indica el índice del jugador controlado.
  - Vector de A elementos de 0 o 1 que indica las acciones del jugador que están activas, estas se describen con detalle en la próxima subsección (espacio de acciones). A es igual al número de acciones.
- Información del estado del partido:
  - Par de números que indica el marcador del partido, formato: [goles equipo local, goles equipo visitante].
  - Número que indica cuantos pasos quedan hasta el final de la partida.
  - Número que indica el modo de juego:

- ◊ 0: partido normal.
  - ◊ 1: saque de centro.
  - ◊ 2: saque de puerta
  - ◊ 3: saque de falta.
  - ◊ 4: saque de esquina (córner).
  - ◊ 5: jugada en posición para tirar.
  - ◊ 6: penalti.
- **Super Mini Mapa (extracted):** esta representación consiste en utilizar el Super Mini Mapa (SMM) es básicamente una pila de matrices binarias que definen el minimapa renderizado en el centro a la parte inferior de la pantalla. La representación Super Mini Mapa consta de cuatro matrices de 96 x 72 que codifican información sobre el equipo local, el equipo visitante, el balón y el jugador activo, respectivamente. La codificación es binaria, lo que representa si hay un jugador, pelota o jugador activo en la coordenada correspondiente, o no.
  - **Píxeles (pixels):** esta representación consiste en una imagen RGB de 1280 x 720 píxeles correspondiente a la pantalla renderizada, por lo tanto para usar este tipo de representación es necesario tener activada la renderización del entorno. Esto incluye el marcador, por lo que el entorno de alguna forma puede saber el resultado del partido con todo lo que eso conlleva (por ejemplo después de muchos entrenamientos puede saber cuando es necesario atacar más o defender más en función del resultado). También incluye el super Mini Mapa (SMM) que se muestra en la parte inferior de la pantalla, por lo que con esta representación también se sabrá la posición de todos los jugadores participantes.
  - **Píxeles en blanco y negro (pixels\_gray):** esta representación es muy similar a la comentada en el punto anterior con la única diferencia que en vez de utilizar una imagen RGB de 1280 x 720 píxeles correspondiente a la pantalla renderizada se utiliza una imagen en escala de grises de 1280 x 720 píxeles correspondiente a la pantalla renderizada.

Por último, destacar que los investigadores pueden definir fácilmente sus propias representaciones en función del estado del entorno mediante la creación de contenedores.

## ESPACIO DE ACCIONES

El espacio de acciones son todas las acciones que tienen disponibles para realizar los futbolistas dentro del entorno, las diferentes acciones que lo componen son:

- **Top:** moverse hacia arriba.
- **Bottom:** moverse hacia abajo.
- **Left:** moverse hacia la izquierda
- **Right:** moverse hacia la derecha.
- **Top-Left:** moverse hacia arriba a la izquierda.
- **Top-Right:** moverse hacia arriba a la derecha.
- **Bottom-Left:** moverse hacia abajo a la izquierda.
- **Bottom-Right:** moverse hacia abajo a la derecha.
- **Short-Pass:** pasar en corto.
- **High Pass:** pasar por alto.
- **Long Pass:** pasar en largo.
- **Shot:** disparar.
- **Do-Nothing:** no hacer nada.
- **Sliding:** hacer entrada.
- **Dribble:** regatear.
- **Stop-Dribble:** parar de regatear.
- **Sprint:** correr.
- **Stop-Moving:** parar de moverse.
- **Stop-Sprint:** parar de correr.

## RECOMPENSAS

Aunque en subsecciones anteriores se ha dado una pincelada acerca de los posibles sistemas de recompensa a escoger [8], en esta subsección se explican de forma detallada.

Las formas que hay de obtener la recompensa son:

- **Diferencia de goles (“scoring”):** en este caso la recompensa se forma con la diferencia de goles:
  - Si se marca un gol a favor: +1.

### 3.1. ENTORNO: GOOGLE RESEARCH FOOTBALL ENVIRONMENT

---

- Si se encaja un gol en contra: -1.

Este tipo de recompensa puede ser difícil de observar durante las etapas iniciales del entrenamiento, ya que puede requerir una larga secuencia de eventos consecutivos hasta conseguir marcar un gol.

- **Puntos de control (“checkpoints”):** en este caso la recompensa no tiene que ver con la diferencia de goles sino que se tiene en cuenta lo cerca que está el balón de la portería contraria, es decir, la recompensa aumenta con ocasiones de gol. Más específicamente, dividimos el campo del oponente en 10 regiones de puntos de control de acuerdo con la distancia euclíadiana a la meta del oponente, la primera vez que el equipo del agente posee el balón en cada una de las regiones del puesto de control, la recompensa aumenta en +0.1. Esta recompensa adicional se otorga una vez por episodio y puede ser de hasta +1.
- **Diferencia de goles más Puntos de control (“scoring,checkpoints”):** en este caso la recompensa es una suma de las dos anteriores, por lo que la recompensa del episodio al marcar un gol desde dentro del área de penalti aumenta en dos puntos, puesto que se consigue un punto por marcar y otro punto por alcanzar la última zona de control.

#### 3.1.6. CREAR UN ENTORNO: PARÁMETROS

Una vez se ha descrito y detallado el entorno *Google Research Football Environment*, por lo que se supone que todos los conceptos teóricos del mismo están comprendidos y superados, ahora se procede a explicar los parámetros que habría que definir para crear el entorno, información extraída del repositorio de *GitHub* (repositorio).

Los parámetros necesarios a definir para crear el entorno a la medida que el usuario desee son:

- **env\_name:** este parámetro define nombre del escenario del entorno a ejecutar, se pueden comprobar los posibles escenarios en la anterior subsección de escenarios disponibles.
- **stacked:** este parámetro es de tipo “boolean”, si es *Verdadero* apila 4 observaciones del entorno, mientras que si es *Falso* no las apila, esto solo se puede usar en representaciones ”pixels”, ”pixels\_gray.” ”extracted”.
- **representation:** este parámetro define la representación del entorno utilizada, se puede comprobar las posibles representaciones en las subsección de espacio de estados.

- **rewards:** este parámetro define el tipo de recompensa a utilizar, se puede comprobar el tipo de recompensa en la subsección de recompensas.
- **write\_goal\_dumps:** este parámetro toma valor “boolean”, si es *Verdadero* se descargan trazas de hasta 200 fotogramas antes del volcado de objetivos y si es *Falso* no.
- **write\_full\_episode\_dumps:** este parámetro es de tipo “boolean”, si es *Verdadero* se deben volcar rastros para cada uno episodio.
- **render:** en esta opción se indica si se desea que el entorno sea renderizado o no. Si es *Verdadero* el renderizado se lleva a cabo y si es *Falso* no se renderiza el entorno.
- **write\_video:** este parámetro toma valor “boolean”, si es *Verdadero* se graba el vídeo al volcar un rastro y si es *Falso* no.
- **dump\_frequency:** este parámetro indica con qué frecuencia escribir volcados de información y vídeos, en términos de número de episodios.
- **logdir:** este parámetro indica el directorio utilizado para los *log*.
- **number\_of\_left\_players\_agent\_controls:** este parámetro indica el número de jugadores del equipo de la izquierda controla el agente.
- **number\_of\_right\_players\_agent\_controls:** este parámetro indica el número de jugadores del equipo de la derecha controla el agente.
- **channel\_dimensions:** este parámetro es una tupla (ancho, alto) que representa las dimensiones de Representación de píxeles o de Super Mini Mapa (SMM).
- **other\_config\_options:** este parámetro permite configurar directamente otras opciones en la configuración

### 3.1.7. JUGAR EN MODO MANUAL

A pesar de que *Google Research Football Environment* es un entorno de *Aprendizaje por Refuerzo* como ya es sabido, se pueden manejar los jugadores de tu equipo a tu antojo como si fuera un videojuego de fútbol, similar a *Pro Evolution Soccer* o *FIFA*, con las evidentes limitaciones en todos los sentidos, aunque podría a ver sido un excelente juego de fútbol hace 30 años porque si retrocedemos a esa época se pueden encontrar juegos como el que se ve en la figura 3.17. Incluso se puede configurar un partido para que el jugador humano juegue contra agentes entrenados previamente y así comprobar su comportamiento.

### 3.1. ENTORNO: GOOGLE RESEARCH FOOTBALL ENVIRONMENT



Figura 3.17: Juego del 1992: Sensible Soccer

Se ha de tener en cuenta que el juego se implementa a través del entorno, por lo que los jugadores controlados por el jugador humano usan la misma interfaz que los agentes. Una implicación importante es que hay una sola acción por cada 100 ms informada al entorno, lo que podría causar un efecto de retraso al jugar.

Las acciones que puede realizar el jugador son: moverse hacia arriba, moverse hacia abajo, moverse hacia la derecha, moverse hacia la izquierda, correr, driblar, pasar en corto, pase alto, tiro, presión, presión del portero, entrada y cambio de jugador.

#### 3.1.8. CURIOSIDAD: COMPETICIÓN KAGGLE

*Kaggle* es una página web destinada a albergar competiciones relacionadas con la *Inteligencia Artificial*, también se usa como repositorio público para almacenar conjuntos de datos y como foro para que los usuarios resuelvan las dudas pertinentes. La comunidad es muy activa y participan en los foros y las distintas competiciones.

El 29 de Septiembre de 2020 se publicó en *Kaggle*, una competición de *Google Research Football Environment* patrocinada por del equipo de fútbol inglés *Machester City F.C.* que juegan en la *Premier League*.



Figura 3.18: Escudo Manchester City F.C.

Que un equipo de esta categoría promocione este tipo de competición es un signo inequívoco de que el *Aprendizaje por Refuerzo* será muy utilizado en el futuro, por lo que promocionar este tipo de competiciones para animar a las personas a participar y avanzar en este área es un signo de ello.

Para esta competición hay dos escenarios nuevos que se pueden utilizar, estos son: *11\_vs\_11\_kaggle* y *11\_vs\_11\_competition*. Cada día, cada equipo puede enviar hasta 5 agentes a la competencia. Cada envío jugará episodios contra otros agentes que tengan una calificación de habilidad similar. Con el tiempo, las calificaciones de habilidades subirán con las victorias o bajarán con las derrotas. Cada agente enviado continuará jugando hasta el final de la competición. En la tabla de clasificación solo se muestra la puntuación del mejor agente de cada participante, aunque se puede realizar un seguimiento del progreso de todos envíos realizados. Estos agentes se entrena utilizando algoritmos de *Aprendizaje por Refuerzo*, el premio por ganar la competición es de 6000\$.

Incluso se ofrecen cupones para utilizar plataformas *Cloud* para realizar el “aprendizaje” de los agentes de forma online con bastante potencia.

Para más información de la competición, haz clic aquí.

### 3.2. GOOGLE COLAB

La experimentación se comenzó realizando en mi ordenador personal pero rápidamente se encuentran varias limitaciones que se listan a continuación:

- El tiempo invertido en entrenar el agente utilizando los diferentes algoritmos *Aprendizaje por Refuerzo Profundo* era demasiado, incluso llegando a semanas para poder concluir una única prueba.
- Imposibilidad de trabajar desde fuera de casa en el proyecto.

Por las razones listadas anteriormente se planteó la búsqueda de una alternativa a procesar los algoritmos de *Aprendizaje Automático* en mi ordenador, las alternativas planteadas son:

- Máquina Virtual en Azure utilizando contenedores.
- Google Colab.

Finalmente se selecciona *Google Colab* como alternativa para realizar el procesamiento de forma online. Las razones son:

- Los cuadernos de trabajo se organizan con celdas, los cuales sirven para introducir en ellas tanto código *Python* como texto en formato *Markdown*, lo que lo hace un *Jupyter Notebook*. Por lo tanto el código se podrá explicar en las celdas de texto y el resultado será estructurado y limpio.
- El cuaderno se puede conectar a *Google Drive* para almacenar directamente en esta plataforma la información.
- No requiere ninguna configuración o instalación adicional puesto que todo el procesamiento se realiza en la nube.
- El usuario puede elegir acelerador de hardware entre *GPU*, *CPU* o *TPU*, según las necesidades del mismo [15].
- Se puede integrar fácilmente con *Git* (herramienta de control de versiones) y *GitHub* (repositorio público).
- Se pueden utilizar comandos en *Bash* en caso de que sea necesario, simplemente poniendo “!” antes del comando, esto es muy útil por si no existe alguna librería y se quiere instalar.
- Permite la importación y exportación de cuadernos.
- Experiencia en lo que al uso de *Google Colab* se refiere.

Una de las desventajas de *Google Colab* es que el uso de sus recursos no es ilimitado. Es posible que no nos podamos conectar a una máquina con GPU o que el tiempo de ejecución durante el que podemos tener nuestros algoritmos activos llegue a su fin. Esto puede ocurrir si hay demasiados usuarios al mismo tiempo. Además, Google limita el uso que podemos hacer de estas GPU para evitar que, por ejemplo, se minen criptomonedas. Por esto, puede pasar que si estamos realizando una ejecución muy prolongada y muy potente, el entorno se desconecte.

Esto supone una gran limitación a la hora de realizar el proceso experimental, por eso se decide invertir en contratar la versión *Pro* de *Google Colab*. Aunque esta versión tampoco ofrece recursos ilimitados, sí que ofrece tiempos de ejecución más largos, la posibilidad de utilizar GPU más rápidas y más memoria RAM, por un precio muy asequible (10 euros al mes).

A partir de contratar *Google Colab Pro*, la mayoría del proceso experimental se pasa a hacer en la nube por los motivos anteriormente expuestos, que se traducen básicamente en una ganancia de tiempo, muy importante cuando hay que hacer una cantidad considerable de pruebas o experimentos. Aunque si deseamos visualizar el proceso necesitaremos hacerlo fuera de *Google Colab* puesto que no cuenta con esta posibilidad. Esto supone que

cuando queramos renderizar el entorno o entrenar los algoritmos de *Aprendizaje por Refuerzo* con las representaciones que implican el renderizado (pixels, pixels\_gray y extracted) tengamos que usar el ordenador personal obligatoriamente. Aún así *Google Colab* es fundamental para entrenar rápidamente los algoritmos utilizando la representación vectorial (single115).

### 3.3. BIBLIOTECAS DE APRENDIZAJE POR REFUERZO

Después de realizar las primeras pruebas, el siguiente paso es comenzar a utilizar algoritmos de *Aprendizaje por Refuerzo*. para eso se plantea la posibilidad de utilizar una librería por tres razones:

- Había bastantes librerías de este tipo para *Python*.
- Estas librerías cuentan con una alta cantidad de algoritmos de *Aprendizaje por Refuerzo* altamente optimizados.
- La parte más interesante de este *Trabajo Fin de Máster* es el proceso experimental y no la implementación manual de este tipo de algoritmos.

#### 3.3.1. POSIBLES ALTERNATIVAS

Entre todas las librerías candidatas, las librerías que se han investigado e incluso se ha llegado a realizar pruebas con ellas son las siguientes:

- **Acme:** es una librería de *Aprendizaje por Refuerzo* relativamente nueva que combina el *Aprendizaje Profundo* y los avances en computación creando soluciones poderosas para desafíos en el campo de la *Inteligencia Artificial*. Las ventajas que proporciona esta librería son:
  - Permite aumentar la reproducibilidad en el *Aprendizaje por Refuerzo* y simplificar la capacidad de los investigadores para desarrollar algoritmos novedosos y creativos.
  - Los algoritmos construidos son legibles, eficientes y orientados a la investigación.
  - Está preparado para permitir descripciones simples de agentes de *Aprendizaje por Refuerzo* que se pueden ejecutar en varias escalas de ejecución, incluidos los agentes distribuidos.
  - Muchas partes de código son reutilizables.

## 90 3.3. BIBLIOTECAS DE APRENDIZAJE POR REFUERZO

---

- Permite crear, probar y depurar fácilmente agentes novedosos en escenarios de pequeña escala antes de escalarlos, todo mientras usamos el mismo código de actuación y aprendizaje

Es librería relativamente nueva, por lo que se busco un par de ejemplos de algoritmos de *Aprendizaje por Refuerzo* implementados con esta librería. Finalmente se descarto precisamente por su juventud, lo que implica poquíssima documentación y la posibilidad de tener errores sin solución.

- **RLLib:** esta es una librería de código abierto de *Aprendizaje por Refuerzo* que utiliza *Python*, forma parte del proyecto *Ray*. Las ventajas de esta librería son:

- Tiene gran escalabilidad, esto es una característica muy a tener en cuenta porque los algoritmos de *Aprendizaje por Refuerzo* son muy intensos en lo que ha computación se refiere por lo que a menudo necesitan escalarse en clúster. *Ray* proporciona las bases para el paralelismo y la escalabilidad, es fácil de usar y permite que los programas de *Python* escalen en cualquier lugar, desde una computadora portátil hasta en un clúster grande.
- Proporciona una API unificada que se puede aprovechar en una amplia gama de aplicaciones.
- Integra muchos algoritmos de *Aprendizaje por Refuerzo*.
- Incluye integraciones para los marcos populares de *TensorFlow* y *PyTorch*.

Con esta librería se realizaron pruebas, partiendo desde el código de ejemplo existente en los ejemplos del propio repositorio de *Github* de *Google Research Football Environment*. Este código estaba preparado para un aprendizaje multientorno y al intentar modificarlo para utilizar entornos simples fue imposible, por lo que se descarto el uso de esta librería a pesar de funcionar perfectamente para los típicos problemas de *Aprendizaje por Refuerzo* como por ejemplo: *Mountain\_Car-v0.01*.

- **OpenAI Baselines:** esta es una librería de código abierto dedicada al *Aprendizaje por Refuerzo*, está creada por *OpenAI*, un laboratorio de investigación con sede en San Francisco cuya misión general consiste en utilizar la *Inteligencia Artificial* en beneficio de la humanidad. *OpenAI* se enfoca en todo tipo de *Inteligencia Artificial* aunque se centra en el *Aprendizaje por Refuerzo* y las *Redes neuronales*. Los puntos fuertes de esta librería son los siguientes:

- Tiene una gran comunidad activa a su alrededor.

- Existe mucha documentación acerca de esta librería, incluyendo documentación oficial.
- Hay muchos ejemplos, tanto ejemplos que forman parte de las diferentes documentaciones como ejemplos subidos por usuarios a internet, lo que facilita el entendimiento a los más nuevos, por lo que comenzar a utilizar *OpenAI Baselines* es bastante sencillo.
- Sus implementaciones están desarrolladas en *Python* el cuál es un lenguaje muy potente en campos relacionados con la *Inteligencia Artificial*.
- Es compatible con entornos *Gym* y con una gran variedad de adaptaciones de entornos lo que hace que pueda abarcar una gran cantidad de problemas.
- Se actualiza periódicamente, por lo que se solucionan los problemas que surgen en el código y tiene un extenso abanico de los algoritmos de *Aprendizaje por Refuerzo* más novedosos.

La librería de *Aprendizaje por Refuerzo* que se ha escogido para la realización del proyecto es **OpenAI Baselines** por la serie de ventajas descritas anteriormente en conjunto con las desventajas encontradas para las otras dos opciones barajadas, por lo que las técnicas explicadas en la sección de Marco Teórico serán llevadas a la práctica con esta librería.

### 3.4. MULTI-ENTORNOS

A continuación, se decidió experimentar con el número de entornos a utilizar para entrenar al agente. Para esto hay que introducir los entornos a utilizar en un vector de entornos (*SubprocVecEnv*) y este vector se utiliza como entrada para el algoritmo de *Aprendizaje por Refuerzo*.

Los entornos vectorizados, es decir, los entornos que permiten utilizar un vector de entornos como entrada en lugar de un entorno independiente, son un método para apilar varios entornos independientes en un solo entorno. En lugar de entrenar a un agente de *Aprendizaje por Refuerzo* en 1 entorno por paso, nos permite entrenarlo en N entornos por paso. Debido a esto, las acciones que se pasan a los entornos lo hacen en forma de vector (de dimensión N). Igual ocurre para las observaciones y para las recompensas.

Cuando se utilizan entornos vectorizados, los entornos se restablecen automáticamente al final de cada episodio. Por lo tanto, la observación devuelta para el i-ésimo entorno cuando *done[i]* sea *Verdadera* (se consiguió el objetivo) será de hecho la primera observación del episodio siguiente, no la última observación del episodio que acaba de terminar.

**SubprocVecEnv**, es el tipo concreto que se utiliza para la vectorización del entorno, este crea un contenedor vectorizado multiproceso para múltiples entornos, distribuyendo cada entorno en su propio proceso, lo que permite una velocidad significativa cuando el entorno es computacionalmente complejo.

Por motivos de rendimiento, si su entorno no está limitado por  $E/S$ , la cantidad de entornos no debe exceder la cantidad de núcleos lógicos en su CPU.

Utilizando *SubprocVecEnv* se realizan una serie de pruebas sobre *Google Research Football Environment* con diferentes números de entornos, se prueba con un *SubprocVecEnv* que contiene tan solo un entorno, con un *SubprocVecEnv* que contiene 2 entornos, con un *SubprocVecEnv* que contiene 4 entornos y con un *SubprocVecEnv* que contiene 8 entornos. El algoritmo utilizado para estas pruebas es el ejemplo por defecto que se encuentra en el propio código del entorno de *GitHub* (este ejecuta el algoritmo *PPO2*) utilizando 50000 como número de pasos y 0 como semilla sobre dos escenarios diferentes (11 contra 11 y 3 contra 1). Las pruebas realizadas se pueden ver a continuación en la figura 3.1.

	8 env	4 env	2 env	1 env
11_vs_11_stochastic	-1.41	-1.44	-1.5	-1.6
academy_3_vs_1_with_keeper	0.997	0.91	0.84	0.752

Tabla 3.1: Resultados Prueba multi-entornos

Como se puede ver en las pruebas, conforme más entornos se utilizan, mejores resultados y ejecuciones más rápidas se obtienen, puesto que se ejecutan varios entornos a la vez, hasta cubrir el total de timesteps, por lo que tanto en la experimentación básica como en la avanzada se decide usar 8 entornos en los casos que sea posible, puesto que en algunos algoritmos como *DQN* no se permite como entrada un *SubprocVecEnv*, por lo que para ejecutar DQN se ha de utilizar un entorno independiente.

## **Capítulo 4**

# **METODOLOGÍA DEL PROCESO EXPERIMENTAL**

Este capítulo corresponde con el proceso experimental realizado en el entorno *Google Research Football Environment* utilizando algoritmos de *Aprendizaje por Refuerzo*.

Concretamente se describe el proceso experimental que se ha seguido en detalle, desde las primeras pruebas y la implementación manual, hasta la experimentación básica y avanzada explicada con detalle.

Con el desarrollo de este proceso experimental se busca poder dar respuesta a las preguntas planteadas durante los objetivos. Por lo tanto una vez se ejecute el proceso experimental desarrollado en el presente capítulo, se pueden analizar los resultados obtenidos y sacar una serie de conclusiones de los mismos, los cuales respondan a qué algoritmo funciona mejor en cada uno de los escenarios y de la eficiencia del *Aprendizaje Progresivo*.

### **4.1. PRIMERAS PRUEBAS**

Una vez con el entorno y todas las librerías instaladas (destacar que la instalación de este se pondrá encontrar en el anexo de informática, al final de la presente memoria), se lleva a cabo una fase que consiste en realizar diferentes pruebas con el entorno.

En primer lugar, se realiza la prueba del entorno utilizando el partido por defecto que trae. En este partido se enfrenta un equipo que se actúa siguiendo un algoritmo aleatorio contra un equipo que actúa siguiendo una serie de reglas predefinidas. Una captura de este partido se puede ver en la

figura 4.1, el partido siempre lo suele ganar el equipo que actúa siguiendo una serie de reglas, de forma evidente, puesto que juegan contra un equipo que se mueve de forma aleatoria.



Figura 4.1: Partido por defecto de *Google Research Football Environment*

Posteriormente, se decide implementar el algoritmo de juego aleatorio de forma manual con la intención de seguir con el proceso de familiarización con el entorno y con la forma de utilizarlo. El código es muy sencillo y se puede visualizar a continuación:

```

1 import gfootball.env as football_env
2
3 env = football_env.create_environment(
4     env_name='academy_empty_goal_close',
5     stacked=False,
6     representation='simple115',
7     render=True)
8
9 for i in range(1, 10):
10     env.reset()
11     acc_reward = 0
12
13     while True:
14         action = env.action_space.sample()
15         observation, reward, done, info = env.step(action)
16         acc_reward += reward
17
18         if done:
19             break
20
21     print("Recompensa episodio {:d}: {:.2f}".format(i, acc_reward))
22
23 env.close()
```

Este algoritmo actúa de forma aleatoria en el escenario definido cuando se crea el entorno, en este caso en concreto está configurado para que el escenario sea “academy\_empty\_goal\_close”.

Posteriormente, se ejecutan unas cuantas de partidas de forma manual, es decir, jugándolas contra el equipo por defecto que actúa en consecuencia con una serie de reglas predefinidas, para de esta manera comprobar el funcionamiento de cada una de las acciones de los jugadores “*in situ*”, así como para tener un concepto más claro del entorno y probar la existencia de las faltas, los corner, los penaltis, etc.

Cabe destacar que se perdieron todos los partidos puesto que el control del equipo desde el teclado era demasiado complicado.

## 4.2. PRUEBAS CON DIFERENTES ALGORITMOS

Como se comentó en el capítulo anterior, *OpenAI Baselines* se selecciona como librería, por lo que se procede a realizar varias pruebas utilizando esta librería, estas pruebas consisten en la ejecución de diferentes algoritmos *Aprendizaje por Refuerzo* contenidos en esta librería sobre *Google Research Football Environment*, utilizando diferentes configuraciones de parámetros. El objetivo de estas pruebas es ir planteando el proceso experimental básico y avanzado, mas concretamente los algoritmos que se podrán utilizar.

### 4.2.1. SELECCIÓN DE ALGORITMOS A UTILIZAR

Destacar que se prueban todos los algoritmos contenidos en la librería *OpenAI Baselines* y se descubre que algunos de estos algoritmos no funcionan en este entorno, finalmente los que acaban funcionando son:

- a2c.
- ppo2.
- deepq (DQN).
- acer.
- acktr.

Para el proceso experimental se han seleccionado tres de los algoritmos listados anteriormente, los algoritmos de *Aprendizaje por Refuerzo Profundo* que se han seleccionado son: A2C, PPO2 y DQN.

### 4.2.2. PARÁMETROS DE LOS ALGORITMOS

Respecto a los parámetros de los algoritmos a utilizar, destacar la investigación que puede haber detrás de poder seleccionar los parámetros óptimos para el entorno de *Google Research Football Environment* puede ser tan grande como para que se utilice para abarcar otro *Trabajo Fin de Máster* puesto que los resultados pueden variar o no en función de estos parámetros, pero al no ser el objetivo de este *Trabajo Fin de máster* se añadirá como trabajo futuro el experimentar a conciencia con la totalidad de las diferentes configuraciones de parámetros disponibles.

Por esto muchos de los parámetros se dejan por defecto, es decir, tal cuál están para estas pruebas configurados en los algoritmos que se utilizan en el desarrollo experimental, sin embargo hay otra serie de parámetros comunes a todos los algoritmos que sí se consideran interesantes para modificar por lo que prueban diferentes valores para ellos, estos son:

- Red subyacente.
- Imputs a utilizar.
- Tipo de recompensa

Estos parámetros, al considerarse muy interesante su modificación para estudiar como varía el aprendizaje del agente, se detallan en profundidad a continuación.

#### REDES SUBYACENTES

El tipo de **Red Subyacente** que utilizan de forma interna los algoritmos de aprendizaje por refuerzo es un parámetro interesante a modificar. Las diferentes alternativas en lo que a tipo de red subyacente utilizada son:

- *mlp*: Perceptrón Multicapa, una red bastante simple pero eficaz, explicada durante la parte teórica anteriormente.
- *cnn*: Red Convolutacional, una red que incorpora una serie de capas convolucionales que se encargan de la selección de características, generalmente utilizada en imágenes. En este caso, se utilizan tanto utilizando imágenes como utilizando el vector de números flotantes que recoge la mayoría de los aspectos del juego, puesto que de alguna manera también puede recoger características de este vector, es como si resumieran todos los aspectos que recoge el vector, por lo que como se podrá ver en los resultados funciona bastante bien.

- *gfootball.impala.cnn*: Modificación de red convolucional creada para el algoritmo *IMPALA*, el cual es un algoritmo de *Aprendizaje por Refuerzo Profundo*, bastante reciente de la familia de los algoritmos actor-critic. Este algoritmo no ha sido explicado ni se ha experimentado con él, puesto que la librería *Baselines* no lo incluye, aunque sí incluye la red convolucional que se usaba en este, puesto que se demuestra que funciona eficientemente en otros algoritmos no solo en *IMPALA*. Cabe destacar que esta versión está optimizada para el entorno *Google Research Football Environment* en concreto, para utilizar la red *IMPALA* original se ha de usar “*impala.cnn*”.

Por lo tanto el proceso experimental se realiza con estas 3 redes para sacar de estas una comparación y ver las ventajas y desventajas de cada una de estas redes.

## INPUTS A UTILIZAR

Como se comentó anteriormente, debido al uso de *Google Colab* se utiliza como entrada (inputs) para los algoritmos de *Aprendizaje por Refuerzo Profundo* para el desarrollo experimental: el vector de flotantes del entorno (*simple115*), puesto que las demás opciones de entrada (inputs) requieren la renderización del entorno y esto desde *Google Colab* no se puede hacer.

Aún así se han realizado algunas pruebas desde el ordenador personal puesto que en él con Ubuntu 19.04 si se puede renderizar el entorno de *Google Research Football Environment*.

La prueba consiste en utilizar el ejemplo por defecto que se encuentra en el propio código del entorno de *Github* (este ejecuta el algoritmo *PPO2* con una red *cnn* y un escenario de 11 contra 11) utilizando 500000 como número de pasos, variando entre las diferentes opciones de entradas (inputs) a utilizar. Los resultados de las pruebas se pueden ver en la tabla 4.1.

	simple115	pixels	pixels_gray	extracted
11_vs_11_stochastic	-1.41	-1.80	-1.56	-1.66
academy_3_vs_1	0.997	0.852	0.854	0.901

Tabla 4.1: Resultados Prueba de tipo de inputs

Los resultados en este caso están más o menos a la par aunque se puede observar que utilizando “*simple115*” se consiguen levemente mejores resultados aunque la diferencia no es demasiado abultada, por lo que en este caso, en concreto, no hay demasiada diferencia entre la utilización de un tipo de entrada (input) u otra, pero esto no quiere decir ni mucho menos que esto vaya a ocurrir en todos los algoritmos, ni mucho menos en todos

los problemas, por lo que acertar con el tipo de entrada es algo importante a la hora de entrenar a los agentes. Aunque al ser una conclusión extraída de tan solo un experimento no se puede decir que sea del todo fiable, por lo que se concluye que hubiera sido interesante si *Google Colab* permitiera la renderización del entorno para poder utilizar todas estas opciones de entrada (inputs) en las diferentes experimentos realizados, para así poder realizar una justa comparación y poder sacar conclusiones más profundas y cercanas a la realidad respecto al tipo de entrada (input) a utilizar.

### TIPO DE RECOMPENSA

Respecto al tipo de recompensa a utilizar, tenemos la opción de utilizar los checkpoints explicados en anteriores secciones referentes al entorno *Google Research Football Environment* o de no utilizarlos y utilizar solamente los goles (tanto los goles a favor como los goles en contra). Para determinar qué tipo de recompensa se utiliza en el proceso experimental posterior o si se utilizan los dos tipos de recompensa, se realiza una pequeña prueba utilizando el ejemplo por defecto que se encuentra en el propio código del entorno de *GitHub* (este ejecuta el algoritmo *PPO2* con una red *cnn*) utilizando 500000 como número de pasos y 0 como semilla (seed) sobre dos escenarios diferentes (11 contra 11, 3 contra 1 y el escenario de pase y tiro), tanto usando los checkpoints (scoring,checkpoints) como sin usar los checkpoint (scoring), los resultados de esta prueba, se pueden ver en la siguiente tabla, es decir, en la tabla 4.2.

	Checkpoints,scoring	scoring
11_vs_11_stochastic	-1.41	-1.46
academy_3_vs_1	0.997	-0.02
academy_pass_and_shoot_with_keeper	0.364	0.03

Tabla 4.2: Resultados Prueba de tipo de recompensas

Después de esta prueba se puede observar que una recompensa que utiliza el sistema de puntos de control, es decir, el sistema de recompensa de checkpoints, funciona mejor que una recompensa que no tiene en cuenta este sistema, esto es algo obvio, puesto que estamos aportando una retroalimentación a la jugada ocurrida, de tal forma que se está premiando las ocasiones en las que nuestro agente se queda más cerca del área y penalizando las acciones que nuestro agente no se acerca tanto al área, puesto que en el fútbol, al igual que en nuestro entorno: el gol es más probable conforme más se acerque la jugada al área, por lo tanto una ocasión es más peligrosa conforme más cercana al área del equipo rival suceda. También se han realizado más pruebas alternando entre estas recompensas durante la fase de puesta en marcha, en todas estas a parte de reafirmar lo comentado anteriormente,

es decir, que en la gran mayoría de los casos, utilizar una recompensa con el sistema de checkpoints funciona mejor que una recompensa que no utiliza este sistema.

Se ha demostrado que en algunos de los escenarios disponibles del entorno utilizar una recompensa que no utiliza checkpoints (scoring) no consigue absolutamente nada, es decir, el agente no aprende por mucho tiempo que esté entrenado, esto es algo lógico, puesto que en escenarios más complejos como en un partido de 11 contra 11 (11\_vs\_11\_stochastic) la dificultad no es trivial y al agente le costará mucho trabajo marcar un gol, por lo que si solo se premian los goles puede que tarde muchísimo en suceder o que incluso ni siquiera lleguen a suceder, mientras que si se utiliza el sistema de checkpoints el agente sabrá que lo están haciendo bien conforme se vayan creando ocasiones de gol cada vez más cerca del área contraria (a esto me refiero con lo de retroalimentación que se ha comentado anteriormente), de tal forma, que si el agente va acumulando mejores recompensas cerca del área contraria siempre intentará realizar las acciones adecuadas para acercarse al área y en algún momento (más cercano que tardío) conseguirá meter un gol al equipo rival, elevando aún más su recompensa.

Por esto se concluye que para el proceso experimental, se utiliza la recompensa que utiliza los checkpoints (scoring, checkpoints), para así evitar consumir el tiempo que se invierte en la experimentación en pruebas que no van a llegar a ninguna parte, como son las que utilizan un sistema de recompensa sin checkpoint (scoring).

### 4.3. EXPERIMENTACIÓN BÁSICA

La primera parte del proceso experimental corresponde con la experimentación básica. En esta se realizan una serie de pruebas totalmente independientes unas de las otras, probando los diversos escenarios existentes en el entorno de *Google Research Football Environment* con los diferentes algoritmos de *Aprendizaje por Refuerzo Profundo* de la librería *Baselines* seleccionados, alternando con las diferentes configuraciones de los parámetros que se ha decidido variar tal y como se ha comentado en secciones anteriores.

Los objetivos de este proceso experimental básico son los siguientes:

- Comprobar cómo se comportan y cómo de buenos son los agentes entrenados utilizando los diferentes *Algoritmos de Aprendizaje por Refuerzo Profundo*.
- Comprobar la dificultad de cada uno de los diferentes escenarios del entorno en los cuales se van a realizar pruebas.

A continuación se enumera de forma resumida en que se basa la experimentación básica.

- Se realizan pruebas en todos los escenarios del entorno *Google Research Football Environment*, exceptuando los que han sido creados recientemente para la actual competición de *Kaggle*.
- Como algoritmos de *Aprendizaje por Refuerzo Profundo* se utilizan las siguientes alternativas: A2C, PPO2 y DQN.
- Como redes sudyacentes se utilizan: MLP, CNN y una adaptación de la CNN (*IMPALA*).
- Como tipo de recompensa para todas y cada una de las pruebas se utiliza el sistema de checkpoints junto con el típico sistema scoring (diferencia de goles).
- En los algoritmos que sea posible (PPO2 y A2C) se utilizan entornos multiproceso, concretamente utilizando *SubprocVecEnv* con 8 entornos y en los algoritmos donde no sea posible (DQN) se utilizan entornos de forma individual y simple.
- El número de pasos de entrenamiento (“timesteps”) es de 1 millón en todas las pruebas, puesto que se considera un número suficiente de pasos de entrenamiento para alcanzar los objetivos de este proceso experimental básico, además también influye un tema de tiempo, puesto que si cada experimento se ejecuta durante 10 millones de pasos de entrenamiento se necesitaría algo menos de un año para poder realizar todo el proceso experimental.
- Se utiliza como semilla (seed) 0, el fin de esto es tener experimentos que sean totalmente reproducibles, característica muy útil a la hora de realizar un proceso experimental.

Destacar que se realizan todas las combinaciones posibles de las diferentes alternativas que se han destacado anteriormente, por ejemplo, sobre el escenario de 11 contra 11 se ejecuta tanto el algoritmo A2C, como el algoritmo PPO2, como el algoritmo DQN y cada uno de estos algoritmos se ejecuta sobre cada tipo de red sudyacente contempladas como alternativas, es decir, MLP, CNN y la modificación de CNN (*IMPALA*).

Por lo que el proceso experimental básico o experimentación básica se puede resumir utilizando la figura 4.2 donde se pueden ver un conjunto de hexágonos organizados en columnas, representando cada una de estas columnas las diferentes posibilidades barajadas respecto a una opción en concreto en lo que a opciones a la hora de lanzar un experimento se refiere, de

## EXPERIMENTACIÓN BÁSICA

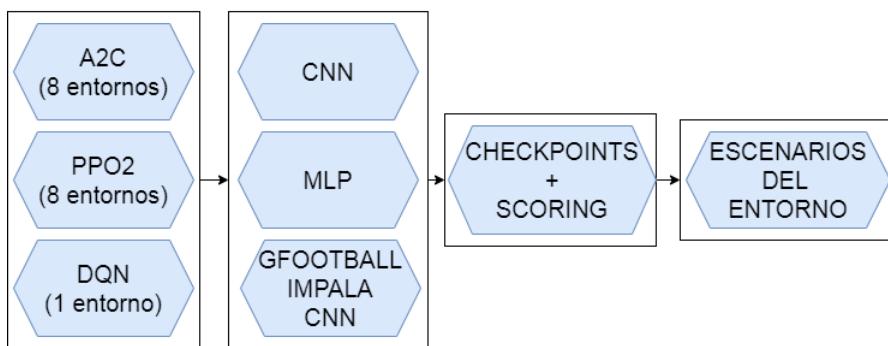


Figura 4.2: Proceso experimental básico de forma visual

tal forma que en la primera columna se pueden ver los diferentes algoritmos utilizados juntos con el número de entornos a utilizar en ese algoritmo en concreto, en la siguiente columna se puede ver los diferentes tipos de redes subyacentes utilizadas, en la siguiente columna se puede ver el sistema de recompensa utilizado (checkpoints) y en la última columna se ve un solo hexágono que representa todos los escenarios posibles del entorno donde realizar las pruebas.

Al final se realizan un total de 144 pruebas, cada una de estas pruebas es una de las posibles combinaciones resultante de unir los hexágonos de las columnas en la figura 4.2, es decir, el resultado de unir una serie de posibles alternativas para lanzar un experimento. Respecto al tiempo, destacar que ha sido elevado, aunque en el momento que se automatizaron todas estas pruebas a partir de bucles, se agilizó bastante en proceso, aunque la limitación de *Google Colab Pro* siempre ha estado presente. Esta limitación consiste en desconectar el entorno de vez en cuando, una vez el tiempo de ejecución es demasiado continuado.

El código del proceso experimental completo se puede encontrar en el siguiente enlace de *GitHub*: [Repositorio de GitHub](#).

### 4.4. EXPERIMENTACIÓN AVANZADA

La segunda parte del proceso experimental corresponde con la experimentación avanzada, en esta se realizan una serie de pruebas que se explican con detalle posteriormente

El objetivo de estas pruebas es poder comprobar si un aprendizaje progresivo es eficaz y si es más eficaz que un entrenamiento directo, tal y como

se hacia en la parte correspondiente a la experimentación básica. Con lo de aprendizaje progresivo se refiere a que un agente sea entrenado primero en escenarios más sencillos y progresivamente se vaya entrenando en escenarios más complicados utilizando la experiencia que previamente ha conseguido de los escenarios más sencillos, hasta finalmente entrenarlo en el escenario más difícil posible.

Este tipo de aprendizaje, es decir, el aprendizaje progresivo está a la orden del día actualmente en el campo del *Aprendizaje por Refuerzo* y ha sido tema de algunos de los más novedosos artículos científicos escritos por personas expertas en la materia. A este tipo de aprendizaje se le conoce como “**Curriculum Learning**” y ha conseguido tanta fama debido a los éxitos recientes conseguidos.

Estos métodos configuran las trayectorias de aprendizaje de los agentes desafiándolos con tareas adaptadas a sus capacidades. En los últimos años, se han utilizado para mejorar la eficiencia de la muestra y el rendimiento asintótico, para organizar la exploración, para fomentar la generalización o para resolver problemas de recompensas dispersas, entre otros [29].

Este tipo de aprendizaje esta inspirado en el aprendizaje humano, puesto que este se organiza en un plan de estudios de situaciones de aprendizaje interdependientes de diversas complejidades.

Para poder realizar este tipo de aprendizaje en primer lugar, es necesario definir la dificultad de cada uno de los escenarios, de tal forma que se pueda visualizar una secuencia de escenarios de dificultad progresiva, que comience por el escenario más sencillo y acabe por el escenario más complicado, por lo que con el proceso experimental básico se fundan las bases de este proceso experimental avanzado, puesto que uno de los objetivos del proceso experimental básico como se comentó anteriormente es el de evaluar la dificultad de cada uno de los diferentes escenarios del entorno, para así poder comprobar cuales son los escenarios más difíciles y cuales son los escenarios más fáciles. Así se puede establecer una escala progresiva de dificultad en lo que a escenarios disponibles del entorno se refiere, aunque para más ayuda algunos de estos escenarios ya definen el nivel de dificultad en el propio nombre (alternando entre nivel fácil, nivel difícil y nivel medio).

Las pruebas o experimentos que se hacen durante esta parte del desarrollo experimental son los siguientes (se destaca que en este caso al igual que ocurre en la experimentación básica se utiliza como semilla (seed) 0):

- Pruebas que consisten en definir varios conjuntos de escenarios de dificultad progresiva, con la idea de que el agente sea entrenado en los escenarios de este conjunto de escenarios, entrenando de forma progresiva como se ha explicado anteriormente. Para poder comparar el resultado final del agente entrenado de forma progresiva obtenido en

el escenario con la dificultad más alta contra un agente entrenado de forma directa en ese escenario de dificultad más alta.

- Posteriormente se realizan “ciclos” en algunos de estos conjuntos de escenarios de dificultad progresiva, para comprobar como funcionan de esta forma los agentes y poder sacar más conclusiones interesantes.

Teniendo en mente, de forma aproximada la escala progresiva de dificultad de los diferentes escenarios del entorno y sabiendo como funciona el fútbol, en el sentido de que por ejemplo, se entiende que para ser un buen delantero es necesario aprender a marcar goles y aprender a desmarcarse utilizando la velocidad; se plantean una serie de conjuntos de escenarios de dificultad progresiva, estos se van a explicar a continuación.

## EXPERIMENTACIÓN AVANZADA

### CONJUNTO 1 DE DIFICULTAD PROGRESIVA

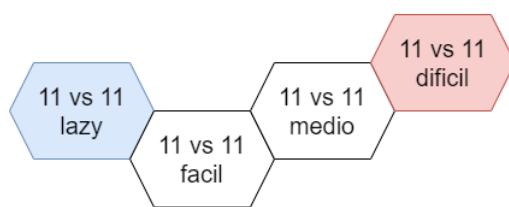


Figura 4.3: Conjunto 1 de escenarios de dificultad progresiva

El primer conjunto de escenarios de dificultad progresiva definido es el de la figura 4.3, este corresponde con escenarios de 11 contra 11 jugadores, es decir, lo que sería un partido normal o estándar de fútbol, los diferentes escenarios que forman este conjunto son:

- *academy\_single\_goal\_versus\_lazy*: este es un escenario de 11 jugadores contra 11 jugadores con la peculiaridad de que el equipo rival es como su nombre dice: “perezoso”, en el sentido de que no se mueven a no ser que los jugadores de nuestro equipo pasen muy cerca de ellos, por esto se considera que este es el entorno más sencillo de este conjunto de entornos y desde el cuál hay que partir.
- *11\_vs\_11\_easy\_stochastic*: este es un entorno de 11 jugadores contra 11 jugadores, destacar que en este escenario los jugadores rivales juegan en nivel fácil.
- *11\_vs\_11\_stochastic*: este es un entorno de 11 jugadores contra 11 jugadores, destacar que en este escenario los jugadores rivales juegan en nivel medio.

- *11\_vs\_11\_hard\_stochastic*: este es un entorno de 11 jugadores contra 11 jugadores, destacar que en este escenario los jugadores rivales juegan en nivel difícil.



Figura 4.4: Conjunto 2 de escenarios de dificultad progresiva

El segundo conjunto de escenarios de dificultad progresiva definido es el de la figura 4.4, este corresponde con escenarios contragolpe, es decir, situaciones rápidas de ataque de nuestro equipo en superioridad numérica, esto por ejemplo puede suceder después de un córner en contra si robamos la pelota y pasamos rápidamente al ataque, los diferentes escenarios que forman este conjunto son:

- *academy\_counterattack\_easy*: este escenario corresponde a una situación de contragolpe, donde los jugadores rivales juegan en nivel fácil.
- *academy\_counterattack\_hard*: este escenario corresponde a una situación de contragolpe, donde los jugadores rivales juegan en nivel difícil.

En estos dos primeros escenarios se ve claramente la escala progresiva de dificultad, a continuación se propone otro escenario más largo donde puede no estar a priori tan clara la escala de dificultad.

El tercer conjunto de escenarios de dificultad progresiva definido es el de la figura 4.5, este corresponde con un proceso de aprendizaje completo para jugar a fútbol, es decir, lo que según mi criterio y apoyándome en los resultados de la experimentación básica son los pasos que debe ir aprendiendo una persona para jugar a fútbol hasta poder llegar a jugar un partido de 11 contra 11 jugadores en dificultad difícil que si lo trasladamos a un ejemplo real puede ser jugar a nivel profesional, los diferentes escenarios que forman este conjunto son:

- *academy\_empty\_goal\_close*: este es el escenario inicial, puesto que a mi parecer es el entorno más fácil posible, puesto que el jugador está solo en el área contraria y sin portero, así que solo tiene que tirar.

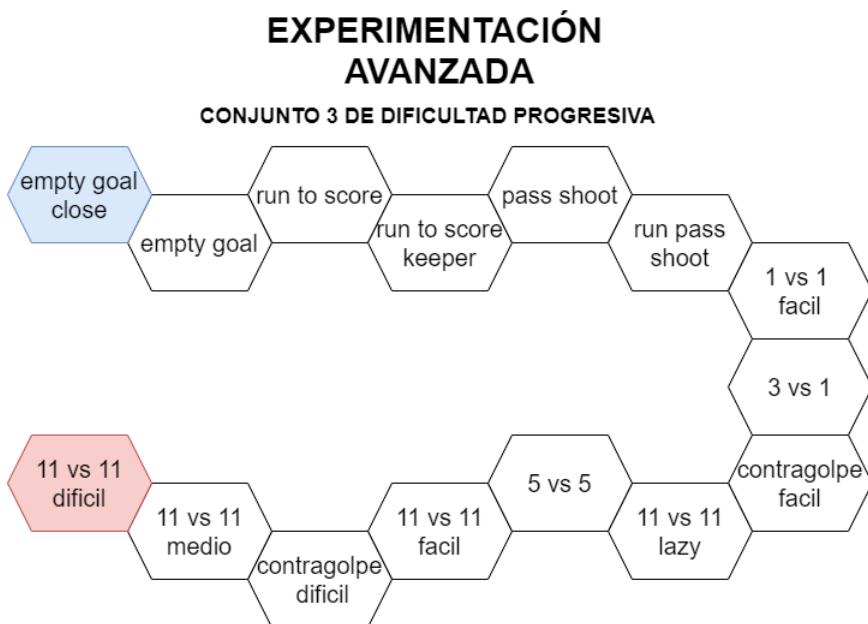


Figura 4.5: Conjunto 3 de escenarios de dificultad progresiva

- *academy\_empty\_goal*: este es el siguiente escenario puesto que es muy parecido con la dificultad añadida que la jugada comienza en el centro del campo en lugar de en el área rival.
- *academy\_run\_to\_score*: este escenario es el primero que es necesario correr, por lo que el agente ahora aprende a hacerlo (si no lo alcanza un defensor rival y le roba el balón) después de haber aprendido anteriormente a tirar a portería.
- *academy\_run\_to\_score\_with\_keeper*: este es el siguiente escenario, es muy parecido al anterior pero con la dificultad añadida de que hay un portero en la portería, por lo que el agente aprende a jugar contra un portero.
- *academy\_pass\_and\_shoot\_with\_keeper*: en este escenario el agente aprende a pasar la pelota, después de aprender tanto a correr como a tirar. Destacar que es necesario pasar para que los rivales no nos arrebaten el balón.
- *academy\_run\_pass\_and\_shoot\_with\_keeper*: en este escenario se mezclan las enseñanzas anteriores, es decir, correr, pasar y tirar, siendo todo necesario para poder obtener una buena puntuación en este escenario.
- *1\_vs\_1\_easy*: ya perfeccionados estos movimientos con el escenario anterior, es la hora de poner en práctica todo contra un rival en un uno

contra uno.

- *academy\_3\_vs\_1\_with\_keeper*: a continuación se incrementan el número de jugadores de tu equipo con respecto al escenario anterior para volver a abrir la posibilidad de pase y se añade la dificultad que hay un portero en la portería rival.
- *academy\_counterattack\_easy*: este escenario es el ya utilizado en conjuntos de escenarios de dificultad progresiva anteriores, se pone en este punto porque después de todo el entrenamiento previo se piensa que el agente ya está preparado para una jugada así.
- *academy\_single\_goal\_versus\_lazy*: este escenario también se ha utilizado anteriormente, es ya un partido de 11 jugadores contra 11 jugadores, después de todo el entrenamiento anterior estimo que el agente está preparado para un partido “real” con los jugadores en modo “perezoso”.
- *5\_vs\_5*: este es un escenario de 5 jugadores contra 5 jugadores, se piensa que el agente a estas alturas del aprendizaje debe actuar bien en este escenario.
- *11\_vs\_11\_easy\_stochastic*: este escenario también se ha utilizado anteriormente, es también un partido de 11 contra 11 pero con un nivel fácil.
- *academy\_counterattack\_hard*: este escenario también se ha utilizado anteriormente, es la situación de contragolpe aunque con un nivel de dificultad alto.
- *11\_vs\_11\_stochastic*: este escenario también se ha utilizado anteriormente, es también un partido de 11 contra 11 pero con un nivel medio.
- *11\_vs\_11\_hard\_stochastic*: este escenario también se ha utilizado anteriormente, es también un partido de 11 contra 11 pero con un nivel alto. Este es el último escenario de este conjunto de escenarios de dificultad progresiva, a mi parecer es el escenario más complicado.

Destacar que se ha dejado fuera la jugada del córner puesto que es una jugada muy concreta que se piensa que puede desajustar el aprendizaje de nuestro agente.

En las figuras anteriores (4.3 4.4 4.5) se marca en color azul el escenario inicial, es decir, desde donde parte el entrenamiento y en color rojo el escenario final, es decir, el último escenario donde se entrena el agente.

Se han planteado diferentes conjuntos de escenarios para poder sacar unas conclusiones sólidas que se apoyen en varios experimentos en lugar de

plantear solo un conjunto de escenarios de dificultad progresiva y opinar en función del mismo, con las conclusiones que se extraigan de estas pruebas se puede alcanzar el objetivo del proceso experimental avanzado. Para ayudar a las conclusiones como se ha comentado anteriormente se entrena un agente en los escenarios finales (los de color rojo en cada uno de los conjuntos de escenarios de dificultad progresiva) de forma independiente para poder compararlo con los agentes entrenados de forma progresiva en estos mismos escenarios.

Para los dos primeros conjuntos de escenarios de dificultad progresiva, es decir, para los escenarios que se representan en las figuras (4.3 4.4) se realizan las pruebas de forma cíclica, como se ha explicado anteriormente. Esto implica que una vez que se ejecute el último escenario del conjunto de escenarios de dificultad progresiva, se vuelve a entrenar el agente realizando todo el recorrido, pero partiendo con lo aprendido hasta en momento.

El código del proceso experimental completo se puede encontrar en el siguiente enlace de *GithHub*: Repositorio de GitHub.



## Capítulo 5

# EXPERIMENTACIÓN Y DISCUSIÓN DE LOS RESULTADOS

En este capítulo se realiza el análisis de los resultados, exponiendo tanto los resultados pertenecientes al proceso experimental básico, como los resultados pertenecientes al proceso experimental avanzado.

### 5.1. EXPOSICIÓN DE RESULTADOS

Para poder analizar los resultados, primero es necesario que estos sean expuestos, por lo tanto la primera parte de este capítulo corresponde a esta exposición de las pruebas correspondientes a la experimentación realizada.

#### 5.1.1. RESULTADOS DE EXPERIMENTACIÓN BÁSICA

En primer lugar, se exponen los resultados extraídos de las pruebas pertenecientes al proceso experimental básico, en términos de recompensa obtenida (“eprewmean”).

Los resultados que obtiene el algoritmo PPO2 se pueden ver en la tabla 5.1 correspondiente a las pruebas realizadas usando *SubprocVecEnv* con 8 entornos, los resultados que obtiene el algoritmo A2C se pueden ver en la tabla 5.2 correspondiente a las pruebas realizadas usando *SubprocVecEnv* con 8 entornos y finalmente los resultados que obtiene el algoritmo DQN con un entorno simple se pueden ver en la tabla 5.3.

	MLP	CNN	IMPALA
11_vs_11_easy_stochastic	-1.25	-0.253	0.163
11_vs_11_hard_stochastic	-2.64	-1.59	-1.63
11_vs_11_stochastic	-2.15	-1.32	-1.21
1_vs_1_easy	-0.352	-0.263	0.071
5_vs_5	-2.35	-2.14	-2.39
academy_3_vs_1_with_keeper	0.83	1.35	1.42
academy_corner	0.701	0.838	0.822
academy_counterattack_easy	0.597	0.885	0.882
academy_counterattack_hard	0.502	0.816	0.782
academy_empty_goal	0.966	1.86	1.92
academy_empty_goal_close	2	2	1.97
academy_pass_and_shoot_with_keeper	0.681	1	0.918
academy_run_pass_and_shoot_with_keeper	0.673	1.45	1.28
academy_run_to_score	0.103	1.96	1.88
academy_run_to_score_with_keeper	0.068	1.44	1.17
academy_single_goal_versus_lazy	0.427	0.997	1.22

Tabla 5.1: “eprewmean” PPO2 utilizando *SubprocVecEnv* con 8 entornos

	MLP	CNN	IMPALA
11_vs_11_easy_stochastic	-1.31	-1.27	-1.45
11_vs_11_hard_stochastic	-2.56	-2.69	-2.6
11_vs_11_stochastic	-2.43	-2.21	-2.19
1_vs_1_easy	-0.453	-0.397	-0.253
5_vs_5	-2.21	-2.21	-2.43
academy_3_vs_1_with_keeper	0.804	0.823	0.819
academy_corner	0.632	0.678	0.663
academy_counterattack_easy	0.392	0.334	0.411
academy_counterattack_hard	0.382	0.37	0.416
academy_empty_goal	0.128	0.115	0.185
academy_empty_goal_close	1.09	1.11	1.89
academy_pass_and_shoot_with_keeper	0.714	0.569	0.701
academy_run_pass_and_shoot_with_keeper	0.692	0.452	0.681
academy_run_to_score	0.107	0.02	0.058
academy_run_to_score_with_keeper	0.012	-0.064	-0.009
academy_single_goal_versus_lazy	0.224	0.203	0.286

Tabla 5.2: “eprewmean” A2C utilizando *SubprocVecEnv* con 8 entornos

## EXPERIMENTACIÓN Y DISCUSIÓN DE LOS RESULTADOS

---

	MLP	CNN	IMPALA
11_vs_11_easy_stochastic	-1.2	-0.9	-0.9
11_vs_11_hard_stochastic	-2.4	-2.8	-2.4
11_vs_11_stochastic	-2	-2.4	-1.6
1_vs_1_easy	-0.4	-0.2	-0.5
5_vs_5	-2.1	-1.5	-1.8
academy_3_vs_1_with_keeper	0.7	0.8	0.9
academy_corner	0.7	0.7	0.7
academy_counterattack_easy	0.5	0.6	0.7
academy_counterattack_hard	0.6	0.6	0.7
academy_empty_goal	0.3	1.5	1.4
academy_empty_goal_close	0.9	2	1.7
academy_pass_and_shoot_with_keeper	0.7	0.8	0.7
academy_run_pass_and_shoot_with_keeper	0.7	0.8	0.8
academy_run_to_score	0	0	0.9
academy_run_to_score_with_keeper	0.1	0.6	0.4
academy_single_goal_versus_lazy	0.6	0.5	0.6

Tabla 5.3: “eprewmean” DQN utilizando 1 entorno simple

A modo resumen se hace una nueva tabla que expone las mejores configuraciones en términos de recompensa obtenida (“eprewmean”), para cada uno de los escenarios del entorno.

MEJORES RESULTADOS	ALGORITMO	RED
11_vs_11_easy_stochastic	PPO2	CNN
11_vs_11_hard_stochastic	PPO2	CNN
11_vs_11_stochastic	PPO2	IMPALA
1_vs_1_easy	PPO2	IMPALA
5_vs_5	DQN	CNN
academy_3_vs_1_with_keeper	PPO2	IMPALA
academy_corner	PPO2	CNN
academy_counterattack_easy	PPO2	CNN
academy_counterattack_hard	PPO2	CNN
academy_empty_goal	PPO2	IMPALA
academy_empty_goal_close	PPO2	CNN
academy_pass_and_shoot_with_keeper	PPO2	CNN
academy_run_pass_and_shoot_with_keeper	PPO2	CNN
academy_run_to_score	PPO2	CNN
academy_run_to_score_with_keeper	PPO2	CNN
academy_single_goal_versus_lazy	PPO2	IMPALA

Tabla 5.4: Mejores resultados de cada combinación algoritmo-red subyacente

### 5.1.2. RESULTADOS DE EXPERIMENTACIÓN AVANZADA

En segundo lugar, se exponen los resultados extraídos de las pruebas pertenecientes al proceso experimental avanzado, se exponen para cada uno de los diferentes conjuntos de escenarios de dificultad progresiva planteados en la sección explicativa del proceso experimental avanzado, en términos de recompensa conseguida (“eprewmean”). Destacar que los escenarios están colocados en las tablas de resultados en orden de ejecución.

En estas pruebas no se varían parámetros ni se pruebas diferentes algoritmos, sino que en esta pruebas se utiliza el algoritmo y parámetros que de forma global han conseguido los mejores resultados durante la experimentación básica, los cuales se analizan durante la próxima sección. Pero como el resultado ha puesto prácticamente al mismo nivel tanto a la red subyacente CNN como a su modificación (IMPALA), utilizando el algoritmo PPO2, se ha decidido lanzar los experimentos de esta sección experimental avanzada utilizando ambas redes subyacentes y PPO2 como algoritmo.

Primero, se exponen los resultados de cada uno de los escenarios para cada uno de los conjuntos de escenarios de dificultad progresiva planteados durante la sección explicativa de la experimentación avanzada:

- Conjunto 1 de escenarios de dificultad progresiva: Partido (definido en la figura 4.3 (tabla 5.5)).
- Conjunto 2 de escenarios de dificultad progresiva: Contragolpe (definido en la figura 4.4 (tabla 5.6)).
- Conjunto 3 de escenarios de dificultad progresiva : Completo (definido en la figura 4.5 (tabla 5.7)).

	CNN	IMPALA
academy_single_goal_versus_lazy	1.45	1.36
11_vs_11_easy_stochastic	2.7	3.53
11_vs_11_stochastic	0.809	1.36
11_vs_11_hard_stochastic	-0.39	-0.179

Tabla 5.5: Resultados de conjunto 1 de escenarios de dificultad progresiva

	CNN	IMPALA
academy_counterattack_easy	0.895	0.799
academy_counterattack_hard	0.957	0.871

Tabla 5.6: Resultados de conjunto 2 de escenarios de dificultad progresiva

## **EXPERIMENTACIÓN Y DISCUSIÓN DE LOS RESULTADOS**

	CNN	IMPALA
academy_empty_goal_close	2	2
academy_empty_goal	1.59	1.72
academy_run_to_score	1.8	1.74
academy_run_to_score_with_keeper	0.105	0.1
academy_pass_and_shoot_with_keeper	0.937	0.693
academy_run_pass_and_shoot_with_keeper	1.06	0.686
1_vs_1_easy	-0.351	-0.049
academy_3_vs_1_with_keeper	0.879	0.742
academy_counterattack_easy	0.305	0.307
academy_single_goal_versus_lazy	0.314	0.186
5_vs_5	-2.12	-1.21
11_vs_11_easy_stochastic	-0.409	0.585
academy_counterattack_hard	0.342	0.302
11_vs_11_stochastic	-1.23	-0.236
11_vs_11_hard_stochastic	0.342	-0.067

Tabla 5.7: Resultados obtenidos en el conjunto de escenarios de dificultad progresiva 3

Segundo, se muestran los resultados de las pruebas cíclicas correspondientes al proceso experimental avanzado. Destacar que se han realizado 3 ciclos (ciclo 0, ciclo 1 y ciclo2), en los conjuntos de escenarios de dificultad progresiva 1 y 2 (definidos en las figuras 4.3 (tabla 5.8) y 4.4 (tabla 5.9)). No se realiza en el conjunto de escenarios de dificultad progresiva 3 porque el tiempo de ejecución hubiera sido demasiado grande.

	ciclo	CNN	IMPALA
academy_single_goal_versus_lazy	0	1.28	1.36
11_vs_11_easy_stochastic	0	2.89	3.53
11_vs_11_stochastic	0	0.721	1.36
11_vs_11_hard_stochastic	0	-0.706	-0.179
academy_single_goal_versus_lazy	1	1.6	1.53
11_vs_11_easy_stochastic	1	2.9	4.99
11_vs_11_stochastic	1	0.641	1.24
11_vs_11_hard_stochastic	1	-0.37	-0.088
academy_single_goal_versus_lazy	2	1.64	1.6
11_vs_11_easy_stochastic	2	2.27	4.68
11_vs_11_stochastic	2	0.389	1.74
11_vs_11_hard_stochastic	2	-0.181	-0.017

Tabla 5.8: Resultados obtenidos en el conjunto de escenarios de dificultad progresiva 1

	ciclo	CNN	IMPALA
academy_counterattack_easy	0	0.89	0.799
academy_counterattack_hard	0	1.18	0.871
academy_counterattack_easy	1	1.15	1.15
academy_counterattack_hard	1	1.19	0.84
academy_counterattack_easy	2	1.2	1.34
academy_counterattack_hard	2	1.22	0.873

Tabla 5.9: Resultados obtenidos en el conjunto de escenarios de dificultad progresiva 2

## 5.2. ANÁLISIS DE RESULTADOS

La segunda parte de este capítulo corresponde al análisis de los resultados expuestos anteriormente, es decir, los resultados correspondientes al proceso experimental realizado. Destacar que de este análisis, se extraen las debidas conclusiones.

Para una mejor explicación del análisis de resultados, se dividen los escenarios del entorno *Google Research Football Environment* en: **escenarios que implican mas de una jugada**, es decir, los escenarios donde el trabajo del agente abarca un partido completo de 90 minutos donde puede haber multitud de jugadas en función del desarrollo del partido y **escenarios que no implican más de una jugada**, como por ejemplo el escenario de contragolpe que tan solo implica una única jugada de contragolpe.

Los escenarios que implican más de una jugada son:

- 11\_vs\_11\_easy\_stochastic
- 11\_vs\_11\_hard\_stochastic
- 11\_vs\_11\_stochastic
- 1\_vs\_1\_easy
- 5\_vs\_5

Los escenarios que *no* implican más de una jugada son:

- academy\_3\_vs\_1\_with\_keeper
- academy\_corner
- academy\_counterattack\_easy
- academy\_counterattack\_hard

## **EXPERIMENTACIÓN Y DISCUSIÓN DE LOS RESULTADOS**

- academy\_empty\_goal
- academy\_empty\_goal\_clos
- academy\_pass\_and\_shoot\_with\_keeper
- academy\_run\_pass\_and\_shoot\_with\_keeper
- academy\_run\_to\_score\_with\_keeper
- academy\_single\_goal\_versus\_lazy

### **5.2.1. ANÁLISIS DE EXPERIMENTACIÓN BÁSICA**

En primer lugar, se analizan tabla a tabla los resultados correspondientes a la experimentación básica del proyecto.

#### **ANÁLISIS DE RESULTADOS DE PPO2**

Utilizando PPO2 como algoritmo de *Aprendizaje por Refuerzo* (tabla 5.1), se consiguen recompensas positivas en la mayoría de los casos, llegando incluso a alcanzar valores cercanos al máximo o incluso el máximo en algunos escenarios del entorno donde este máximo es 2, es decir, en los escenarios que no implican más de una jugada, estos 2 puntos se consiguen sumando 1 del tipo de recompensa checkpoints cuando el agente consigue que el equipo se acerque mucho a la portería y sumando 1 por marcar gol.

Si nos centramos en los escenarios que implican más de una jugada el máximo no está definido, puesto que puede alcanzar tanto como diferencia de goles consiga marcar un equipo durante los 90 minutos que dura un partido, sumando 1 por cada gol anotado y restando 1 por cada gol encajado, así como añadiendo el tipo de recompensa checkpoints al final, en estos escenarios se puede ver que las recompensas son negativas en la mayoría de los casos, lo que implica que el partido se ha perdido, puesto que esa puntuación se consigue al restar por cada gol del equipo rival, tan solo se salvan la dificultad fácil del escenario de 11 contra 11 (1\_vs\_11\_easy\_stochastic), con un tipo de red subyacente en concreto (IMPALA).

Respecto al tipo de red subyacente seleccionada utilizando PPO2 como algoritmo, se puede ver como por regla general los resultados de MLP están por debajo de los conseguidos tanto por CNN como por su variación (IMPALA), siendo los resultados conseguidos por estas dos últimas redes subyacentes bastante parejos en casi todos los casos, de forma evidentemente puesto que una es una modificación de la otra.

### **ANÁLISIS DE RESULTADOS DE A2C**

Utilizando A2C como algoritmo de *Aprendizaje por Refuerzo Profundo* (tabla 5.2), se consiguen recompensas menores a las que se consiguen con el algoritmo PPO2 en todos los casos, llegando incluso a conseguir recompensa negativa (bastante cercana a 0) en algunos escenarios que no implican más de una jugada, contrastando con los resultados positivos que conseguía el algoritmo PPO2 en este tipo de escenarios. Destacando los escenarios que implican más de una jugada, decir que se consiguen recompensas bastante más bajas que con el algoritmo anterior.

Respecto al tipo de red seleccionada utilizando A2C como algoritmo, a diferencia del caso anterior utilizando una red MLP se consiguen resultados más altos que con una CNN o su modificación (IMPALA), en la mayoría de los casos.

### **ANÁLISIS DE RESULTADOS DE DQN**

Utilizando DQN como algoritmo de *Aprendizaje por Refuerzo Profundo* (tabla 5.3), se consiguen unos resultados intermedios entre los conseguidos anteriormente con PPO2 e IMPALA, aunque en el caso en concreto del escenario 5\_vs\_5 consigue los mejores resultados.

### **ANÁLISIS DE LA TABLA RESUMEN DE RESULTADOS**

Si nos centramos en la tabla 5.4, es decir, en la tabla resumen, se puede comprobar a simple golpe de vista que el algoritmo que ha conseguido mejores recompensas en cada uno de los posibles escenarios del entorno, así como la red subyacente que ha utilizado este algoritmo para obtener las mejores recompensas. Esta tabla expresa claramente el dominio del algoritmo PPO2 puesto que obtiene el mejor resultado en todos los escenarios excepto en uno, mientras que se puede ver como en términos de con qué red subyacente funciona mejor, hay bastante igualdad, a pesar de esto, la mejor recompensa se obtiene en más ocasiones con la red CNN en lugar de con su modificación (IMPALA), aunque la diferencia entre estas es mínima como se ha comentado anteriormente durante este análisis.

### **ANÁLISIS DEL PROCESO DE ENTRENAMIENTO**

A continuación, se analiza de forma gráfica algunas de las situaciones observadas durante los experimentos básicos. Las gráficas, muestran en el eje X el número de pasos de entrenamiento (timesteps) y en el eje Y la recompensa obtenida por el agente (“eprewmean”).

## **EXPERIMENTACIÓN Y DISCUSIÓN DE LOS RESULTADOS**

La primera gráfica, se muestra en la figura 5.1, esta corresponde al entrenamiento del agente utilizando el algoritmos PPO2 con la red IMPALA sobre el escenario `academy_counterattack_hard`. Se destaca esta figura para mostrar como en algunos escenarios el crecimiento de “eprewmean” es muy rápido al durante los primeros pasos de entrenamiento y muy lento durante los siguientes.

Por otro lado, existen escenarios como es `academy_3_vs_1_with_keeper`, utilizando PPO2 e IMPALA donde en los primeros paso de entrenamiento no mejora demasiado el “eprewmean” y es a partir de la mitad del total donde empieza a aumentar más rápidamente, esta gráfica se puede ver en figura 5.2

También existen escenarios donde el crecimiento es más o menos constante como en `academy_run_pass_and_shoot_with_keeper`, utilizando PPO y CNN, la gráfica se puede ver en la figura 5.3.

En contraste con las gráficas anteriores donde a pesar de que en ocasiones baja la “eprewmean” se recupera para conseguir mejores resultados, también existen los casos donde el entrenamiento del agente es prácticamente ineficiente, esta gráfica se puede ver en la figura 5.4 correspondiente al entrenamiento en el escenario `1_vs_1_easy`, utilizando A2C y CNN. En estos casos la “eprewmean” está subiendo y bajando continuamente sin lograr una mejoría notable con el trascurso de los pasos de entrenamiento.

Incluso en algunos casos que la tendencia respecto a la “eprewmean” es descendente como en el caso del escenario `11_vs_11_hard_stochastic` utilizando A2C y CNN, que se puede ver en la figura 5.5.

Destacar que hay una gráfica por experimento realizado, para ver la totalidad de estas gráficas correspondientes al proceso experimental básico visitar el repositorio del proyecto: repositorio GitHub.

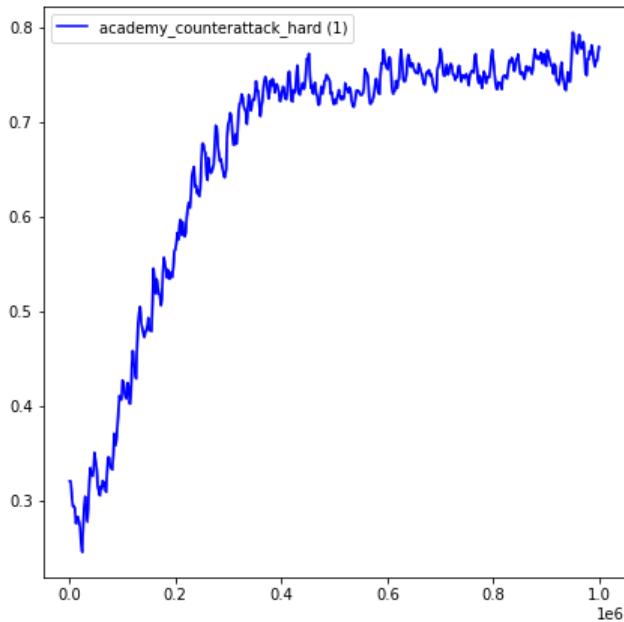


Figura 5.1: Gráfica correspondiente al entrenamiento del agente en academy\_counterattack\_hard utilizando PPO2 e IMPALA

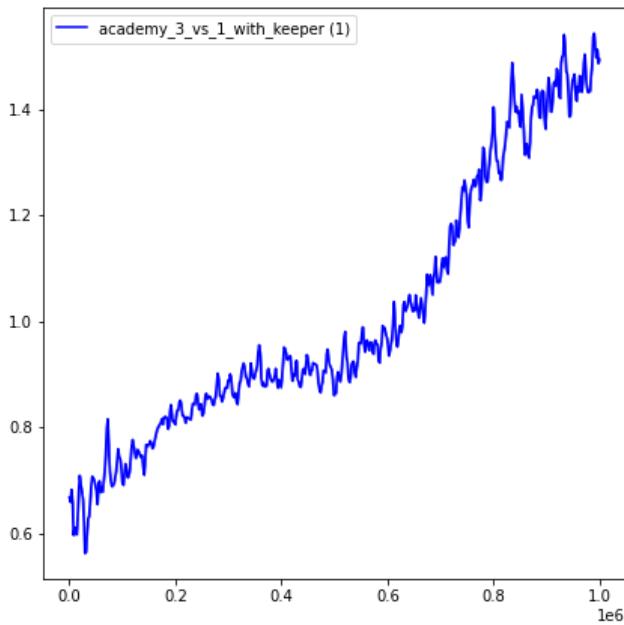


Figura 5.2: Gráfica correspondiente al entrenamiento del agente en academy\_3\_vs\_1\_with\_keeper utilizando PPO2 e IMPALA

## EXPERIMENTACIÓN Y DISCUSIÓN DE LOS RESULTADOS

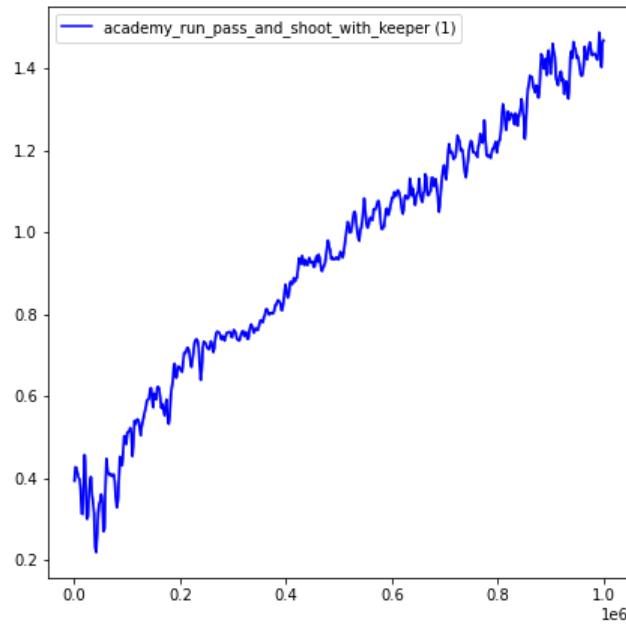


Figura 5.3: Gráfica correspondiente al entrenamiento del agente en academy\_run\_pass\_and\_shoot\_with\_keeper utilizando PPO2 e CNN

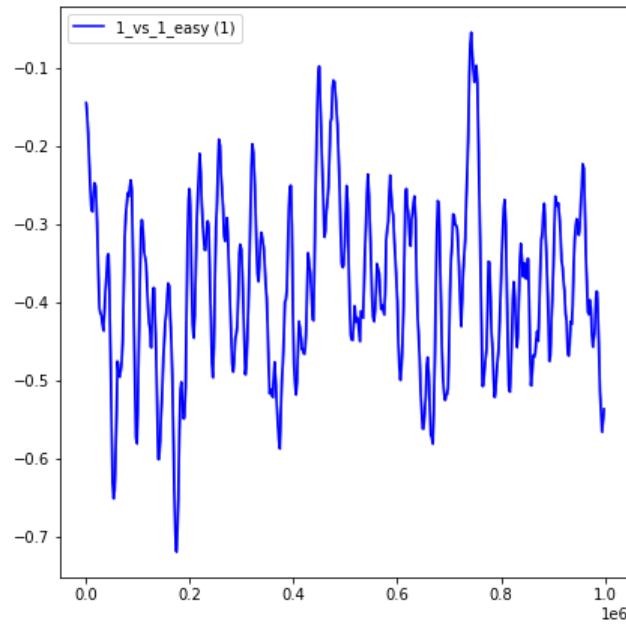


Figura 5.4: Gráfica correspondiente al entrenamiento del agente en 1\_vs\_1\_easy utilizando A2C e CNN

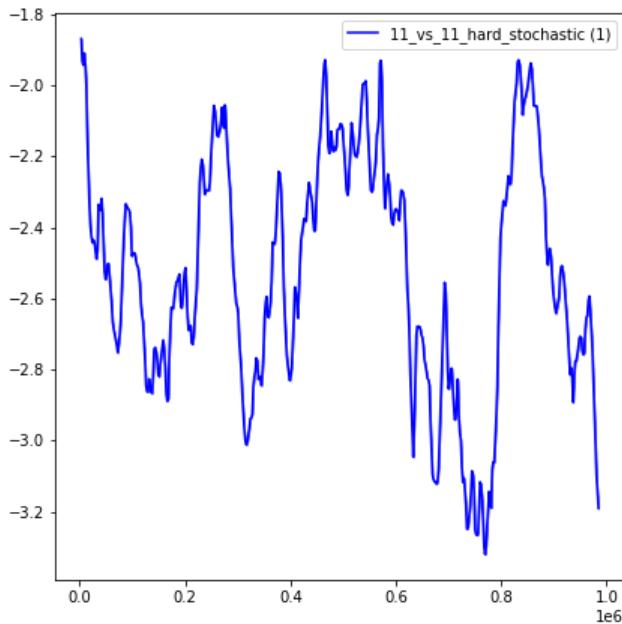


Figura 5.5: Gráfica correspondiente al entrenamiento del agente en 11\_vs\_11\_hard\_stochastic utilizando A2C e CNN

### 5.2.2. ANÁLISIS DE EXPERIMENTACIÓN AVANZADA

En segundo lugar, se analizan tabla a tabla los resultados correspondientes a la experimentación avanzada del proyecto.

Esta parte del proceso experimental se separa en dos subpartes, por lo que primero se analizan los resultados correspondientes al aprendizaje progresivo y posteriormente se analizan los resultados correspondientes al aprendizaje progresivo utilizando ciclos.

#### ANÁLISIS DEL CONJUNTO 1 DE ESCENARIOS DE DIFICULTAD PROGRESIVA

Respecto al agente entrenado en el conjunto 1 de escenarios de dificultad progresiva (partido) (tabla 5.5) se comprueba como tanto en el escenario de 11 contra 11 en dificultad fácil (11\_vs\_11\_easy\_stochastic) como el escenario de 11 contra 11 en dificultad media (11\_vs\_11\_stochastic), utilizando este sistema de aprendizaje progresivo se obtiene una recompensa positiva lejos de la que se obtiene con un aprendizaje simple realizado durante la experimentación básica en esos escenarios, respecto al escenario de 11 contra 11 en dificultad difícil (11\_vs\_11\_hard\_stochastic) a pesar de que mejoran los resultados aún sigue siendo un reto para el agente puesto que la recompensa

## **EXPERIMENTACIÓN Y DISCUSIÓN DE LOS RESULTADOS**

sa oscila el valor 0 por lo que no se consigue ni una buena recompensa ni una mala recompensa, pero si una mejora respecto al aprendizaje simple realizado durante la experimentación básica en ese escenario.

Respecto a las redes subyacentes utilizadas, al igual que ocurre en la experimentación básica los resultados están bastante cercanos tanto utilizando una red subyacente CNN como utilizando su modificación (IMPALA), aunque se puede ver que los resultados conseguidos con IMPALA están por encima que los conseguidos con CNN.

### **ANÁLISIS DEL CONJUNTO 2 DE ESCENARIOS DE DIFICULTAD PROGRESIVA**

Respecto al agente entrenado en el conjunto 2 de escenarios de dificultad progresiva (contragolpe) (tabla 5.6) se comprueba que se mejoran las recompensas conseguidas respecto a la experimentación básica, tanto en el contragolpe de dificultad fácil (academy\_counterattack\_easy) como en el contragolpe de dificultad difícil (academy\_counterattack\_hard). En este caso se consiguen mejores resultados utilizando CNN, aunque ambas redes subyacentes obtienen resultados muy parecidos.

### **ANÁLISIS DEL CONJUNTO 3 DE ESCENARIOS DE DIFICULTAD PROGRESIVA**

Respecto al conjunto 3 de escenarios de dificultad progresiva (completo) (tabla 5.7) se comprueba como en la gran mayoría de los escenarios las recompensas se encuentran por debajo de las conseguidas con una experimentación básica (existiendo excepciones, sobre todo utilizando IMPALA como red subyacente, con la que se consigue mejor recompensa utilizando este tipo de aprendizaje en algunos escenarios), aunque de forma general las recompensas estén por debajo, la diferencia es mínima en casi todos los escenarios. La excepción es que el último escenario, es decir, el escenario de 11 contra 11 en dificultad difícil (11\_vs\_11\_hard\_stochastic) (el escenario más difícil de todos) consigue al fin una recompensa positiva, es decir, consigue la mejor recompensa en comparación con el resto de experimentos realizados en este escenario durante desarrollo del proyecto. Esta recompensa positiva la consigue utilizando como red subyacente CNN.

### **ANÁLISIS DE CICLOS SOBRE CONJUNTO 1 DE ESCENARIOS DE DIFICULTAD PROGRESIVA**

Respecto al primer experimento utilizando ciclos, concretamente usando el conjunto 1 de escenarios de dificultad progresiva (partido) (tabla 5.8) se

comprueba como los resultados de ciclos posteriores no tienen porqué mejorar los resultados de ciclos inferiores, por ejemplo los conseguidos en el último ciclo (ciclo 2) no tienen porqué ser superiores en términos numéricos que los conseguidos en el primer ciclo (ciclo 0). Respecto a las redes subyacentes utilizadas se comprueba que funciona algo mejor en este caso IMPALA.

### **ANÁLISIS DE CICLOS SOBRE CONJUNTO 2 DE ESCENARIOS DE DIFICULTAD PROGRESIVA**

Respecto al segundo experimento utilizando ciclos, concretamente usando el conjunto 2 de escenarios de dificultad progresiva (contragolpe) (tabla 5.9) se comprueba con los resultados, que se verifica lo expuesto en el análisis anterior sobre los resultados conseguidos en ciclos posteriores del aprendizaje progresivo. Respecto a las redes se puede ver que para el escenario del contragolpe fácil (academy\\_counterattack\\_easy) consigue mejor recompensa la red CNN y para el escenario contragolpe difícil (academy\\_counterattack\\_hard) consigue mejor recompensa la red IMPALA.

### **ANÁLISIS DEL PROCESO DE ENTRENAMIENTO**

A continuación, se analiza de forma gráfica algunas de las situaciones de los experimentos avanzados para contrastarlos con las gráficas obtenidas de forma simple durante la experimentación básica. Las gráficas, muestran en el eje X el número de pasos de entrenamiento (timesteps) y en el eje Y la recompensa obtenida por el agente (“eprewmean”).

Para poder apreciar de forma visual la diferencia que existe entre:

- La “eprewmean” del agente en el escenario más complicado, es decir: 11\_vs\_11\_hard\_stochastic, entrenado de forma simple durante la experimentación básica. Cuya gráfica se puede ver en la figura 5.6
- La “eprewmean” del agente en el escenario más complicado, es decir: 11\_vs\_11\_hard\_stochastic, entrenado siguiendo un aprendizaje progresivo en el conjunto 1 de escenarios de dificultad progresiva (partido). Cuya gráfica se puede ver en la figura 5.7
- La “eprewmean” del agente en el escenario más complicado, es decir: 11\_vs\_11\_hard\_stochastic, entrenado siguiendo un aprendizaje progresivo en el conjunto 3 de escenarios de dificultad progresiva (completo). Cuya gráfica se puede ver en la figura 5.8
- La “eprewmean” del agente en el escenario más complicado, es decir: 11\_vs\_11\_hard\_stochastic, entrenado siguiendo un aprendizaje progre-

## **EXPERIMENTACIÓN Y DISCUSIÓN DE LOS RESULTADOS**

sivo con ciclos, en el conjunto 1 de escenarios de dificultad progresiva (partido). Cuya gráfica se puede ver en la figura 5.9

La comparación se hace utilizando las gráficas correspondientes con el algoritmo PPO2 y a la red subyacente CNN, para que esta comparación sea justa. Las recompensas son más altas utilizando el aprendizaje progresivo, destacando que la más alta se obtiene utilizando el aprendizaje progresivo en el conjunto 3 de escenarios de dificultad progresiva (completo).

Destacar que hay una gráfica por experimento realizado, para ver la totalidad de estas gráficas correspondientes al proceso experimental avanzado visitar el repositorio del proyecto: repositorio GitHub.

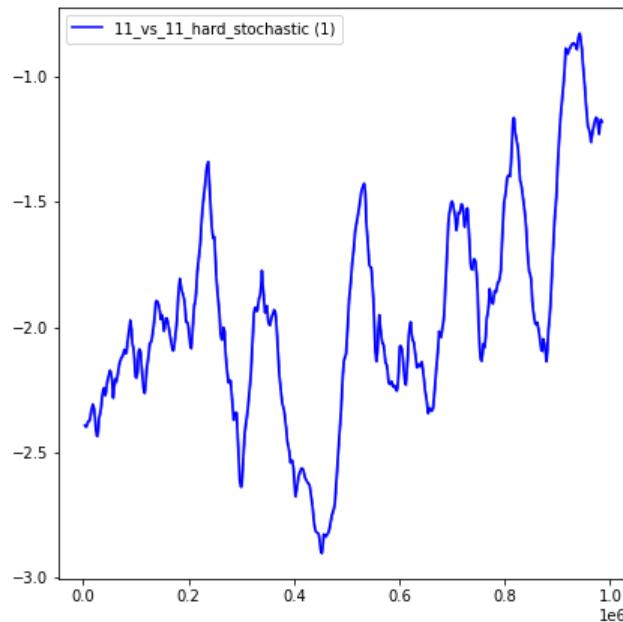


Figura 5.6: Gráfica de comparación correspondiente al entrenamiento de forma simple

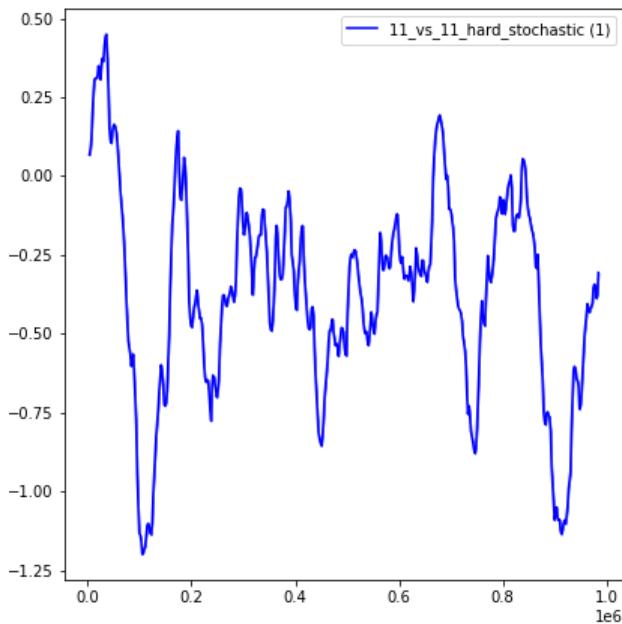


Figura 5.7: Gráfica de comparación correspondiente al entrenamiento utilizando aprendizaje progresivo sobre el conjunto 1

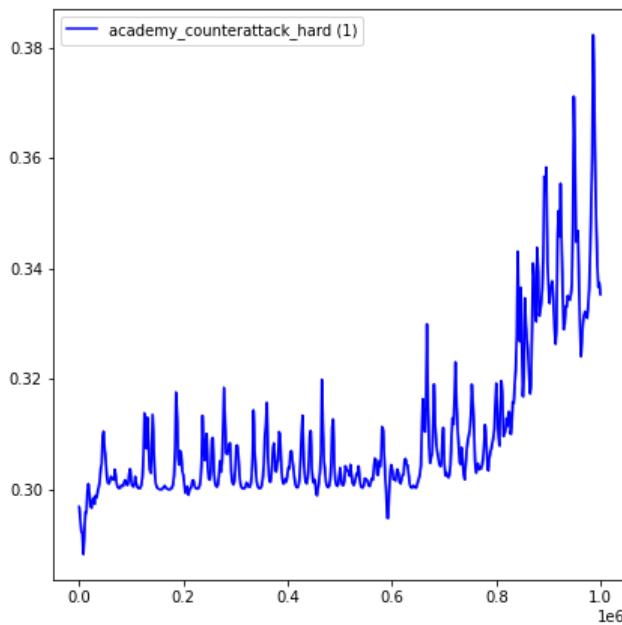


Figura 5.8: Gráfica de comparación correspondiente al entrenamiento utilizando aprendizaje progresivo sobre el conjunto 3

## EXPERIMENTACIÓN Y DISCUSIÓN DE LOS RESULTADOS

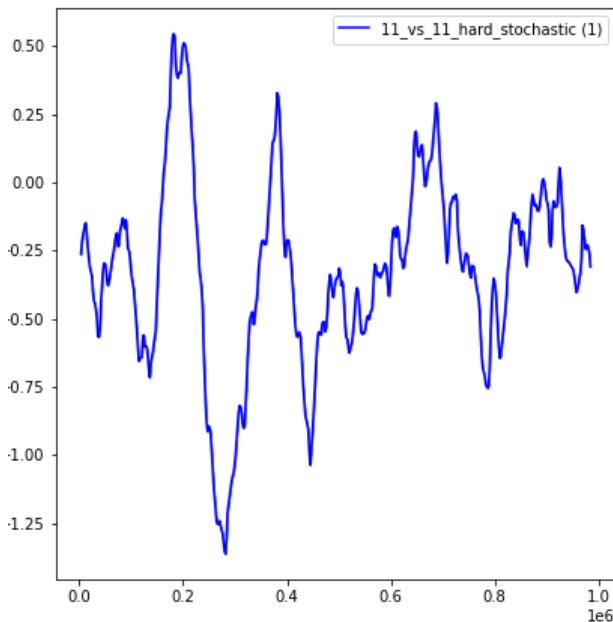


Figura 5.9: Gráfica de comparación correspondiente al entrenamiento utilizando aprendizaje progresivo de forma cíclica sobre el conjunto 1

### 5.2.3. DISCUSIÓN DE RESULTADOS

Las conclusiones que se han extraído del proceso experimental en su totalidad se van a discutir a continuación.

En primer lugar, un resumen de las conclusiones globales del proceso experimental, extraídas durante la “puesta a punto” y verificadas después de la conclusión del proyecto son:

- El uso de la librería *Baselines* ha sido “clave”, puesto que ha funcionado correctamente utilizando el entorno *Google Research Football Environment* durante todo el proceso (a pesar de que algunos algoritmos de *Aprendizaje por Refuerzo Profundo* no se han podido ejecutar sobre el entorno), también ha permitido alternar con facilidad entre los diferentes algoritmos con tan solo cambiar de función y por último ha permitido el uso de algunas funciones interesantes con tan solo modificar un parámetro; como por ejemplo es guardar el progreso del agente cada cierto numero de pasos o aplicar una experiencia inicial al agente.
- El proceso experimental es un proceso “pesado” en términos de tiempo, cada una de las pruebas de este proceso han llegado hasta un total de 1 millón de pasos de entrenamiento (“timesteps”) puesto que el tiempo

consumido por cada una de las pruebas es bastante elevado, razón por la que no se ha podido llegar a realizar más pasos de entrenamiento en algunas pruebas en las que se podría haber considerado interesante. Cabe destacar que el tiempo total que se ha invertido en la ejecución de la totalidad de las pruebas es de 2 meses.

- Los tipos de inputs son mejores o peores dependiendo del escenario del entorno seleccionado, de tal forma que un tipo de input puede ser mejor en un determinado escenario mientras que otro tipo de input puede ser mejor en otro escenario, aunque están bastante parejos. Durante los procesos experimentales básico y avanzado se utiliza “simple115”, es decir, el vector de características del entorno puesto que *Google Colab* no permite la renderización del entorno lo que limita el uso del resto de inputs.
- La variación de la totalidad de parámetros que tiene cada algoritmo supondría bastante más tiempo por esto también ha sido descartada al no ser un objetivo de este *Trabajo Fin de Máster*, aunque si se varían algunos parámetros interesantes como son la red subyacente.
- El uso del tipo de recompensa “scoring” (diferencia de goles) carece de sentido puesto que en algunos escenarios si no se consigue marcar gol en ningún momento no hay ningún tipo de retroalimentación para el agente acerca de como está realizando la acción, por lo que se usa “checkpoints” en todos los casos.
- Conforme más entornos se utilicen dentro de un *SubprocVecEnv* el algoritmo funciona más rápido y de forma general obtiene mejores resultados.

En segundo lugar, las conclusiones extraídas del **proceso experimental básico** son:

- El algoritmo de *Aprendizaje por Refuerzo Profundo* que mejor funciona de forma general en el entorno de *Google Research Football Environment* es PPO2 con bastante diferencia sobre el resto, mientras que el algoritmo que peor funciona es A2C. Respecto al algoritmo DQN no consigue malos resultados aunque realmente no se puede realizar una comparación demasiado realista con respecto al resto de algoritmos, puesto que estos resultados se consiguen con 1 entorno y los anteriores con 8 entornos utilizando un *SubprocVecEnv*. Estos datos no son sorprendentes puesto que PPO2 es uno de los algoritmos que está más optimizado y es más usado actualmente en el campo del *Aprendizaje por Refuerzo*.

## **EXPERIMENTACIÓN Y DISCUSIÓN DE LOS RESULTADOS**

- La red subyacente que mejor funciona en general en el entorno de *Google Research Football Environment* es CNN, así como su modificación (IMPALA), ambas funcionan bastante bien, CNN funciona mejor en algunos escenarios (como 11\_vs\_11\_easy\_stochastic, 11\_vs\_11\_stochastic o academy\_corner) e IMPALA funciona mejor en otros escenarios (como 11\_vs\_11\_hard\_stochastic, 1\_vs\_1\_easy o academy\_3\_vs\_1\_with\_keeper). Respecto a MLP está bastante por debajo de las anteriormente comentadas debido a las capas convolucionales que son capaces de resumir las características del vector de características del entorno, optimizando de esta manera el rendimiento del algoritmo de *Aprendizaje por Refuerzo Profundo*.
- Los escenarios del entorno *Google Research Football Environment* que implican más de una jugada son más complicados para el agente, que los escenarios que no implican más de una jugada, esta diferencia es debida precisamente al matiz de que los escenarios implican más de una jugada y estas jugadas pueden ser bastante diversas; puede haber desde saques de esquinas a penaltis, mientras que en los escenarios que no implican más de una jugada, tan solo desarrollan una jugada concreta hasta su fin, es decir, solo se da una situación concreta del juego, por lo que el agente puede dedicarse explícitamente a aprender como mejorar en esa jugada.
- En los escenarios que disponen de varias dificultades como los escenarios de 11 contra 11 (11\_vs\_11\_easy\_stochastic, 11\_vs\_11\_stochastic y 11\_vs\_11\_hard\_stochastic) o los escenarios de jugada de contragolpe (academy\_counterattack\_easy y academy\_counterattack\_hard), se verifica que el agente funciona mejor en niveles más bajos y funciona peor en niveles más altos, por lo que se puede decir que están bien ajustados los niveles de dificultad por parte de *Google Research Football Environment*.
- El escenario más sencillo es *academy\_empty\_goal\_close*, puesto que en este escenario el agente tan solo tiene que lograr que el jugador realice la acción de tirar para conseguir el gol, puesto que está a una distancia idónea para marcar el gol.
- El escenario más complejo es *11\_vs\_11\_hard\_stochastic*, puesto que este escenario el agente tiene que controlar a todo el equipo durante los 90 minutos que dura un partido y tomar las decisiones adecuadas en cada una de las posibles diferentes situaciones del juego (pudiendo tener una gran cantidad de situaciones diferentes tanto de ataque como de defensa, desde tener que defender un contragolpe hasta tener que intentar marcar un gol en una jugada de córner), a esto se le suma que la dificultad del escenario es difícil.

- Con los resultados de este proceso experimental básico se puede interpretar la dificultad de cada uno de los escenarios viendo las recompensas del agente utilizando diferentes algoritmos y diferentes redes subyacentes, después de 1 millón de pasos de entrenamiento. Por lo que si se ordenan todos los escenarios del entorno en niveles de dificultad, desde el escenario más sencillo hasta el escenario más complejo, se puede formar un recorrido adecuado para que el agente entrene de forma progresiva.

Por último, las conclusiones extraídas del **proceso experimental avanzado** son:

- La escala de dificultad confeccionada para los conjuntos de escenarios de dificultad progresiva que participan en esta experimentación, no solo deben tener en cuenta los resultados de la experimentación básica, puesto que la experiencia de cómo funciona el fútbol y el sentido común son fundamentales para optimizar esta parte del proceso experimental, por lo que no se seguirá al pie de la letra un recorrido de escenarios teniendo en cuenta la dificultad asignada durante el proceso experimental básico aunque si se toma como punto de partida para modificarlo teniendo en cuenta mi experiencia y el sentido común. En el conjunto 1 y 2 de escenarios de dificultad progresiva *se buscan situaciones de entrenamiento más específicas*, en el conjunto 1 concretamente un partido completo de 11 contra 11, utilizando niveles de dificultad sencillos para entrenar al agente antes de afrontar los niveles de dificultad complicados, mientras que en el conjunto 2 se centra concretamente en jugadas de contragolpe, utilizando un nivel de dificultad sencillo para entrenar el agente antes de afrontar el nivel de dificultad difícil. Respecto al conjunto 3 de escenarios de dificultad progresiva, *se basa en un proceso de entrenamiento completo*, es lo que bajo mi punto de vista y basándome en los resultados conseguidos durante la experimentación básica, el recorrido de entrenamiento que debe de seguir un equipo real de fútbol, yendo desde entrenar acciones más básicas como es una jugada donde se tiene que tirar a puerta para marcar gol, hasta jugar un partido a 90 minutos de 11 contra 11 contra un equipo complicado (dificultad difícil).
- El aprendizaje progresivo (“curriculum learning”) funciona muy bien y si lo comparamos con el aprendizaje de forma individual que hacía el agente en cada uno de los escenarios durante la experimentación básica, se puede ver una clara mejoría en todos los sentidos, aunque es especialmente apreciable en los escenarios que implican más de una jugada, puesto que estos eran los que a priori conseguían peores resultados.

## **EXPERIMENTACIÓN Y DISCUSIÓN DE LOS RESULTADOS**

- El comprobar el funcionamiento del aprendizaje progresivo (“curriculum learning”) en el conjunto 3 de escenarios de dificultad progresiva, es otra confirmación del buen funcionamiento del mismo. Este entrenamiento completo hace que se consiga la mejor puntuación de toda la experimentación en lo que al escenario más complicado se refiere, es decir, el partido de 11 contra 11 en dificultad difícil: 11\_vs\_11\_hard\_stochastic, a pesar de que el funcionamiento del agente durante el entrenamiento progresivo en el resto de escenarios no es tan bueno; la explicación de forma general es que el agente va aprendiendo poco a poco las distintas situaciones del juego hasta terminar jugando partidos completos, tanto de 5 contra 5 como finalmente de 11 contra 11.

En los escenarios: 11\_vs\_11\_easy\_stochastic y 11\_vs\_11\_stochastic, el agente no funciona tan bien como funcionaba en el conjunto 1 de escenarios de dificultad progresiva, de forma evidentemente puesto que el agente en ese caso se entrenaba concretamente para la situación de partidos de 11 contra 11. Cuando se trata del escenario de dificultad difícil: 11\_vs\_11\_hard\_stochastic, obtiene el mejor resultado del proceso experimental puesto que el agente después del proceso completo de aprendizaje progresivo ya está preparado para afrontar todas las situaciones del juego de forma bastante positiva, siendo capaz de resolver cualquier situación que se plantee en este escenario de la máxima dificultad. Como hipótesis se puede intuir que el conjunto 1 y 2 de escenarios que entranan situaciones más específicas pueden caer en sobreentrenamiento y perder su capacidad de generalización, por ejemplo en el conjunto 2 de escenarios de dificultad progresiva el agente una vez ha concluido su entrenamiento, hace que las situaciones de contragolpe las realice muy bien pero en el resto de acciones sea bastante malo.

- Con la parte del aprendizaje progresivo utilizando ciclos se demuestra que los resultados obtenidos en los ciclos anteriores no tienen por qué mejorar los resultados obtenidos en los ciclos posteriores. Cuando un agente se vuelve a entrenar en escenarios en los que ya ha sido entrenado anteriormente, de forma que vuelve a recorrer todos escenarios del ciclo, este agente puede bajar su rendimiento en algunos escenarios aunque esto también puede significar que pueda conseguir una mejoría en otro escenario más difícil del conjunto de escenarios de dificultad progresiva, esto se debe a que al entrenar el agente en escenarios más difíciles, por ejemplo 11\_vs\_11\_hard\_stochastic, puede hacer que se alcancen situaciones más complejas que no se alcanza en escenarios más sencillos como por ejemplo 11\_vs\_11\_stochastic, entonces cuando el agente vuelve al escenario más sencillo en el siguiente ciclo, este agente tiene en cuenta estas situaciones, por lo tanto aunque ya no sea

extremadamente bueno en las acciones aprendidas en el escenario más sencillo, el agente gana capacidad de generalización, demostrando la falsedad de la teoría enunciada anteriormente durante las conclusiones del proceso experimental básico, que decía: “el conjunto 1 y 2 de escenarios de dificultad progresiva pueden caer en sobrentrenamiento”. En definitiva esta forma de aplicar el “curriculum learning” es una opción interesante si se desea conseguir una buena recompensa en un conjunto de escenarios de dificultad progresiva.

- Es tan importante el plantear un conjunto de escenarios de dificultad progresiva equilibrado y diverso, como utilizar el entrenamiento cílico, por lo que se intuye que el experimento más óptimo hubiera sido utilizar el conjunto 3 de escenarios de dificultad progresiva (completo) con el aprendizaje por ciclos para conseguir los mejores resultados en el escenario más complicado (11\_vs\_11\_hard\_stochastic), esto conseguiría un agente muy robusto preparado para todo tipo de situaciones y con una alta capacidad de generalización, pero esto ha sido imposible por términos de tiempo invertido en realizar un ciclo en este conjunto de escenarios de dificultad progresiva

# Capítulo 6

## CONCLUSIONES Y FUTURAS MEJORAS

### 6.1. CONCLUSIONES GENERALES

En primer lugar, se puede destacar que se han adquirido conocimientos respecto a la creación de una experimentación software partiendo desde cero. Como consecuencia de lo mencionado anteriormente se han aprendido y afianzado conceptos sobre ingeniería del software.

Se ha ampliado el conocimiento, en lo que a lenguajes de programación se refiere, puesto que la herramienta se ha desarrollado en *Python*, de este modo he afianzado y ampliado los conocimientos básicos que la titulación de graduado en ingeniería informática me aportó. Extrapolando la anterior afirmación puedo destacar que he conseguido aprender y reforzar no solo conceptos referentes a *Python* sino conceptos referentes a múltiples áreas de la informática, como por ejemplo: *Aprendizaje por Refuerzo* o *Redes Neuronales Artificiales* entre otras.

La realización de este trabajo creo que ha sido muy positiva para mí, por la forma en que se ha enfocado, es decir hacia el mundo laboral, de tal forma que pienso que me ayudará en mi futuro, así como pienso que ha sido una experiencia satisfactoria para mí, donde he aprendido, me he esforzado y he disfrutado con el trabajo bien hecho.

Como conclusión final y después de evaluar todos los planteamientos que se hicieron al comienzo del proyecto, los cuales se pueden comprobar en el primer capítulo del proyecto, se puede afirmar que todos los objetivos han sido cumplidos, aunque también se piensa que siempre es posible una mejora, dedicado a aquellas personas que quieran realizar estas futuras mejoras, se pueden inspirar en las posibles mejoras que les sugiero yo en la próxima

sección.

También se ha de comentar que tanto el director de este proyecto: D. Juan Gómez Romero, como su autor: Ángel Murcia Díaz, estamos muy satisfechos con el trabajo realizado, y esperamos que este proyecto sea de gran ayuda para que se puedan realizar múltiples proyectos de investigación en el futuro centrados en las TIC.

## **6.2. FUTURAS MEJORAS**

En esta sección se exponen una serie de proposiciones o ideas futuras para el lector o investigador que esté interesado en continuar investigando el entorno *Google Research Football Environment* utilizando algoritmos de *Aprendizaje por Refuerzo*, utilizando este *Trabajo Fin de Máster* como punto de partida para este cometido.

Las futuras mejoras propuestas son:

- Realizar una experimentación variando todos los parámetros de los algoritmos para poder comprobar como influyen cada uno de estos y sacar conclusiones al respecto.
- Hacer un algoritmo genético donde cada individuo de la población sea un algoritmo de *Aprendizaje por Refuerzo*, para que después de un número considerable de épocas y mucha potencia de procesamiento y tiempo después, pueda conseguir un resultado muy optimizado. Destacar que esta idea fue propuesta en un “brainstormy” al inicio del proyecto pero que no ha resultado por falta de tiempo.
- Realizar un estudio de la interpretabilidad de los algoritmos utilizados, ya que utilizan *Redes Neuronales* y estas es sabido que no tienen una alta interpretabilidad, para de esta manera poder “razonar” el comportamiento de los agentes en cada uno de los casos después del entrenamiento de los mismos.
- Intentar extender este proyecto a un estudio comparativo de librerías, es decir, utilizar las librerías de *Aprendizaje por Refuerzo* mencionadas anteriormente y ejecutar con todas ellas el mismo algoritmo para poder compararlas en términos de tiempo y resultado.
- Entrenar los agentes dependiendo la posición que ocupen en el campo (delantero, centrocampista o defensa), para que a la hora de la verdad solo actúe el agente que corresponda.
- Realizar un programa con interfaz gráfica incluida donde poder configurar las pruebas deseadas y que este programa utilice *OpenAI Baseli-*

*nes* para realizar las pruebas previamente configuradas por el usuario sobre el entorno *Google Research Football Environment*.



## Apéndice A

# ANEXO DE INFORMÁTICA

En este anexo se explica la “puesta a punto”, es decir, a preparar el entorno e instalar las librerías necesarias para poder ejecutar el entorno *Google Research Football Environment*.

Como este entorno se ha ejecutado tanto de forma local como de forma remota usando *Google Colab*, se explica posteriormente para ambos casos.

### A.1. De forma local

En primer lugar, se explica de forma local, destacar que como requisito es necesario que el sistema operativo pueda descargar python 3.7 y trabajar con él (es muy importante la versión, ya que con otras versiones pueden surgir algunos problemas), aunque se recomienda encarecidamente utilizar Ubuntu para ejecutar el entorno de forma local.

En caso de estar utilizando Ubuntu, en primer lugar, es necesario instalar las siguientes librerías a través de terminal:

```
1 sudo apt-get install git cmake build-essential libgl1-mesa-dev libsdl2-dev libsdl2-image-dev libsdl2-ttf-dev libsdl2-gfx-dev libboost-all-dev libdirectfb-dev libst-dev mesa-utils xvfb x11vnc libsdl-sge-dev python3-pip
```

En caso de estar utilizando Mac, en primer lugar, es necesario ejecutar los siguientes comandos a través de terminal:

```
1 brew install git python3 cmake sdl2 sdl2_image sdl2_ttf sdl2_gfx boost boost-python3
```

```
1 brew install sdl sdl_image sdl_mixer sdl_ttf portmidi
```

A continuación se descarga el entorno en si mismo utilizando los siguientes comentados:

```
1 git clone https://github.com/google-research/football.git
2 cd football
```

En este punto es encarecidamente recomendable utilizar un entorno virtual, para esto:

```
1 python3 -m venv football-env
2 source football-env/bin/activate
```

En caso de que no este instalado venv es necesario instalarlo utilizando este comando:

```
1 pip3 install virtualenv
```

A continuación se instala el entorno descargado anteriormente:

```
1 pip3 install .
```

Posteriormente, se ejecutan los siguientes comandos en terminal para instalar las librerías necesarias para que se puedan ejecutar los algoritmos de aprendizaje automático incluidos en la librería Baselines.

```
1 python3 -m pip install --upgrade pip setuptools
```

Si el ordenador donde se va a instalar va a utilizar la CPU, se utiliza el siguiente comando:

```
1 pip3 install tensorflow==1.15.* or pip3
```

Si el ordenador donde se va a instalar tiene la capacidad de utilizar la GPU se utiliza el siguiente comando:

```
1 pip3 install tensorflow-gpu==1.15.* ,
```

```
1 pip3 install dm-sonnet==1.*;
```

Por último, se instala la librería Baselines:

```
1 pip3 install git+https://github.com/openai/baselines.git@master
```

## A.2. De forma remota: Google Colab

En segundo lugar, se explica de forma remota utilizando Google Colab para la causa, en este caso tan solo es suficiente con ejecutar las siguientes celdas de código en un cuaderno:

```
1 !python3 -m pip install --upgrade pip setuptools
```

```
1 !pip3 install tensorflow-gpu==1.15.*
```

```
1 !pip3 install dm-sonnet==1.*
```

```
1 !pip3 install git+https://github.com/openai/baselines.git@master
```

```
1 !apt-get update
2 !apt-get install libsdl2-gfx-dev libsdl2-ttf-dev
3
4 !git clone -b v2.1 https://github.com/google-research/football.git
5 !mkdir -p football/third_party/gfootball_engine/lib
6
7 !wget https://storage.googleapis.com/gfootball/
     prebuilt_gameplayfootball_v2.1.so -O football/third_party/
     gfootball_engine/lib/prebuilt_gameplayfootball.so
8 !cd football && GFOOTBALL_USE_PREBUILT_SO=1 pip3 install .
```

El proceso es similar al anterior pero adaptado a la ejecución de forma remota en *Google Colab*. La inclusión del signo “!” es debido a que como originariamente las celdas de código son para ejecutar código en python, si se quiere ejecutar comandos de BASH este signo tiene que preceder el comando.

Destacar que en este caso da exactamente igual el sistema operativo que se utilice puesto que se ejecuta de forma remota, siendo tan solo es necesario que se abra un cuaderno de *Google Colab* través del navegador.



## Apéndice B

# ANEXO DE VÍDEOS

En este anexo se explican los vídeos realizados para el *Trabajo Fin de Máster*, estos vídeos tienen el objetivo de que el lector pueda comprender de forma gráfica todo lo explicado en este proyecto, los vídeos están subidos a *YouTube* por lo que todo el mundo puede acceder a verlos.

Los vídeos realizados son los siguientes:

- Link del primer vídeo: en este primer vídeo, tan solo se comparan de los diferentes escenarios del entorno.
- Link del segundo vídeo: en este segundo vídeo, se puede comprobar cómo funciona un agente durante su proceso de entrenamiento, concretamente en el entorno del partido de 11 contra 11 de dificultad difícil (11\_vs\_11\_hard\_stochastic), para comprobar que lo comentado durante el proyecto (en términos numéricos de recompensa) se verifica. En el vídeo, se ve un agente sin entrenar, un agente en la mitad de su entrenamiento y un agente al final de su entrenamiento, para comparar su comportamiento y eficacia.
- Link del tercer vídeo: en este tercer vídeo, se ve a un agente jugar en el escenario de 11 contra 11 de dificultad difícil (11\_vs\_11\_hard\_stochastic), este agente es previamente es entrenado en el conjunto 3 de escenarios de dificultad progresiva, con el objetivo de comprobar la diferencia entre un entrenamiento normal y un entrenamiento aplicando el “Curriculum Learning”, es decir, el aprendizaje en niveles progresivos de dificultad.

Destacar que algunas partes de los vídeos se avanzan en cámara rápida para conseguir ver en menos tiempo el desarrollo del partido o de la jugada en caso de que el escenario no sea el de un partido completo.



# Bibliografía

- [1] “10 Real-Life Examples of Machine Learning”. En: (2019). URL: <https://www.futureinsights.com/10-real-life-examples-of-machine-learning/>. [En Línea. Última consulta: 17-09-2020].
- [2] “9 Applications of Machine Learning from Day-to-Day Life”. En: (2017). URL: <https://medium.com/app-affairs/9-applications-of-machine-learning-from-day-to-day-life-112a47a429d0>. [En Línea. Última consulta: 9-09-2020].
- [3] G. Acuña. “¿Qué son las Redes Neuronales Artificiales que pretenden emular la inteligencia humana?” En: (2020). URL: <https://www.latercera.com/que-pasa/noticia/que-son-las-redes-neuronales-artificiales/993209/>. [En Línea. Última consulta: 19-08-2020].
- [4] “Agentes inteligentes y la naturaleza de su entorno”. En: (2009). URL: <https://poiritem.wordpress.com/2009/11/16/6-4-2-agentes-inteligentes-y-la-naturaleza-de-su-entorno/>. [En Línea. Última consulta: 1-09-2020].
- [5] I. Alvarado. “Las redes neuronales: qué son y por qué están volviendo”. En: (s.f.). URL: [https://ml4a.github.io/ml4a/es/neural\\_networks/](https://ml4a.github.io/ml4a/es/neural_networks/). [En Línea. Última consulta: 25-09-2020].
- [6] J. Barrios. “ADAM: a method for stochastic optimization”. En: (s.f.). URL: <https://www.juanbarrios.com/redes-neurales-convolucionales/>. [En Línea. Última consulta: 28-09-2020].
- [7] D. Calvo. “Función de activación – Redes neuronales”. En: (s.f.). URL: <https://www.diegocalvo.es/funcion-de-activacion-redes-neuronales/>. [En Línea. Última consulta: 25-09-2020].
- [8] “Discusion Kaggle: Google Research Football with Manchester City F.C.” En: (2020). URL: <https://www.kaggle.com/c/google-football/discussion/187657>. [En Línea. Última consulta: 8-09-2020].

- [9] F.Sancho. “Todo lo que Necesitas Saber sobre el Descenso del Gradiante Aplicado a Redes Neuronales”. En: (2019). URL: <https://medium.com/metadatos/todo-lo-que-necesitas-saber-sobre-el-descenso-del-gradiante-aplicado-a-redes-neuronales-19bdbb706a78>. [En Línea. Última consulta: 25-09-2020].
- [10] J. Fan y col. “A Theoretical Analysis of Deep Q-Learning”. En: (2020). URL: <https://arxiv.org/abs/1901.00137>. [En Línea. Última consulta: 28-09-2020].
- [11] J. Fernandez. “Reinforcement Learning”. En: (s.f.). URL: [http://www.tsc.uc3m.es/~jesusfbes/MLG\\_RL.pdf](http://www.tsc.uc3m.es/~jesusfbes/MLG_RL.pdf). [En Línea. Última consulta: 18-09-2020].
- [12] “Función de activación – Redes neuronales”. En: (s.f.). URL: <http://halweb.uc3m.es/esp/Personal/personas/jmmarin/esp/DM/tema3dm.pdf>. [En Línea. Última consulta: 26-09-2020].
- [13] I. Gavilán. “Catálogo de componentes de redes neuronales (III): funciones de pérdida”. En: (2020). URL: <https://www.diegocalvo.es/funcion-de-activacion-redes-neuronales/>. [En Línea. Última consulta: 25-09-2020].
- [14] J. Gómez. “Github: jgromero - eci2019-DRL”. En: (2020). URL: <https://github.com/jgromero/eci2019-DRL>. [En Línea. Última consulta: 5-11-2020].
- [15] “Google Colab: Jupyter Lab con esteroides”. En: (2017). URL: <https://sitiobigdata.com/2019/11/22/google-colab-jupyter-lab-con-esteroides/>. [En Línea. Última consulta: 14-11-2020].
- [16] “Google football environment — installation and Training RL agent using A3C”. En: (2019). URL: <https://towardsdatascience.com/google-football-environment-installation-and-training-rl-agent-using-a3c-d058a44f0fad>. [En Línea. Última consulta: 6-09-2020].
- [17] “IAT: Inteligencia artificial: qué es, tipos, técnicas, ventajas”. En: (2020). URL: <https://iat.es/tecnologias/inteligencia-artificial/>. [En Línea. Última consulta: 18-09-2020].
- [18] D. Kingma y J. Lei. “ADAM: a method for stochastic optimization”. En: (2017). URL: <https://arxiv.org/pdf/1412.6980.pdf>. [En Línea. Última consulta: 29-09-2020].
- [19] K. Kurach, R. y O. Bachem. “Introducing Google Research Football: A Novel Reinforcement Learning Environment”. En: (2019). URL: <https://ai.googleblog.com/2019/06/introducing-google-research-football.html>. [En Línea. Última consulta: 18-11-2020].

- [20] K. Kurach y col. “Google Research Football: A Novel Reinforcement Learning Environment”. En: (2020). URL: <https://arxiv.org/pdf/1907.11180.pdf>. [En Línea. Última consulta: 1-12-2020].
- [21] J. Lopez. “Aplicación de Aprendizaje por Refuerzo Adversario en Juegos de Atari”. En: (2019). URL: [https://e-archivo.uc3m.es/bitstream/handle/10016/30007/TFG\\_Jesus\\_Lopez\\_Baeza-Rojano\\_2019.pdf?sequence=1&isAllowed=y](https://e-archivo.uc3m.es/bitstream/handle/10016/30007/TFG_Jesus_Lopez_Baeza-Rojano_2019.pdf?sequence=1&isAllowed=y). [En Línea. Última consulta: 16-09-2020].
- [22] E. Martín. “Por qué machine learning sera la tecnología más importante en 2018”. En: (2017). URL: [https://elpais.com/tecnologia/2017/11/28/actualidad/1511866764\\_933798.html](https://elpais.com/tecnologia/2017/11/28/actualidad/1511866764_933798.html). [En Línea. Última consulta: 16-08-2020].
- [23] V. Mnih y col. “Asynchronous Methods for Deep Reinforcement Learning”. En: (2016). URL: <https://arxiv.org/abs/1602.01783>. [En Línea. Última consulta: 5-10-2020].
- [24] M. Morales. “Deep Reinforcement Learning. Editado por Manning.” En: (2018). [En Línea. Última consulta: 22-09-2020].
- [25] “OpenAI Baselines: DA2C y ACKTR”. En: (s.f.). URL: <https://openai.com/>. [En Línea. Última consulta: 28-09-2020].
- [26] P. Roy. “13 Examples of Machine Learning Applications in Real World”. En: (2019). URL: <https://learning.naukri.com/articles/13-examples-of-machine-learning-applications-in-real-world/>. [En Línea. Última consulta: 17-09-2020].
- [27] G. Pérez. “¿Cómo funcionan las redes neuronales convolucionales?” En: (2018). URL: <https://es.quora.com/C%C3%B3mo-funcionan-las-redes-neuronales-convolucionales>. [En Línea. Última consulta: 28-09-2020].
- [28] G. Pérez. “Introducción al deep learning parte 2: Redes Neuronales Convolucionales”. En: (2019). URL: <https://mc.ai/introduccion-al-deep-learning-parte-2-redes-neuronales-convolucionales/>. [En Línea. Última consulta: 24-09-2020].
- [29] R. Portelas y col. “Automatic Curriculum Learning For Deep RL: A Short Survey”. En: (2020). URL: <https://arxiv.org/pdf/2003.04664.pdf>. [En Línea. Última consulta: 16-11-2020].
- [30] “Reinforcement learning con Mario Bros – Parte 1”. En: (s.f.). [En Línea. Última consulta: 18-09-2020].
- [31] “Revista de Robots”. En: (2020). URL: <https://revistaderobots.com/>. [En Línea. Última consulta: 15-08-2020].

- [32] J. Rodriguez. “How Google uses Reinforcement Learning to Train AI Agents in the Most Popular Sport in the World”. En: (2019). URL: <https://www.kdnuggets.com/2019/06/google-reinforcement-learning-ai-agents-sport.html>. [En Línea. Última consulta: 1-09-2020].
- [33] S. Russel y P.Norvig. “INTELIGENCIA ARTIFICIAL UN ENFOQUE MODERNO Segunda edición ”. En: (2004). URL: <https://luismejias21.files.wordpress.com/2017/09/inteligencia-artificial-un-enfoque-moderno-stuart-j-russell.pdf>. [En Línea. Última consulta: 24-09-2020].
- [34] “Sanlam: Introducción a la Inteligencia Artificial”. En: (2018). URL: <https://www.sanlamgis.com/Documents/SGIP%20-%20Introduccion%20a%20la%20Inteligencia%20Artificial%20y%20el%20Aprendizaje%20de%20M%C3%A1quina.pdf>. [En Línea. Última consulta: 14-08-2020].
- [35] “SAS: Inteligencia artificial, qué es y porqué es importante”. En: (s.f.). URL: [https://www.sas.com/es\\_es/insights/analytics/what-is-artificial-intelligence.html](https://www.sas.com/es_es/insights/analytics/what-is-artificial-intelligence.html). [En Línea. Última consulta: 18-09-2020].
- [36] J. Schulman y col. “Proximal Policy Optimization Algorithms”. En: (2017). URL: <https://arxiv.org/pdf/1707.06347.pdf>. [En Línea. Última consulta: 5-10-2020].
- [37] R.S. Sutton y col. “Reinforcement Learning. Editado por MIT”. En: (2018). [En Línea. Última consulta: 22-09-2020].
- [38] U. Tewari. “Shamcode: google-research / football”. En: (s.f.). URL: <http://news.shamcode.ru/blog/google-research--football/>. [En Línea. Última consulta: 1-09-2020].
- [39] O. Vega. “Una introducción a los procesos de decisión markovianos con costo promedio”. En: (2018). URL: [https://misclaneamatematica.org/welcome/default/download/tbl\\_articulos.pdf2.a9d7b23c8fcf943b.363630342e706466.pdf](https://misclaneamatematica.org/welcome/default/download/tbl_articulos.pdf2.a9d7b23c8fcf943b.363630342e706466.pdf). [En Línea. Última consulta: 29-09-2020].
- [40] M. Vitelli y A. Nayebi. “CARMA: A Deep Reinforcement Learning Approach to Autonomous Driving”. En: (s.f.). URL: [https://web.stanford.edu/~anayebi/projects/CS\\_239\\_Final\\_Project\\_Writeup.pdf](https://web.stanford.edu/~anayebi/projects/CS_239_Final_Project_Writeup.pdf). [En Línea. Última consulta: 25-09-2020].
- [41] A. Zai y B.Brown. “Deep Reinforcement Learning in Action.” En: (2018). [En Línea. Última consulta: 23-09-2020].

