UNIVERSIDAD DEL PAÍS VASCO

TESIS FÍN DE MÁSTER

Knowledge Transfer in Deep Reinforcement Learning

Autor:
Rubén MULERO

Supervisores:
Dr. Aitor ALMEIDA
Dr. Basilio SIERRA

Tesis presentada en cumplimentación de los requisitos
del Máster en Ingeniería Computacional y Sistemas Inteligentes

en el

Grupo de Robótica y Sistemas Autónomos
Departamento de Ciencias de la Computación e Inteligencia Artificial

con la colaboración de

Fundación Deusto (DeustoTech)

10 de diciembre de 2018

«His delight at seeing this creation of human hands was mixed with the bitterness of finally understanding that nothing like it ever would be created again.»

Dmitry Glukhovsky

UNIVERSIDAD DEL PAÍS VASCO

Resumen

Escuela de Máster y Doctorado
Departamento de Ciencias de la Computación e Inteligencia Artificial

Máster en Ingeniería Computacional y Sistemas Inteligentes

Knowledge Transfer in Deep Reinforcement Learning

por Rubén MULERO

El auge del aprendizaje automático como método para generar una Inteligencia Artificial (IA), está generando un campo de investigación en el que se están poniendo en práctica varios conceptos ya formulados en los años 40 y 60 y, que antaño, eran imposibles de realizar debido a las implicaciones tecnológicas que eran necesarias. Hoy en día, se dispone de un nivel de potencia en hardware que permite poner en práctica los postulados que intentaban dar vida *inteligente* a una máquina. El interés y la motivación de crear esa vida se está convirtiendo en un motor clave en el desarrollo de una sociedad tecnológica más avanzada. La potencia alcanzada por los procesadores gráficos (GPU) ha hecho viable crear redes neuronales complejas que permiten simular una inteligencia viva, tan viva como lo pudiera ser una persona. Esta inteligencia, es capaz de decidir una serie de acciones en un contexto determinado y *aprender* a mejorar para ser más eficiente y adaptable a los cambios.

Dentro de las distintas aproximaciones desarrolladas, podemos encontrar dos de ellas que hoy en día, están siendo muy relevantes y ambiciosas: 1) el llamado *reinforcement learning*[1] que busca generar una máquina inteligente a través de un sistema de recompensas que otorga ciertos puntos a partir de las acciones que realiza en un entorno; 2) el llamado *deep learning*[2] que genera un sistema de redes neuronales profundas para entrenar modelos que son capaces de estudiar cada pixel de una pantalla para saber qué puede existir en una región determinada. Haciendo uso de estos dos conceptos, aparece una nueva aproximación llamada *deep reinforcement learning*[3] que busca juntar las bondades de las redes neuronales profundas con un sistema de recompensa que haga que una IA sea capaz de tomar las mejores decisiones posibles en un entorno determinado.

En esta Tesis Fin de Máster se ha llevado a cabo un estudio por el cual, se ha experimentado si las redes neuronales profundas son capaces o no de transferir el conocimiento adquirido mediante un entrenamiento previo. Para demostrar si éste hecho es o no factible, se ha creado un agente inteligente capaz de jugar a un video-juego usando únicamente como entrada de datos los píxeles de una pantalla. Con esta premisa, se ha puesto a prueba la experiencia adquirida por el agente en otro juego completamente distinto para observar cuál es su nivel de adaptabilidad, y si la experiencia previa adquirida, juega un papel fundamental a la hora de aprender a jugar a un video-juego distinto.

[1] https://www.cs.ubc.ca/ murphyk/Bayes/pomdp.html
[2] http://deeplearning.net/
[3] https://deepmind.com/blog/deep-reinforcement-learning/

Agradecimientos

La realización de esta tesis ha sido posible gracias al trabajo y la guía de varias personas. En primer lugar, agradecerle al Dr. Aitor Almeida el haber confiado en mi para proponerme un tema de experimentación tan interesante y haberme guiado en el proceso de realización de la tesis. En segundo lugar, agradecerle al Dr. Basilio Sierra su gran trabajo y paciencia para poder conseguir que esta tesis sea atractiva, amena e interesante. En tercer lugar, agradecerle a Mikel Villamañe su apoyo para realizar el máster y depositar su confianza en mi a la hora de recomendarme.

Índice general

Índice de figuras

Índice de tablas

Lista de Abreviaturas

AI	**A**rtificial **I**ntelligence
ML	**M**achine **L**earning
MDP	**M**arkov **D**ecision **P**rocess
RL	**R**einforcement **L**earning
MC	**M**onte **C**arlo
DL	**D**eep **L**earning
TL	**T**ransfer **L**earning
DRL	**D**eep **R**einforcement **L**earning
DQN	**D**eep **Q**-Learning **N**etwork
ANN	**A**rtificial **N**euronal **N**etworks
CNN	**C**onvolutional **N**euronal **N**etworks
TD	**T**ime **D**ifference
MC	**M**onte **C**arlo
PE	**P**olicy **E**valuation
PI	**P**olicy **I**nterval
AC	**A**ctor **C**ritic
SGD	**S**tochastic **G**radient **D**escent

Lista de Símbolos

T	tensor	$\mathbb{R}^2 \otimes \mathbb{R}^*$
V_π(s)	función de valor	$E \bullet \{\sum_{i=0}^{\infty} \gamma^i r_{t+i+1} \vert s_t = s, \pi\}$
V^*	función de valor óptima	
V_{π^*}	política óptima función de valor	
$Q_\pi(s,a)$	función de acción-valor	$E \bullet \{\sum_{i=0}^{\infty} \gamma^i r_{t+i+1} \vert s_t = s, a_t = a, \pi\}$
$Q^*(s,a)$	función de acción-valor óptima	
$Q_{\pi^*}(s,a)$	política óptima función de a-v	
$L(\theta)$	función *loss* de la red neuronal	$E \bullet \{(r + \gamma \, \text{máx}_{a'} \, Q(s', a' \vert \theta') - Q(s, a \vert \theta))^2\}$
t	timestep	
x	imagen transformada	
s	estado	
a	acción	
r	recompensa	
π	política en los estados	
π^*	política óptima en todos los estados	
θ	pesos de la red neuronal	
φ	imagen original completa	
γ	factor de descuento	

A mi madre, por inculcarme que siempre debes escoger aquel camino que desemboque en aquello que más guste hacer. A mi pareja, por enseñarme que no todo en la vida es blanco o negro. A mi hermana, por enseñarme que las cosas se pueden ver de muchas formas distintas y que no todo es lo que parece. A mi hermano, por ser quien me ha dado la oportunidad de poder estudiar.

Capítulo 1

Introducción

1.1. Motivación

El ser humano tiene la capacidad de aprender de forma continuada a través de una serie de experiencias. Dicho cúmulo de experiencias es la base que determina que sea capaz de tomar las mejores decisiones posibles en cada instante temporal de su vida. Estas decisiones son las que se entienden como *decisiones inteligentes* desde el punto de vista humano. La ciencia ha hecho sus mejores esfuerzos por reproducir dicho aprendizaje y comportamiento inteligente de forma artificial, intentando crear un ser que pudiera ser tan *inteligente* como un humano lo pudiera ser. Y, aunque se han hecho bastantes avances al respecto, aún queda mucho camino por recorrer para que llegue el día en el que se cree un ser artificial que, al menos, pueda competir o mejorar la toma de decisiones que puede realizar un ser humano en un determinado contexto.

La creación de los primeros ordenadores allá en 1823 [Babbage y Baily, 1823] cuando no eran más que meras calculadoras para tabular funciones polinomiales, abrieron un nuevo campo en el que una máquina era capaz de llevar acabo acciones que podía realizar un ser humano de una forma más rápida y eficaz. Ello abrió una nueva era en la que ya se vaticinó[1] que una máquina podría ser capaz de "pensar"de una forma más rápida y eficiente de lo que lo haría un ser humano. El problema radicaba en que, las soluciones existentes, no eran capaces de ser autónomas, no podían dar soluciones óptimas si el ser humano no realizaba al menos algún tipo de influencia sobre la máquina (meter los datos necesarios para realizar una suma, por ejemplo). Por lo tanto, las máquinas no podían ser inteligentes ya que todas ellas prescindían al menos de dos importantes capacidades: 1) la capacidad de aprender de forma autónoma; 2) la capacidad de tomar decisiones inteligentes en base a lo aprendido que maximicen un resultado final.

El progreso en los distintos campos de investigación, como la psicología, la medicina, la neurociencia, la economía, las matemáticas y la ingeniería, ha creado un caldo de cultivo que ha propiciado los pilares fundamentales para la aparición de lo que hoy en día, se llama la *Inteligencia Artificial*.

1.2. La Inteligencia Artificial

En el año 1965 Jonh McCarthy acuño la expresión de *Inteligencia Artificial* (IA) como: *"...la ciencia e ingenio de hacer máquinas inteligentes, especialmente programas de cómputo inteligentes."*[2]. Eso generó, lo que hoy en día llamamos como IA, la creación de programas que permitan de forma autónoma aprender a tomar decisiones que

[1] https://es.wikipedia.org/wiki/Isaac_Asimov
[2] https://es.wikipedia.org/wiki/John_McCarthy

maximicen el valor esperado de un resultado concreto. La IA se ha convertido en un ambicioso campo por el que crear máquinas autómatas para poder realizar acciones cotidianas de manera automática. Por ejemplo, conducción autónoma, coloreado automático de fotografías en blanco y negro, traducción automática, respuestas inteligentes a ciertas preguntas, simulaciones eficientes en entornos complicados, detección de elementos en una imagen, reconocimiento de patrones, planificación automática, controles de sistemas, juego automático etc.. El campo de aplicación es bastante grande y la especialización puede darse en muchas áreas distintas. La IA se clasifica en varias ramas, cada una de ellas tiene una aproximación distinta aunque en esta tesis nos vamos a centrar en la rama de la *Inteligencia Artificial Computacional*

1.2.1. IA Computacional

La Inteligencia Artificial Computacional [Fulcher, 2008] consiste en la inteligencia que se desarrolla mediante un aprendizaje iterativo. En cada iteración, la IA es capaz de obtener los resultados de las acciones que ha realizado y modificar sus valores internos (pesos, calculo de probabilidades....) para poder perfilar y mejorar los resultados obtenidos. Este tipo de inteligencia necesita del uso de datos empíricos ya que son la base de su conocimiento para ir *aprendiendo* de las experiencias pasadas e intentar mejorar las decisiones futuras. Algunos ejemplos de la IA Computacional son: 1) Las redes neuronales artificiales; 2) los sistemas difusos; 3) la computación evolutiva. Esta aproximación es bastante más moderna que otras soluciones que hacen uso de sistemas expertos (acciones que se realizan cuando se dispara una regla determinada), redes bayesianas (estados condicionados a probabilidades) o razonamientos basados en casos (dado un caso muy determinado se toman una o varias acciones concretas y cerradas para resolverlo).

1.3. El aprendizaje

Para que una IA pueda convertirse en una entidad inteligente necesita de un proceso de aprendizaje automático, por el cual sea capaz de aprender a tomar decisiones inteligentes en cada caso. Para entender el proceso de aprendizaje de una IA primero se tiene que aprender el proceso de aprendizaje humano. La psicología enumera distintas metodologías para que un ser humano sea capaz de aprender (aprendizaje colaborativo, aprendizaje basado en competencias, etc...) de todas ellas la más destacable, por su simplicidad y su eficiencia es la llamada *aprendizaje por refuerzo* [Kaelbling, Littman y Moore, 1996].

El aprendizaje por refuerzo consiste en una metodología de aprendizaje por la cual una entidad (*agente*) es capaz de aprender, dado un estado o situación (*estado* ó s) cuán de buena ha sido la acción realizada en el entorno que le rodea (*acción* ó a) mediante un programa de recompensa (*recompensa* ó r) en un paso o instante temporal determinado (*timestep* ó t). El agente, mediante un proceso continuado de acciones, busca maximizar la recompensa final obtenida, siendo el objetivo final obtener la máxima recompensa posible (r_{max}). La figura 1.1 muestra un resumen de lo que es el aprendizaje por refuerzo.

Dependiendo del contexto en el que se desarrolle el aprendizaje por refuerzo, un *agente* puede ser considerado: una persona, un animal, un programa informático, etc.... Una *acción* generalmente suele ser considerada una cantidad que puede ser medida cualitativamente o cuantitativamente, como izquierda o derecha, 0 y 1 o *darle al botón azul* o *darle al botón rojo*. Un *estado* puede ser varias cosas y depende mucho

del entorno. El estado se puede dividir en dos tipos distintos en función de la información que reciba el agente: 1) estados de baja dimensionalidad o *low-dimensionality*; 2) estados de alta dimensionalidad o *high-dimensionality*. El primero consiste en que la información del estado recibida por el agente es de poca dimensión, como por ejemplo la velocidad del viento, la altura con respecto al eje de rotación de la tierra...Son medidas simples que el agente recibe combinadas para crear un estado en un paso temporal determinado. El segundo consiste en que la información recibida por el agente es de una dimensión bastante grande. Un ejemplo sería una serie de imágenes; el agente recibe una pila de imágenes que le muestran una transición de un proceso determinado. Cada imagen tiene una resolución que se representa mediante una matriz con un tamaño $AxLxC$ donde A es el ancho de la imagen, L es el largo de la imagen, C es el canal de la imagen. Por lo tanto, una serie determinada de imágenes apiladas formaría el estado s del agente en un paso temporal determinado t.

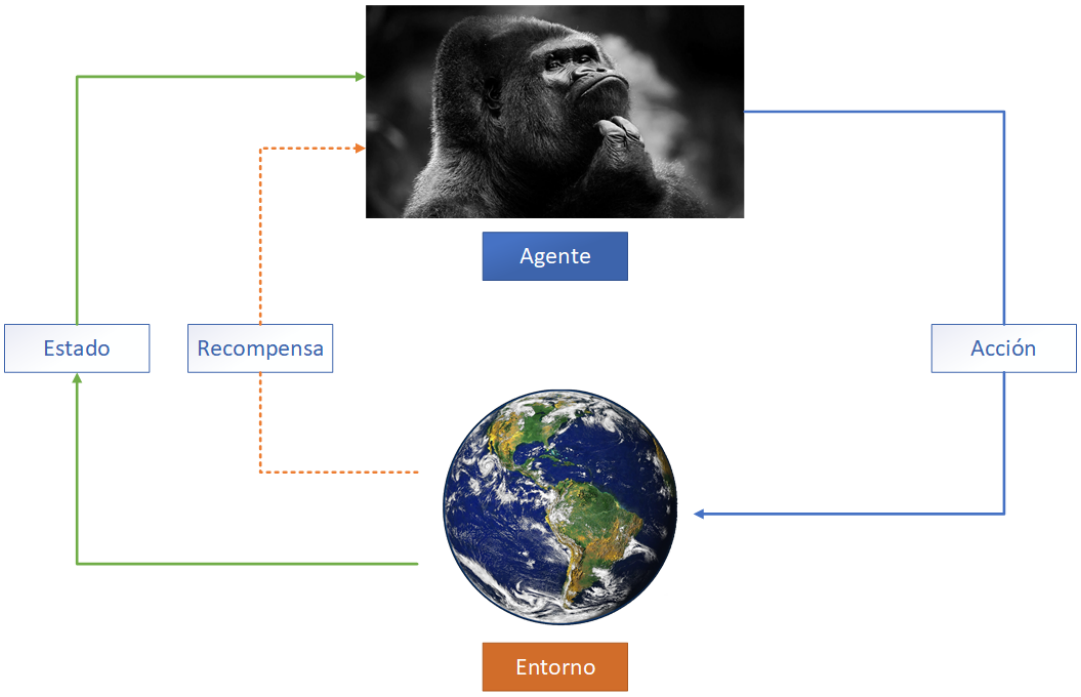

FIGURA 1.1: Diagrama conceptual del aprendizaje por refuerzo

Dentro del mundo de la computación, el aprendizaje por refuerzo o *Reinforcement Learning* (RL) toma como base los procesos de decisión de Markov o *Markov Decission Process* (MDP) postulados por Richard Bellman [Bellman, 1957]. Los MDP (Figura 1.2)[3], consisten en un proceso de control temporal estocástico discreto. En cada paso temporal t, el proceso se encuentra en algún estado s_t, y el agente tiene que decidir cualquier acción a_t que esté disponible en el estado s_t en el que se encuentre. El proceso responde en el siguiente paso temporal, moviéndose de forma aleatoria a un nuevo estado s_{t+1} y dándole al agente una recompensa r_t calculada mediante una función $R(s_t, s_{t+1})$. La probabilidad de que un agente pase de un estado s_t a un estado s_{t+1} se encuentra mapeada en una matriz de probabilidades de estados $M[P(s_t, s_{t+1})]$ donde se identifica, por cada estado, las probabilidades

[3]Imagen de waldoalvarez bajo la licencia CC BY-SA 4.0

de terminar en el siguiente estado. La acción tomada a_t en el estado s_t condiciona de forma directa el nuevo estado s_{t+1}. Las acciones a_t tomadas en los estados s_t son independientes a todas las acciones y estados anteriores por lo que un MDP satisface la propiedad de Markov: *el futuro es independiente al pasado dado el presente*: $P[S_{t+1}|S_t] = P[S_{t+1}|S_1, ..., S_{t+1}]$. Además, las acciones tomadas pueden ser influenciadas por una política de decisiones π que es una distribución sobre acciones dados los estados: $\pi(a|s) = P[A_t = a|S_t = s]$. Dicha política determina completamente el comportamiento del agente y depende en gran medida del estado en el que se encuentre. Resumidamente, una política π es una matriz que tiene mapeadas las probabilidades de tomar una acción a_t en cada estado s_t. Por lo tanto, la idea general de aprendizaje por refuerzo consiste en realizar una serie de iteraciones o *episodios* en los cuales se obtenga una retroalimentación que permita al agente tomar las mejores acciones a_t en cada estado s_t para obtener la mayor recompensa posible r_{max}.

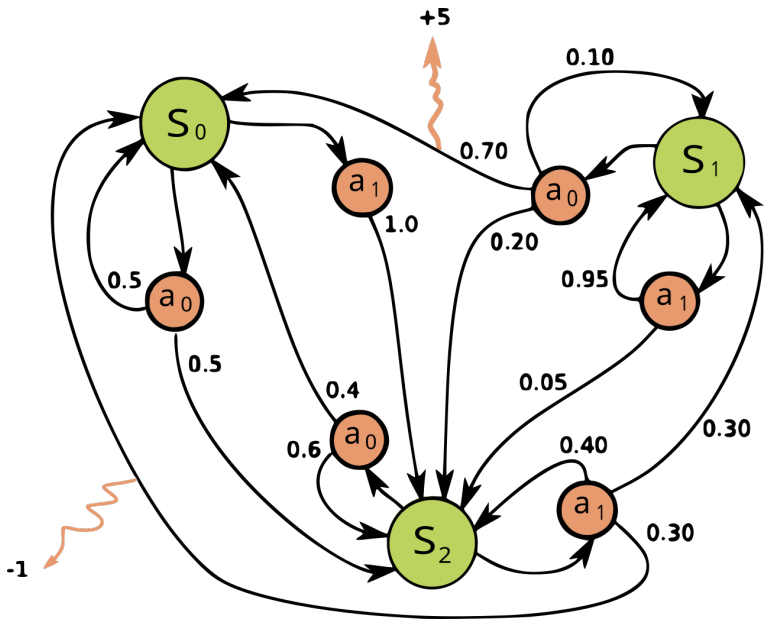

FIGURA 1.2: Diagrama de un MDP

1.3.1. Resolución MDP

Los MDP pueden ser resueltos matemáticamente de dos manera distintas: 1) mediante programación dinámica [Dixit, 1990]; 2) mediante programación lineal [Schrijver, 1998]. A la hora de diseñar un algoritmo que maximice la recompensa r_{max}, se tiene que tener en cuenta el contexto por el cual se quiere entrenar un agente inteligente para aplicar una de las dos estrategias de optimización matemática. Dicho contexto está directamente relacionado con el entorno por el cual el agente tiene que interactuar. Si el entorno es completamente conocido (conocemos los estados, las probabilidades de transición entre ellos etc..) entonces se podrá adoptar una estrategia de optimización mediante programación dinámica, ya que buscaremos realizar una planificación dentro del MDP. En cambio, si todos o parte de los estados del entorno son desconocidos o todos y parte de las probabilidades de cambio entre los estados son desconocidos, entonces se tendrá que adoptar una estrategia de optimización mediante programación lineal.

Para la primera estrategia, que es la programación dinámica o *Dynamic Programming*, se intenta optimizar un algoritmo que resuelva un MDP por el cual se tiene un completo conocimiento de sus estados, transiciones, recompensas etc. En éste tipo de optimización se busca crear una política π que se vaya optimizando de manera iterativa mediante una evaluación continua de los valores que se van obteniendo en cada estado, y para ello se pueden adoptar dos modelos estratégicos: 1) una estrategia predictiva por la cual dado un MDP y una política π determinada se busca conseguir una función de valor V_π ajustada a esa política dictada; 2) una estrategia de control por la cual dado un MDP determinado se busca conseguir una función de valor óptima V^* y una política óptima π^*. Para conseguir estos dos modelos se hace uso de las ecuaciones llamadas *Bellman Expectation Equation* y *Bellman Optimality Equation* [Bellman, 2013].

Para la segunda estrategia, que es la llamada programación lineal o *Lineal Programming*, se busca optimizar un algoritmo que resuelva un MDP del cual no se tiene un completo conocimiento de sus estados, transiciones, recompensas, etc. El MDP en este caso es desconocido. Los algoritmos aplicados se hacen llamar precisamente *model free* porque no se precisa de ese conocimiento completo. En este tipo de optimización se busca crear una política π basada en las experiencias obtenidas en distintas partidas completas o *Episodios*. La política se irá ajustando en una frecuencia determinada hasta converger a la política óptima π^*. Dos ejemplos muy usados son los métodos de *Monte Carlo* (MC) o *Temporal-Difference* (TD). El método de MC calcula las políticas de decisión de una agente por medio de la media acumulada de la experiencia en cada episodio finalizado que realiza el agente. El método de TD calcula la políticas de decisión del agente por cada cambio de estado que realiza. La diferencia fundamental entre estos dos algoritmos para resolver un MDP *model free* es que MC tiene una varianza muy alta pero el sesgo es inexistente y en cambio TD tiene una varianza muy baja y algo de sesgo. La Figura 1.3 muestra un ejemplo por el cual una persona sale de su oficina hacia su casa y quiere saber el tiempo total que consume para completar ese *Episodio*. Como se puede apreciar, con el método de MC el tiempo total se calcula una vez se ha terminado el *Episodio* completo, en cambio, con el método TD el tiempo se va calculando por cada paso temporal t realizado.

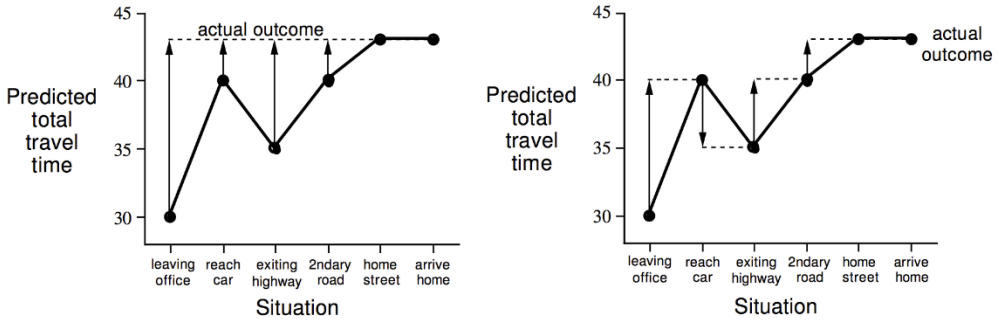

FIGURA 1.3: Métodos *model free*; (a) izquierda: Monte-Carlo; (b) derecha: Temporal-Difference, imagen por David Silver en sus cursos online

1.3.2. Exploración o Explotación

El aprendizaje por refuerzo tiene una problemática compleja, que se transforma en un problema de decisión debido a la inexistencia de una forma eficaz de paliarla. Este problema es el llamado *exploration or exploitation problem* o matemáticamente formalizado como el problema de *multi-armed bandit*[4]. El problema viene a decir que no se sabe nunca a ciencia cierta cuándo un agente debería intentar explorar nuevas soluciones para intentar maximizar la recompensa obtenida o cuando debería intentar centrarse en perfeccionar las decisiones en un área concreta. Esto choca bastante porque sucede en la vida real muy a menudo; por ejemplo si una persona invierte en bolsa nunca se sabe cuándo es el mejor momento para vender acciones, igual se puede tomar una decisión directa y rápida y vender para conseguir una recompensa inmediata que sería una suma de dinero, o esperar un tiempo y perder dinero en aras a encontrar una venta que le haga obtener una recompensa mucho más grande que se traduciría en muchísimo más dinero que la venta inmediata. La elección no es algo baladí, y es uno de los problemas a los que el investigador se tiene que enfrentar cuando debe diseñar una IA para resolver un problema en un contexto determinado.

Para intentar paliar esta situación, en los problemas del aprendizaje por refuerzo se suele aplicar un ruido para perturbar las probabilidades de decisión del agente y hacer que tome decisiones que en otras condiciones no tomaría. Dichas perturbaciones suelen ser procesos aleatorizados, que van menguando en cada iteración hasta desaparecer y dejar al agente tomar sus decisiones directamente. El objetivo es intentar explorar el espacio de estados mientras el agente acumula *experiencia* que puede utilizar más adelante como una forma de tomar mejores decisiones. Un ejemplo común y básico es el epsilon voraz o $\epsilon - greedy$[5], que consiste en una estrategia voraz basada en las pautas del enfriamiento estadístico para tomar acciones discretas (izquierda o derecha) mediante una comparación con un pseudo-generador aleatorio que determina una acción final. Otra estrategia, es el proceso de Ornstein-Uhlenbeck [Bibbona, Panfilo y Tavella, 2008] utilizado en entornos donde las acciones a tomar son espacios continuos (girar 30º a la derecha, o 56º a la izquierda). Este proceso añade una pequeña cantidad de ruido siguiendo los parámetros de la fórmula. Dicho ruido contiene una aleatorización (no es siempre la misma cantidad) ya que se apoya en valores anteriormente obtenidos y en la obtención de valores aleatorios siguiendo una distribución normal; además, éste método responde a un valor ϵ que determina el punto final de adición de ruido cuando se quiere dejar de explorar.

1.4. Deep Reinforcement Learning

Como se ha mencionado en la sección 1.3, el aprendizaje por refuerzo consiste en una sucesión de estados y acciones que buscan maximizar una recompensa final. Dependiendo del contexto, los estados y las acciones varían bastante; no es lo mismo un helicóptero, cuyo estado podría ser la información que dictamina dónde se encuentra en el aire, la velocidad del viento que le golpea, etc.. que un coche cuyo estado podría ser la información de la velocidad en carretera, la orientación respecto a la zona central de la carretera o la entrada de una imagen captada por una cámara delantera. Los estados contribuyen a dar información sobre dónde se encuentra el

[4] https://en.wikipedia.org/wiki/Multi-armed_bandit

[5] https://imaddabbura.github.io/blog/data%20science/2018/03/31/epsilon-Greedy-Algorithm.html

agente y dan sentido a las acciones tomadas por éste, ya que dichas acciones podrían ser más o menos complejas o simplemente podrían ser acciones en un espacio de dimensiones continuo o discreto.

Un ejemplo sencillo de entender y que es mostrado de forma muy extendida en la literatura son los video-juegos. Un video-juego, consiste en un juego virtual que se desarrolla mediante la interacción con una entrada visual (una pantalla) en la que un *agente* se mueve libremente en un *entorno* determinado mediante las *acciones* que manda un jugador. Por lo tanto, la principal vía de entrada son las acciones del mando y el estado es una sucesión de imágenes que dotan de un sentido a aquello que se está viendo (una serie de imágenes que generan una transición que da sentido al juego). Un ordenador como tal, no es capaz de entender las imágenes que se le envían, pero si es capaz de entender una matriz de grandes proporciones en las que en cada celda se guarda los colores que forman un pixel de la pantalla. En éste campo, es donde entra el concepto de las redes neuronales profundas o *Deep Learning* (DL), basadas en las *redes convolucionales* [Szegedy y col., 2016].

El DL ha sido todo un hito en el campo del aprendizaje automático. Permite a una máquina aprender a distinguir distintos elementos de una imagen mediante el uso de redes neuronales artificiales. Las neuronas toman como entrada los píxeles de una imagen (alto y ancho) así como la paleta *Red Blue Green* (RGB) para poder identificar elementos en una imagen. Gracias a una serie de patrones que dictaminan cómo tienen que ir conectadas una redes neuronales con otras, se hace posible que puedan distinguir elementos en una imagen, y asociarlos a algún tipo de etiqueta o *recompensa*.

Por tanto, de la combinación del DL y del RL nace el Deep Reinforcement Learning (DRL), que consiste en el proceso por el cual una máquina es capaz de aprender a tomar una serie de acciones que maximicen una función de recompensa de manera automática, tomando como entrada únicamente los píxeles de una pantalla y asociándolos a una serie de recompensas. La aplicación del DRL es muy ambiciosa porque permite ser utilizado en entornos reales, por ejemplo, se puede entrenar una IA en un simulador de coches, para luego crear un sistema de conducción automática en coches reales que tome únicamente como entrada las imágenes de una cámara situada en la parte delantera del mismo y permita realizar ciertas acciones que salven la vida a un conductor. La aplicación de DRL es muy grande y puede ayudar a solventar todo tipo de problemas en distintas ramas. El mayor inconveniente de este campo de la investigación, es que actualmente no está altamente explotado y muy pocos investigadores han conseguido hacer grandes progresos. Quizás hoy en día, se puede hacer una mención especial a la empresa DeepMind[6] propietaria de Google, la cual año tras año sorprende al mundo con distintos avances y contribuciones.

1.5. Objetivos de la Tesis

El objetivo de esta tesis fin de máster, consiste en realizar un estudio por el cual se demuestre la siguiente hipótesis nula H_0: *"El conocimiento adquirido por un agente en un juego determinado puede ser reutilizado en otros juegos para entrenar agentes inteligentes de una manera más rápida y que obtengan mejores resultados en menos tiempo"*. Por lo tanto, no se busca generar el algoritmo más eficiente para enseñar a una IA a jugar a un juego, ya que el actual estado del arte contempla varias aproximaciones por las cuales se han conseguido distintos algoritmos que maximizan las recompensas obtenidas por el agente en distintos entornos. Lo que se busca es demostrar si una

[6] https://deepmind.com/

vez que una IA ha sido entrenada satisfactoriamente, su conocimiento puede ser reutilizado en otros campos y que dicho conocimiento la ayude a aprender de una manera más veloz.

Para desarrollar la hipótesis nula planteada se han definido los siguientes subobjetivos de la hipótesis formal

- **SO1**: analizar si las redes neuronales convolucionales entrenadas en un video-juego determinado pueden generalizarse a otros video-juegos similares.
- **SO2**: analizar si las redes neuronales densas entrenadas en un video-juego determinado pueden generalizarse a otros video-juegos similares.
- **SO3**: analizar si las redes neuronales convolucionales entrenadas en un video-juego determinado pueden generalizarse a otros video-juegos que no son similares.
- **SO4**: analizar si las redes neuronales densas entrenadas en un video-juego determinado pueden generalizarse a otros video-juegos que no son similares.

Capítulo 2

Estado del arte

2.1. Introducción

En éste capítulo se va realizar un repaso del estado del arte para saber en qué punto se encuentra la investigación que se quiere llevar a cabo durante el proceso de esta tesis. Este capítulo está dividido en las siguientes secciones: 1) Reinforcement Learning: explica el estado del arte acerca del RL clásico, desde sus inicios psicológicos y matemáticos hasta su modelación en los ordenadores; 2) Convolutional Neuronal Networks: esta sección explica el nacimiento de las neuronas artificiales, las redes neuronales artificiales y qué uso se les ha dado dentro del mundo de la informática para ser capaces de reconocer imágenes y clasificarlas de forma exitosa; 3) Deep Reinforcement Learning: aquí se explica el nacimiento y uso del DRL, la motivación y el uso que se consigue gracias a éste hito; 4) Transfer Learning: se expone esta nueva línea de investigación que busca crear modelos capaces de ser generalistas para ser aprovechados en la resolución de distintos problemas, reutilizando la experiencia obtenida en otros contextos; además se hará un rápido resumen sobre el estado actual de esta línea en torno al DRL.

2.2. Reinforcement Learning

En el campo de la psicología, la toma de decisiones consiste en un proceso por el cual un ser humano toma una decisión determinada en el ciclo de su vida para solventar un problema buscando siempre el resultado más óptimo posible. Dicha decisión puede acarrear o no una serie de consecuencias, haciendo que el problema a afrontar se resuelva de forma satisfactoria o termine siendo un completo desastre. Para estudiar el proceso por el cual se generan dichas tomas de decisiones, la psicología y la neurociencia enfocaron sus primeros estudios en el cambio de actitud de los animales bajo ciertas circunstancias. Por ejemplo, en la psicología, [Thorndike, 1898], estudió el cambio de actitudes futuras en base a experimentaciones de prueba y error de varios tipos de animales. Por otro lado, en la neuropsicología, [Schultz, Dayan y Montague, 1997], se pudo demostrar cómo a nivel neuronal, dichos cambios de actitud afectaban directamente al proceso neuronal de tomar una decisión gracias a ese *refuerzo* generado por una recompensa o castigo que evocaba que la acción realizada era positiva o negativa. Estos estudios demostraban empíricamente, que el comportamiento podía ser modelado y controlado para que un sujeto pudiera corregir ciertas acciones que pudiera realizar en el futuro.

Los investigadores han buscado una forma científica de modelar matemáticamente este proceso cognitivo, lo que se consiguió por primera vez en [Bellman, 1957] mediante los Procesos de Decisión de Markov o *Markov Decision Process* (MDP). Los MDP, también llamados *stochastic dynamic programs* o *stochastic control problems*, son

descritos por autores como [Puterman, 2014] como modelos para la toma de decisiones secuenciales cuando los resultados son inciertos. Esto quiere decir que los MDP simulan un modelo matemático por el cual se puede representar mediante estados y acciones los resultados posibles que un ser humano podría llegar a alcanzar mediante el proceso neuro-psicológico de toma de decisiones. A partir de esta modelización matemática, se buscaron maneras de resolver de forma más eficiente los MDP. Una aproximación es el llamado *Dynamic Programming* que busca optimizar y descubrir una política óptima de resolver un MDP cuando se conocen todos sus estados. Varios autores como [Howard, 1964], [Bellman y Dreyfus, 2015], [Bertsekas, 1976] o [Dreyfus y Law, 1977] estudiaron tanto los MDP como las formas de resolverlos mediante esta técnica. La otra aproximación es la llamada *Linear Programing* que busca resolver un MDP cuando todos sus estados o acciones no son completamente conocidos. En este punto, existen varias aproximaciones como por ejemplo la de [Watkins, 1989] en su tesis doctoral, que presentó un algoritmo de modelo libre o *model free* [1] para resolver los MDP llamado *Q-Learning*. Dicho algoritmo, busca maximizar las recompensas de un agente inteligente mediante el cálculo de la estimación de la recompensa esperada futura dado el estado actual; de ésta forma el agente es capaz de actuar en cada estado del MDP para conseguir alcanzar los mejores estados posibles. Más adelante, [Watkins y Dayan, 1992] en su sección 3, demostraron matemáticamente que el algoritmo convergía haciendo uso de un MDP artificial controlado al que llamo *Action Replay Process* (ARP). Otra aproximación distinta fue la de [Metropolis y Ulam, 1949] que acuñó el nombre de un casino en Mónaco: **Monte Carlo** (MC). MC es otro algoritmo *model free* que acumula experiencias basadas en episodios completos para luego realizar una media que devuelve el valor que tomarán las acciones del agente. Dicho valor medio obtenido, converge en la función de valor y determina cual debe de ser la política óptima a seguir.

Q-Learning y MC fueron los algoritmos que dieron forma a la resolución de MDPs en entornos no conocidos, acercándose a lo que un ser humano puede experimentar cuando se encuentra en un entorno en el que no sabe qué resultados pueden originar sus acciones. Con dichas contribuciones, [Sutton y Barto, 1998] formalizaron lo que hoy en día se conoce como el Aprendizaje por Refuerzo o *Reinforcement Learning*. Por lo tanto, el concepto de RL es formalizado matemáticamente como un MDP que debe ser resuelto mediante una técnica iterativa de prueba/error por la cual un algoritmo determinado se va ajustado poco a poco para definir una política de decisión que consigue un agente inteligente capaz de simular la toma de decisiones de un ser inteligente real.

Varios autores han aplicado de forma satisfactoria los conceptos postulados del RL, generando agentes inteligentes capaces de jugar de forma efectiva a varios juegos. [Mahadevan y Connell, 1992] crearon un robot llamado OBELIX que aprendió cómo poder empujar cajas. [Schaal, 1997] construyó un humanoide capaz de aprender a interpretar un problema de balance de una polea. [Tesauro, 1995] en los laboratorios de IBM creó un agente capaz de jugar al juego de mesa Backgammon[2] mediante el uso de algoritmos basados en TD-λ que consiste en un algoritmo parecido al *Q-Learning* pero que contempla λ pasos futuros. Otros autores como, [Riedmiller y col., 2009], presentan un robot inteligente capaz de jugar al fútbol mediante el uso de RL. Por último, [Mülling y col., 2013], demuestran cómo se puede jugar al tenis de mesa mediante un robot que ha aprendido mediante el uso de técnicas de RL.

[1] Recordemos: consisten en algoritmos que resuelven casos de MDP con estados desconocidos
[2] https://en.wikipedia.org/wiki/Backgammon

2.3. Convolutional Neuronal Networks

La modelización matemática de las redes neuronales para los ordenadores, [McCulloch y Pitts, 1943], generó la creación de lo que hoy en día se conocen como las Redes Neuronales Artificiales o *Artificial Neuronal Networks* (ANN). A partir de ciertas variaciones de las ANN, [Fukushima y Miyake, 1982], modelizaron un nuevo tipo de ANN que era capaz de reconocer formas, patrones y colores de la misma forma que lo pudiera hacer un ser humano. Dicha modelización estaba basada en el concepto de *Convolución* [Hubel y Wiesel, 1959] por el cual se detectaron las células simples y complejas que se encargaban de la selectividad de orientación y de la detección de los bordes en los estímulos visuales. Gracias a eso, se estableció lo que hoy en día se llaman las Redes Neuronales Convolucionales o *Convolutional Neuronal Networks* (CNN). El problema de la aproximación planteada por Fukushima, era que en aquella época no existían los componentes tecnológicos necesarios para poder reproducir de forma realista los conceptos presentados por los autores, por lo que todo el estudio no pudo ser reproducido en una computadora que fuese capaz de distinguir los distintos elementos de una imagen. No fue hasta más adelante cuando, [LeCun y col., 1990], [LeCun y col., 1998] aplicaron los conceptos postulados por Fukushima e hicieron uso del concepto de *backpropagation* [Werbos, 1994, Rumelhart, Hinton y Williams, 1986] para crear la primera red neuronal capaz de reconocer números del 0 al 9 escritos a mano usando únicamente como entrada las imágenes de los extractos bancarios que contenían dichos dígitos. A esta red neuronal se la bautizó con el nombre de **LeNet**.

Las CNN describieron una nueva forma de hacer que un ordenador fuese capaz de hacerse una idea de cómo representar el mundo real mediante el análisis de imágenes, esto es, se hizo posible que un ordenador pudiese "ver" de forma artificial. En [Bengio, 2009], los autores describieron la importancia de poder generar agentes inteligentes haciendo uso de las CNN, capturando imágenes y reconociendo posibles elementos en ellas. [LeCun, Kavukcuoglu y Farabet, 2010], postularon los principios del llamado *Computer Vision*, o la habilidad por la cual una máquina es capaz de reconocer formas y elementos dentro de una imagen para poder identificarlos y clasificarlos.

EL impacto de las CNN y la visión por computación se hizo altamente popular cuando, [Krizhevsky, Sutskever e Hinton, 2012], sorprendieron al mundo siendo capaces de crear un modelo llamado **AlexNet** capaz de clasificar todas las imágenes de una conocida página web de competición llamada ImageNet[3], con una tasa de error muy baja (15,3 %) y entrenando la red neuronal en un cómputo de tiempo bastante razonable (5 días). Para ello, introdujeron dos nuevos cambios principales que no contenía la aproximación propuesto por **LeNet**: 1) uso de la activación ReLU [Hahnloser y Seung, 2001] en las CNN y en las Densas lo cual evitaba un problema común llamado *Vanishing Gradient* [Hochreiter y col., 2001] que ocurría en el proceso de "backpropagation" cuando se calculaban los gradientes para ajustar los pesos de la red neuronal; 2) la inclusión de *Dropout* [Srivastava y col., 2014] que hacía que ciertas neuronas de una capa fuesen desactivadas para evitar problemas de sobreentrenamiento u *overfitting* a la hora de entrenar los modelos. Más adelante, [Simonyan y Zisserman, 2014], presentaron un modelo de red neuronal llamado **VGG-16** que mediante una reemplazo de los filtros de las capas convolucionales de **AlexNet**, consiguieron entrenar un modelo capaz de clasificar las mismas imágenes de ImageNet con una tasa del 92,3 % de precisión. Después, [Szegedy y col., 2015],

[3] www.image-net.org

presentaron una nueva versión de red neuronal llamada **INCEPTION**. Esta red crea una red neuronal dentro de otra red neuronal, mediante la creación de varias capas conectadas por CNN y redes Densas de manera escalonada. De esta forma, esta red ha conseguido obtener una tasa del 93,3 % de precisión haciendo la misma prueba con el conjunto de entrenamiento de ImageNet.

Cuando una red neuronal aumenta en profundidad, esto es, cuando la red contiene más y más nodos conectados, surgen dos problemas que hacen que pierdan efectividad a la hora de realizar los cálculos necesarios. El primer problema que se produce consiste en que la señal que modifica los pesos de las redes neuronales, pierde fuerza haciendo que las capas anteriores (las menos profundas) pierdan capacidad de aprendizaje. Este problema es el que se ha descrito anteriormente como *Vanishing Gradient*. El segundo problema que se produce, es que al aumentar la profundidad de la red su espacio de parámetros también aumenta y, por lo tanto, se tienen que modificar todos los parámetros de todos los nodos para que estos puedan ser optimizados en el proceso de entrenamiento; ello hace que el error asociado al entrenamiento de dichos nodos sea mayor que en una red pequeña. Para solventar estos dos problemas, [He y col., 2016], presentaron una nueva aproximación llamada Redes Residuales. Esta red permite evitar estos problemas construyendo una red mediante unos módulos llamados modelos residuales, que mitigan los problemas descritos. La estructura de la red es similar a la aproximación **VGGNet** y utiliza algunas ideas de la red **INCEPTION**. Gracias a estos cambios producidos y a la adición de los modelos residuales se genera una red neuronal llamada **ResNet**. Éste diseño de red neuronal ha obtenido en la misma prueba de ImageNet una tasa del 95,51 % de precisión, mejorando de forma muy significativa el estado del arte sobre a clasificación de imágenes.

2.4. Deep Reinforcement Learning

El DRL es un concepto innovador que se apoya en un cúmulo de distintas aportaciones hechas en diferentes campos científicos. La primera aproximación conocida fue realizada por [Mnih y col., 2015] en la que mostraron el funcionamiento del algoritmo DQN (Deep Q-Network) basado en el algoritmo *Q-Learning*. En este trabajo, los autores muestran cómo el algoritmo es capaz de jugar a una serie distinta de juegos de la consola Atari 2600[4] sin la necesidad de la intervención de un ser humano. En la experimentación, se muestra un modelo de redes neuronales profundas que toman como entrada imágenes de un video-juego en blanco y negro reescaladas a 84x84 píxeles para conseguir entrenar un agente que sea capaz de realizar acciones discretas en un espacio de dimensión entre 2 y 18. El algoritmo hace uso de $\epsilon - greedy$ como función de aplicación de ruido y una red neuronal profunda formada por 3 capas de neuronas convolucionales y 2 capas de neuronas completamente conectadas o densas. La red neuronal se alimenta de las imágenes recibidas y del cálculo del *discounted reward* descrito por las ecuaciones de Bellman que permite estimar la recompensa futura dado el estado actual (*Q-Learning*).

Un poco más adelante, [Van Hasselt, Guez y Silver, 2016] muestran una mejora del algoritmo DQN llamada *Deep Double Q-Learning* basado en el algoritmo *Double Q-Learning* [Hasselt, 2010]. Este algoritmo, realiza un entrenamiento más eficiente que el DQN porque corrige un fallo que tiene el algoritmo original por el cual sufre una tendencia a sobreestimar los valores vinculados a ciertas acciones específicas. Para corregir esta sobreestimación, los autores proponen una modificación

[4] https://es.wikipedia.org/wiki/Atari_2600

de la función de cálculo del *target* de la red descomponiendo la operación máxima en el cálculo y_t en una selección de acción y una evaluación de la acción. Esto es, $y_t = r_{t+1} + \gamma \max_{a'} Q(s'_{t+1}, a'|\theta'_t) \rightarrow y_t = r_{t+1} + \gamma Q(s_{t+1}, \arg\max Q(s, a|\theta_t), \theta'_t)$. Esta solución optimiza el algoritmo original manteniendo la red neuronal, la exploración y la lógica general intactas.

Las contribuciones basadas en los algoritmos de *Q-Learning* tienen la particularidad de no ser eficientes cuando el espacio de acciones es continuo. Por ello, [Lillicrap y col., 2015] muestran un algoritmo llamado Deep Deterministic Policy Gradient (DDPG) basado en el algoritmo original Deterministic Policy Gradient (DPG) desarrollado por [Silver y col., 2014]. La idea de este algoritmo, se basa en la introducción de una arquitectura basada en dos redes neuronales distintas, una llamada **Actor Network**, que recibe el estado del agente (varias imágenes que generan una transición) y realiza las acciones emitiendo resultados continuos (neuronas con activaciones tangente o sigmoide que dan salidas continuas entre un intervalo cerrado). Una vez que la Actor Network toma las acciones pertinentes, otra red neuronal llamada **Critic Network** recibe el estado del agente y las acciones realizadas y evalúa mediante una implementación del DQN una estimación de los resultados posibles a obtener haciendo uso del *discounted reward*. Aunque el algoritmo DDPG es una solución para los espacios continuos, no siempre puede llegar a ser la mejor opción en todos los contextos debido a que su proceso de aprendizaje es demasiado lento y no siempre puede terminar convergiendo, ya que es más inestable que la aproximación DQN. Debido a esto, [Mnih y col., 2016] presentan una mejora llamada A3C que usa un algoritmo basado en Actor-Critic pero siendo asíncrono. La idea es eliminar la memoria que guarda las experiencias realizadas por el agente y sustituirlo por la ejecución paralela de distintos agentes asíncronos. La experiencia obtenida por cada agente asíncrono, servirá para entrenar la red neuronal y actualizar los pesos para definir políticas de decisión más optimizadas. Por último, [Espeholt y col., 2018], mostraron una versión aún más optimizada del algoritmo A3C pero haciendo uso de sistemas distribuidos.

Otros autores han propuesto aproximaciones en las cuales se hacen uso de algoritmos evolutivos para entrenar agentes inteligentes. En [Hein y col., 2017] los autores muestran un algoritmo para entrenar un agente inteligente basado en un algoritmo online que actualiza de forma dinámica las políticas de decisión de un agente haciendo uso de controladores difusos (*Fuzzy controlers*). Los autores proponen una arquitectura que mezcla algoritmos evolutivos, reglas *Fuzzy* y redes neuronales para ser capaces de crear un agente inteligente en un espacio de acciones continuo. También hay autores que han mostrado aproximaciones por las cuales se entrena un agente inteligente a partir de la imitación de un agente de control [Peng y col., 2018a]. Otros autores, [Mirowski y col., 2018], han conseguido entrenar un agente inteligente, capaz de moverse por un mapa haciendo uso únicamente de las imágenes obtenidas por el servicio de Google Maps Street View[5] para poder ir de un punto inicial ("estoy aquí") a un punto destino ("voy allí").

La aparición de DRL ha sido útil y bastante impactante cuando se ha hecho posible la creación de agentes inteligentes capaces de ganar a los mejores jugadores del mundo en ajedrez y shogi [Silver y col., 2017] o en el juego Go [Silver y col., 2016].

[5]https://mapstreetview.com/

2.5. Transfer Learning

El aprendizaje de los modelos de DL es lento y requieren de un coste computacional muy elevado. Además, en ocasiones sucede que cuando se quiere entrenar un modelo de DL para un entorno determinado, puede darse el caso de no tener los conjuntos de entrenamiento suficientes para poder llevarlo a cabo. Debido a estas necesidades, los investigadores comenzaron a abrir una nueva línea de investigación que fuese capaz de aprovechar las experiencias adquiridas por una red neuronal, ya entrenada en un dominio de interés para poder ser aplicadas en otro dominio. A este concepto se le llamo aprendizaje por transferencia o *Transfer Learning* [Pan y Yang, 2010, Lazaric, 2012].

El concepto de TL se ha estudiado y usado de forma profunda dentro del DL, para poder aprovechar el conocimiento adquirido por las redes durante el entrenamiento. Dicho conocimiento es posible reutilizarlo para poder identificar patrones similares entre distintos tipos de imágenes y así reducir los tiempos totales de una CNN por ejemplo. Algunos autores, [Yosinski y col., 2014], han identificado métricas útiles para cuantificar y establecer bajo qué circunstancias es posible generalizar o no la transferencia de conocimiento de las redes, bajo distintos modelos de DL y conjuntos de datos de entrenamiento. Otros autores, [Karpathy y col., 2014], demuestran cómo los *feature maps* que representan las propiedades de las imágenes extraídas por una CNN pueden ser aprovechadas para entrenar un conjunto de datos distinto y aumentar el rendimiento de entrenamiento de un modelo basado en CNN. Los autores presentan una analogía haciendo uso del concepto de bolsa de palabras o *bag of words* muy usado en la clasificación supervisada (k-means, Support Vector Machines...) como una representación de datos para poder reducir los tiempos de entrenamiento. En, [Esteva y col., 2017], los autores presentan una aproximación por la cual han hecho uso de la red INCEPTION V3 pre-entrenada con las imágenes de la web de competición ImageNet para poder diseñar un algoritmo capaz de clasificar cáncer en la piel. En, [Huang y col., 2013], los autores presentan una arquitectura que hace uso de la transferencia de conocimiento entre las capas ocultas de las redes neuronales para poder entrenar, con pocos datos, un sistema de detección de lenguajes.

El TL dentro del RL es una línea de investigación, por la cual se estudia la capacidad de generalizar modelos que permitan entrenar agentes inteligentes para poder realizar tareas distintas en un menor tiempo de entrenamiento. Algunos investigadores, como [Taylor y Stone, 2009] ya postulaban algunos conceptos y métricas a tener en cuenta para verificar si una transferencia de conocimiento entre dos agentes inteligentes podía ser o no factible y establecía una serie de métricas para evaluar la calidad de la transferencia. Autores como, [Konidaris y Barto, 2006], presentan técnicas para acelerar el proceso de aprendizaje, mediante el uso de una función que da al agente una primera estimación sobre las recompensas posibles a obtener entre tareas similares. Dicha aproximación se obtiene de las recompensas que ha obtenido en el pasado. Otros autores, [Taylor y Stone, 2007], presentan una aproximación para conseguir una transferencia de conocimiento entre agentes inteligentes que no realizan las mismas tareas. Para ello, presentan un algoritmo que hace el uso de reglas para hacer las transferencia de conocimiento oportunas. En, [Taylor, Kuhlmann y Stone, 2008], los autores presentan una metodología de transferencia de conocimiento llamada MASTER, que de forma automática aprende a realizar un mapeo entre las distintas tareas consecutivas que realiza un agente para acelerar el proceso de aprendizaje. Por último, [Barreto y col., 2017] proponen una serie de conceptos

para sentar las bases del TL desarrollando un *framework*[6] que está basado en el desarrollo de dos conceptos previos: 1) los *successor features* que son una generalización de los *successor representations* propuesta por [Dayan, 1993]; 2) el *generalized policy improvement* que es un teorema que extiende el teorema clásico de mejora de la política propuesto por [Bellman, 2013].

Como se puede observar, existen bastantes aproximaciones de autores que hacen uso de técnicas de TL para poder acelerar el entrenamiento de agentes inteligentes que sean capaces de realizar tareas maximizando las recompensas. Ahora bien, el TL es una línea de investigación que no está explotada dentro del DRL. No existe a día de hoy un modelo concreto que sea capaz de generalizar el aprendizaje entre distintos agentes inteligentes para resolver problemas transfiriendo el conocimiento de las CNN y las redes Densas. Algunos autores, han presentado aproximaciones para entrenar agentes inteligentes a través de la mímica, como por ejemplo la aproximación presentada por [Parisotto, Ba y Salakhutdinov, 2015] que muestran el entrenamiento de un agente inteligente mediante un método llamado *Actor-Mimic*. Dicho método consigue demostrar la existencia de transferencia de conocimiento a través de la mímica, cuando la red generada es grande. Otros autores presentan aproximaciones parecidas, como [Peng y col., 2018b], que presentan una red neuronal que aprende a entrenar un agente inteligente a través de la mímica haciendo uso de un agente de control que funciona correctamente. Otros autores como [Narasimhan, Kulkarni y Barzilay, 2015], presentan una red basada en DQN para poder jugar a un juego basado en texto y tomar la mejores opciones posibles entre los resultados obtenidos por la aproximación presentada; los autores comentan que el modelo presentado en dicho contexto ha sido capaz de hacer uso de las bondades del TL, pero no han sido capaces de presentar un modelo generalista para todos los casos.

Como se puede ver, estas aproximaciones no usan un modelo generalista que intente transferir el conocimiento adquirido anteriormente de un agente a otro. Se necesita por lo tanto, un estudio profundo para ver hasta qué punto se puede transferir o no el conocimiento entre agentes entrenados para poder calcular en distintos escenarios, si la transferencia de conocimiento aporta una mejora en los tiempos de entrenamiento y en las recompensas obtenidas. Por lo tanto, y después de haber realizado un completo repaso por la literatura, se puede observar que uno de los campos más interesantes, y de los cuales aún se desconocen muchas cuestiones, es la línea de investigación basada en el TL para los modelos construidos con DRL. En esta tesis, se va a seguir dicha línea de investigación y se van a presentar una serie de experimentos para poder expandir un poco más ese conocimiento o al menos, para poder reafirmarlo.

[6]Consiste es un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar.

Capítulo 3

Sistema de Reinforcement Learning

3.1. Introducción

Una vez presentada la hipótesis formal H_0 de esta tesis y explicada la motivación del uso del DRL, en este capítulo se van a presentar las distintas herramientas técnicas utilizadas así como el funcionamiento lógico del algoritmo utilizado durante la experimentación. Además, se describirá cuáles han sido las configuraciones realizadas en cada herramienta utilizada argumentado el porqué de la elección de los distintos parámetros existentes.

Este capítulo está dividido en las siguientes secciones: 1) Entorno de aprendizaje: explica cuál es el banco de trabajo utilizado para crear un agente inteligente; 2) Redes Neuronales profundas: explicación de la configuración de la red neuronal utilizada; 3) Algoritmo de Reinforcement Learning: qué algoritmo se ha utilizado y porqué; 4) Algoritmo DQN: explicación de cómo los puntos 2 y 3 se juntan para generar el algoritmo final que se utilizará en toda la fase de experimentación.

3.2. Entorno de aprendizaje

Para poder realizar la experimentación de forma efectiva, se necesita un entorno de aprendizaje válido que permita al investigador aplicar de forma directa los algoritmos que diseñe. Para evitar tener que generar un entorno determinado y tener que estar diseñando las recompensas necesarias para cada estado en el que pueda estar el agente a entrenar, se ha hecho uso de una librería de desarrollo que permite generar un banco de trabajo, para que el investigador pueda: 1) tener un entorno funcional ya creado; 2) un sistema de recompensas asociado a las acciones que realice el agente en dicho entorno; 3) liberar al investigador de cualquier funcionalidad técnica que involucre tener que conectar el programa desarrollado al entorno de aprendizaje. Con estas premisas, en esta tesis se ha hecho uso de las librería *OpenAIGym* [Brockman y col., 2016] que permite la creación de un banco de trabajo para poder realizar desarrollos rápidos basados en RL o DRL.

3.2.1. OpenAIGym: Entorno de aprendizaje de algoritmos RL o DRL

OpenAIGym está escrita en el lenguaje de programación Python[1] lo que le permite ser integrada con otro tipo de librerías basadas en el mismo lenguaje de programación que veremos más adelante. *OpenAIGym* ofrece una serie de video-juegos

[1] https://www.python.org

TABLA 3.1: Descripción de los métodos disponibles en *OpenAIGym*.

Método	Devuelve
make(*juego*)	Instanciación del entorno del video-juego deseado.
action_space_n	Número total de acciones disponibles en el entorno.
reset()	Imagen con estado inicial del video-juego en un episodio.
step(*acción*)	Recompensa r_t, nuevo estado s_{t+1}, estado terminal (booleano).
render()	Imagen renderizada real del juego en pantalla en s_t.

preinstalados (entornos) que permiten poder desplegar los algoritmos de RL diseñados previamente por el investigador, para no tener que preocuparse acerca del sistema de recompensas a desarrollar o la conexión que se tenga que realizar entre el programa y el emulador que ejecute el video-juego.

La Tabla 3.1 describe los distintos métodos existentes en la librería *OpenAIGym* para poder interactuar con un video-juego determinado. Para poder hacer uso de un video-juego determinado, se tiene que instanciar dicho juego haciendo uso del método **make** que, dado el nombre de un juego de la forma *NombreJuego-Vk* donde k es el número de fotogramas que se saltan por cada paso temporal t, se conseguirá hacer todo el proceso lógico de conectar el programa de IA con el emulador del juego a reproducir. En el caso de esta tesis se ha hecho uso de tres juegos distintos basados en la máquina arcade Atari 2600[2]: 1) *SpaceInvadersDeterministic-v4*; 2) *DemonAttackDeterministic-v4*; 3) *QbertDeterministic-v4*. La Figura 3.1 contiene un fotograma de cada juego que será utilizado durante el proceso de experimentación. En las experimentaciones realizadas, hemos decidido saltar cuatro fotogramas por cada paso temporal t. Esta decisión se ha tomado para evitar altos costes computacionales ya que procesar todos los fotogramas es bastante costoso en términos de computación.

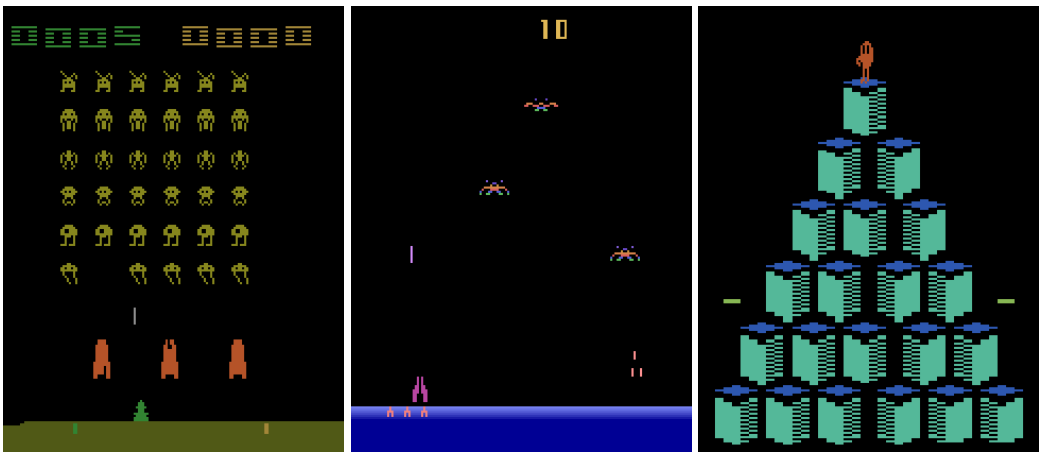

FIGURA 3.1: Juegos utilizados en la experimentación; (a) izquierda: *Space Invaders*; (b) centro: *Demon Attack*; (c) derecha: *Qbert*

El método **action_space_n** devuelve el numero total de acciones del video-juego instanciado. Esto es clave para definir dos cuestiones: 1) el rango máximo de acciones que se pueden enviar al video-juego mediante el método **step** (será descrito en

[2]https://en.wikipedia.org/wiki/Atari_2600

breve); 2) como entrada de la red neuronal para crear una máscara con un vector *one-hot*[3] que limite las acciones a predecir.

El método **reset** reinicializa el juego a su estado inicial y se usa para determinar el comienzo de un nuevo episodio cuando el agente alcanza un estado terminal o cuando se comienza un entrenamiento desde el principio (Episodio número 0). Este método devuelve un tensor con los valores [*Alto*, *Ancho*, *Canal*], donde *Alto* representa el número de pixeles de alto de una imagen, *Ancho* representa el número de pixeles de ancho de una imagen y *Canal* representa los valores que toman la paleta de colores **RGB**[4]. En nuestro caso, y para los juegos que hemos explicado anteriormente, los valores obtenidos han sido un tensor con la forma de $[210, 160, 3]$. Por lo tanto, el método **reset** nos devolverá una imagen de $210x160x3$ que es un fotograma que representará el estado inicial del juego s_0. La razón del uso de **reset** es debido a la satisfacción de dos cuestiones: 1) obtener la primera imagen inicial del juego; 2) forzar al entorno a comenzar un nuevo episodio cuando el agente ha llegado a un estado terminal.

El método **step** toma como entrada un número discreto entre [0, **action_space_n**]. Este método, toma como referencia el estado actual del agente s_t (la imagen actual o fotograma) y realiza la acción solicitada dentro del entorno. Cuando dicha acción ha sido realizada, el método devuelve tres variables: 1) un fotograma que representa el siguiente estado fruto de la acción realizada s_{t+1}; 2) una recompensa r_t asociada; 3) un valor booleano (True o False) indicado si el agente ha entrado o no en estado terminal (cuando se ha perdido la partida). Con este método, se podrán obtener los datos necesarios para poder saber qué repercusiones han tenido las acciones realizadas por el agente y se podrán guardar dichos valores en un vector que satisface un MDP [s_t, s_{t+1}, r_t, terminal].

El método **render** muestra en una ventana gráfica el renderizado del tensor obtenido en las funciones **reset** y **step**. Este método se utiliza para comprobar visualmente en un monitor cómo se comporta el agente entrenado mediante el modelo programado. En el caso que nos ocupa, este método se ha utilizado únicamente para evaluar visualmente cómo se comporta el agente entrenado y para interpretar los resultados obtenidos en los entrenamientos.

3.2.2. Análisis taxonómico de los juegos utilizados

En esta tesis se han hecho uso de los video-juegos *SpaceInvaders*, *DemonAttack* y *Qbert* para validar la hipótesis formal H_0 presentada. Los dos primeros tienen ciertas "similitudes" en cuanto a forma de jugar y objetivos, mientras que el último, es completamente distinto. No obstante, para poder cuantificar realmente dichas "similitudes" y "diferencias"'se ha realizado un análisis taxonómico de los juegos para comprender exactamente qué los diferencia y qué los iguala. Dentro de este análisis se han extraído dos tipos diferentes de características: 1) **Características lógicas**: que consisten en las características que definen el comportamiento del juego, esto es, cómo se mueve el agente en el espacio, cómo es el sistema de recompensas del juego, si conseguir una recompensa válida es sencillo o no etc . . . ; 2) **Características visuales**: que identifican qué partes visuales dentro de cada juego son estáticas o dinámicas, esto es, qué partes del juego nunca se van a alterar durante una episodio y qué partes sí.

[3] https://hackernoon.com/what-is-one-hot-encoding-why-and-when-do-you-have-to-use-it-e3c6186d008f

[4] *Red, Green, Blue* la composición básica de los colores con un rango de valores de entre 0 a 256

Características lógicas

La tabla 3.2 resume las características lógicas que identifican las diferencias fundamentales de cada juego. La columna *Espacio Acciones* representa el espacio total de acciones disponibles para el agente en el entorno, la columna *Grado Libertad* representa el grado que posee el agente para moverse por el entorno, la columna *Duración Terminal* indica la media y la desviación típica en segundos del tiempo transcurrido para que el agente entre en un estado terminal si no ejecuta ninguna acción, la columna *Reward Positivo Esperado* describe la media y la desviación típica de la diferencia de los t necesarios para obtener un reward positivo en cada juego calculada en los primeros 50000 t, la columna *Tipo Reward* describe cómo están fijadas las recompensas en el juego y por último, la columna *Dificultad Reward* indica si el agente necesita ir pasando niveles para ir consiguiendo las recompensas o si todas ellas se pueden obtener directamente en el nivel del juego.

Para el juego *SpaceInvaders*, el espacio de acciones es de 6 (*[Fire, Right, Left, RightFire, LeftFire]*). El grado de libertad del agente es 1D Continuo ya que este solo puede recorrer de forma continua la región inferior del entorno de izquierda a derecha hasta unos límites determinados. La facilidad de morir, es media porque el agente tiene una serie de protecciones que permiten que no reciba daño de sus enemigos. El reward es inmediato porque todas las naves enemigas están aglutinadas en una zona y la facilidad de impactar a una es bastante sencilla. El tipo de reward es variable porque cada nave tiene una recompensa distinta dependiendo de la fila en la que esté situada, siendo las más altas las naves que más recompensa dan y las más bajas las que menos. Esto último es interesante porque permite una aleatoriedad bastante alta a la hora de obtener recompensas en los distintos episodios.

Para el juego *DemonAttack*, el espacio de acciones es de 6 (*[Fire, Right, Left, RightFire, LeftFire]*). El grado de libertad del agente es de 1D Continuo, gozando de las mismas características que el juego *SpaceInvaders*. La facilidad de morir es muy baja porque solo uno de los demonios dispara al agente. De hecho, los demonios responden más agresivamente cuando el agente dispara, por lo tanto, si el agente no dispara las partidas pueden hacerse muy largas. Esto nos dice que, si una partida es muy larga y la recompensa es muy baja entonces el agente ha estado "perdiendo el tiempo" sin hacer nada. El reward no es inmediato porque se necesitan varios pasos temporales para poder matar a un enemigo y esto ocurre porque la cantidad de enemigos en pantalla es pequeña y es complicado acertar a uno de ellos. El tipo de recompensa es incremental, porque los demonios salen por oleadas y cada oleada tiene una recompensa distinta, a más oleadas, enemigos con más recompensa.

Por último, para el juego *Qbert*, el espacio de acciones es de 6 (*[Noop, Fire, Up, Right, Left, Down]*). El grado de libertad es 2D Discreto porque el agente se puede mover en cualquier dirección y los lugares válidos en los que puede estar el agente son limitados (el número de cubos disponibles, en el resto de lugares el agente muere). La facilidad de entrar en un estado terminal es muy alta porque aunque la media de duración de los episodios sea más alta que el juego *SpaceInvaders* debido a los enemigos, el agente puede salirse del área del juego traduciéndose en un estado terminal inmediato. Con esta información se puede deducir que los episodios de este juego serán muy cortos y que si por alguna razón existe un episodio largo pero con una puntuación muy baja, es debido a que el agente ha estado repitiendo movimiento todo el rato en una región cerrada de los cubos y no ha intentado explorar más fuera de éstos.

TABLA 3.2: Características lógicas de los video-juegos utilizados

Game	Espacio Acciones	Grado Libertad	Duración Terminal(seg)	Duración Terminal(t)	Reward Positivo Esperado (t)	Tipo Reward	Dificultad Reward
SpaceInvaders	6	1D Continuo	$28{,}8 \pm 4{,}970$	680 ± 0	$112{,}706 \pm 113{,}132$	Incremental	Mismo nivel
DemonAttack	6	1D Continuo	$462 \pm 175{,}087$	7686 ± 0	$287{,}691 \pm 1466{,}251$	Incremental	Distinto nivel
Qbert	6	2D Discreto	69 ± 0	1038 ± 0	$97{,}850 \pm 124{,}664$	Definido	Distinto nivel

Características visuales

Además de las características lógicas, se debe tener en cuenta otro tipo de características. Dichas características, corresponden al entorno visual de cada juego. Las CNN identifican qué partes de cada juego son estáticas y cuáles son dinámicas. Este análisis genera un punto de inflexión, que ayuda a interpretar la capacidad de adaptabilidad de las CNN cuando se cambia de un juego a otro. En esta tesis se han considerado dos propiedades para identificar la adaptabilidad de las CNN: 1) **Contenido dinámico** que identifica las partes del juego que contienen elementos en movimiento (enemigos, vidas, puntuaciones, agente..) y cómo se comportan; 2) **Contenido estático** que identifica aquellas partes que son fijas en el juego y qué papel juegan.

La Figura 3.2, muestra qué partes de cada juego contienen contenidos dinámicos y estáticos. Todos los juegos comparten la característica de tener un fondo negro que representa "la nada" y que es interpretado por las redes neuronales como una región de píxeles que no aportan información relevante.

En *SpaceInvaders*, el contenido inferior es estático y no cambia durante la vida del juego. En dicho contenido estático, el agente representa un contenido dinámico que se desplaza continuamente de izquierda a derecha hasta llegar a los márgenes permitidos. Los enemigos representan otro contenido dinámico que se mueve de forma continua de derecha a izquierda mientras va descendiendo poco a poco. Este movimiento se realiza a la misma velocidad en ambas direcciones y describe el mismo recorrido en todos los episodios. En la parte superior, se encuentra la puntuación, la cual desaparecerá cuando aparezca la nave especial que tiene una recompensa mayor, indicando al agente la necesidad de destruirla para optimizar la recompensa máxima.

En *DemonAttack*, el contenido inferior es estático y muy parecido al juego *SpaceInvaders*, aunque la cantidad de contenido estático es mayor (ocupa más parte de la pantalla). En dicho contenido estático se encuentran: 1) las vidas del agente; 2) el agente que describe el mismo tipo de movimiento que el agente del juego *SpaceInvaders* con la salvedad de que este posee límites más extensos. Los enemigos, los disparos y la puntuación son contenidos dinámicos. Los enemigos se mueven de forma estocástica y realizan disparos cuya representación gráfica no siempre es igual. Los disparos del agente tienen la misma forma y son bastante similares al juego *SpaceInvaders*. Por último, la puntuación está siempre ubicada en la misma región de la pantalla pero va cambiando conforme el agente va aumentando su puntuación total.

En *Qbert*, el contenido central es estático. Dicho contenido lo representan las cajas por las cuales se puede mover el agente de forma libre. El agente, los enemigos y los pedestales izquierdo y derecho representan los contenidos dinámicos del juego. El agente y los enemigos, describen un movimiento discreto diagonal que puede ir hacia arriba o hacia abajo. El agente no tiene limitación en sus movimientos por lo que si realiza una acción que le desplace hacia un punto que no se encuentre dentro del contenido estático, el episodio entrará directamente en un estado terminal.

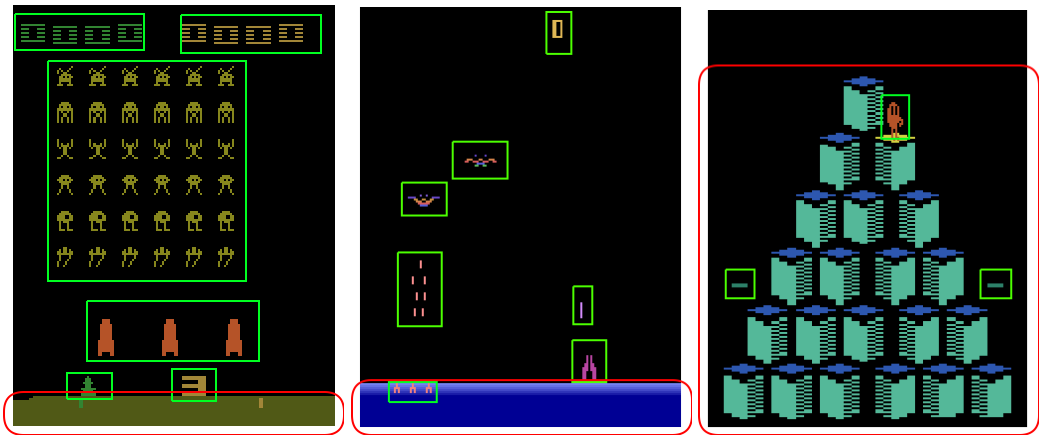

FIGURA 3.2: Zonas dinámicas y estáticas de los distintos juegos: (a) Verde: Contenidos dinámicos; (b) Rojo: Contenidos estáticos

3.3. Redes Neuronales Profundas

Como bien se ha explicado, el agente se entrena mediante un entorno de aprendizaje que ejecuta un video-juego, pero para que este agente pueda "ver" qué ocurre en dicho entorno y pueda "decidir" las mejores acciones a tomar, se necesita crear un modelo que le permita visualizar los tensores que recibe mediante el entorno de aprendizaje y, a partir de éstos, ejecutar las mejores acciones que maximicen la recompensa obtenida. Para hacerlo posible, se ha hecho uso de las Redes Neuronales Profundas o *Deep Learning* (DL), que consiste en una técnica de *machine learning* formada por una serie de algoritmos basados en redes neuronales artificiales que identifican distintos patrones, formas, sombras y figuras dentro de una imagen mediante el uso de filtros o *feature maps* para luego ser capaz de tomar una decisión acertada bajo un modelo concreto formada por redes neuronales completamente conectadas. En la aproximación realizada en este tesis, se ha creado un modelo que tiene dos capas principales: 1) Una capa formada por Redes Neuronales Convolucionales o *Convolutional Neuronal Networks* (CNN); 2) Una capa formada por neuronas ocultas completamente conectadas o *Dense*. Ambas capas serán explicadas más adelante. Gracias a esta arquitectura desarrollada, se consigue calcular una función de salida lineal que determina cual debería ser la mejor acción a realizar en cada estado del agente.

La Figura 3.3 muestra la arquitectura base DL utilizada en la experimentación de esta tesis. Dicha arquitectura está basada en el artículo académico presentado por [Mnih y col., 2015] que demostró allá en el 2015 que la arquitectura diseñada era capaz de jugar a una serie de video-juegos basados en la máquina arcade Atari. Aunque la arquitectura presentada por los autores es muy eficiente y está probada que funciona de forma correcta, en esta tesis se ha probado una aproximación un poco distinta para intentar mejorar la eficiencia de la red haciendo unos pequeños ajustes en su arquitectura e intentar mejorar un poco la eficiencia de la red a la hora de obtener sus resultados de salida.

La Tabla 3.3 desglosa los parámetros utilizados en la red neuronal usada en esta tesis. En la tabla se especifica qué tipo de capa se ha utilizado, qué tipo de configuración se ha aplicado a dicha capa y cual ha sido la función de activación de las neuronas en caso de tenerla.

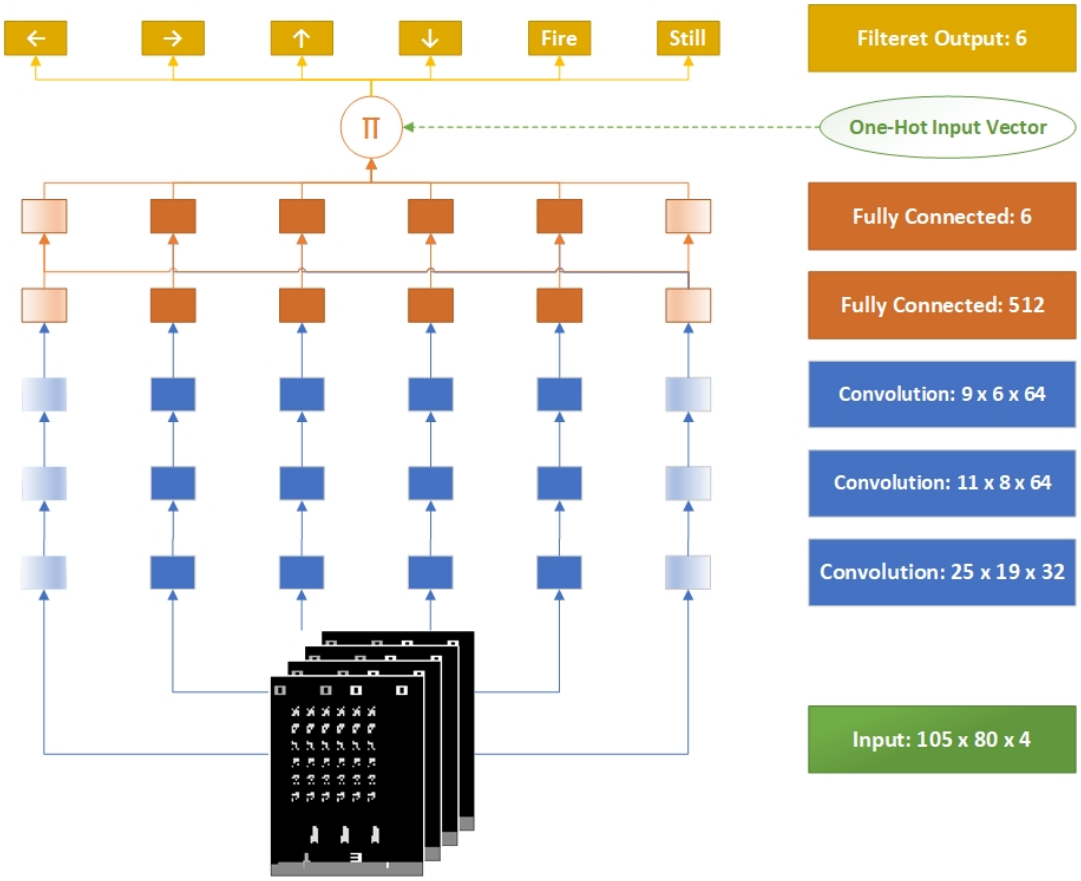

FIGURA 3.3: Representación gráfica de la red neuronal utilizada, ejemplo con un espacio de acciones de 6 elementos.

La capa de Entrada recibe un array multidimimensional de datos (tensor) formado por una imagen de $105x80x4$ que representa el estado s_t y recibe también un vector one-hot de [1, action_space_n] que representa la acción tomada por el agente a_t (por ejemplo si un agente tiene un espacio de acciones de 2, el vector podría ser [1,0] ó [0,1] para representar que el agente ha tomado la elección de izquierda o derecha). Este último vector, será utilizado más adelante para multiplicar la salida de la última capa Densa de la red para producir la salida final.

Las capas Convolucionales (tres en total), extraen los elementos de la imagen mediante la aplicación de N filtros de tamaño nxn y se les aplica una reducción llamada *Strider*[5] de la capa convolucional que reduce todos los *feature maps*[6] generados a mxm. La activación neuronal utilizada en estas neuronas son las unidades de rectificación lineal o *Rectified Linear Uni* (ReLU) [Hahnloser y Seung, 2001] que tienen una función de activación de la forma $f(x) = \text{máx}(0, x)$.

Flatten es una operación intermedia que "aplana" el tensor producido por la capa anterior. Se suele usar principalmente para conectar las salidas de las capas convolucionales con las entradas de las capas densas. Lo que hace esta operación es

[5]El filtro aplicado en la convolución es más pequeño que la imagen (suelen ser de $1x1$ o $3x3$) por lo que se aplica F veces hasta completar la región completa de la imagen. *Strider* define cuántos saltos queremos dar entre las regiones para aplicar el filtro. Por ejemplo un *Strider* de $s = 2$ hará que se produzcan saltos de 2 en 2 pixeles haciendo que se aplique la convolución menos veces.

[6]Se hacen llamar *feature maps* a los resultados obtenidos al multiplicar la imagen de entrada por un filtro para realizar la operación convolucional.

TABLA 3.3: Arquitectura de la red neuronal utilizada.

Capa/Operación	Configuración	Activación
Entrada red	Imagen de $105x80x4$ más vector de acciones one-hot [1, action_space_n]	N/A
Convolucional 1	32 Filtros de $8x8$ y downsample $4x4$ (Strider)	ReLU
Convolucional 2	64 Filtros de $4x4$ y downsample $2x2$ (Strider)	ReLU
Convolucional 3	64 Filtros de $3x3$ y downsample $1x1$ (Strider)	ReLU
Flatten	N/A	N/A
Densa 1	512 neuronas ocultas completamente conectadas	ReLU
Densa 2	Capa de salida, action_space_n neuronas ocultas completamente conectadas	Lineal
Salida Filtrada final	Multiplicación de Densa 2 red por vector acciones one-hot [1, action_space_n]	N/A

transformar los tensores de dimensión n producidos por la salida de la última capa convolucional en un vector de una dimensión. Por ejemplo, en un sistema de una red neuronal basado en una capa convolucional que admite un tensor $[64, 32, 32]$, el resultado de usar Flatten sería un vector de 65536 elementos.

La capa Densa o *Fully connected layer* es una capa formada por D neuronas ocultas completamente conectadas, esto quiere decir que todas las neuronas de una capa oculta están conectadas a todas las neuronas de la siguiente capa oculta. En el caso de esta arquitectura se ha hecho uso de una capa Densa de 512 neuronas ocultas con una activación ReLU y una capa final Densa con D neuronas ocultas igual al espacio de acciones y con una activación lineal ($f(x) = x$) que representa las probabilidades finales de cada acción. Esta última capa final con esta activación, será clave para el cálculo de la función de ajuste de la red $L(\theta)$ (*loss function*) mediante el uso de *Q-Learning* que será explicado más adelante.

La última capa de esta arquitectura, es una pequeña modificación de la arquitectura presentada por [Mnih y col., 2015] que multiplica la salida final lineal que produce la última capa por un vector one-hot que representa la acción a_t tomada en el estado s_t que se ha introducido a la red. Esta multiplicación por el vector one-hot genera una máscara que elimina las probabilidades que no interesen que la red pueda predecir. Un ejemplo sería el siguiente: si tenemos un espacio de acciones $a = 3$ (izquierda, nada, derecha) y solo interesaría que la red pudiera predecir únicamente 0 o 2 (izquierda, derecha), entonces se le pasaría el estado s_t y un one-hot vector de la forma $(1, 0, 1)$ para decirle a la red que solo interesa predecir esos dos valores. La razón por la cual se ha realizado esta última capa, es porque a la hora de calcular la función *loss* $L(\theta)$, la aplicación de la máscara agiliza los cálculos y mejora el tiempo total de las experimentaciones.

La red neuronal presentada se ha generado haciendo uso de la librería Keras[7] que ofrece una API de alto nivel basada en el lenguaje de programación Python. Keras actúa como capa de abstracción entre una librería que permite realizar cálculos de tensores y el programador. De las distintas librerías existentes para calcular tensores, se ha hecho uso de una de las más populares llamada TensorFlow[8], la cual es la encargada de realizar las computaciones necesarias para generar grafos y nodos que representan operaciones matemáticas de arrays multidimimensionales de datos (tensores). La elección de Keras es debido a que ofrece una forma sencilla y rápida de implementar modelos de redes neuronales para una persona que no tiene conocimientos avanzados sobre el manejo de operaciones de tensores. El uso de TensorFlow viene motivado, a que a día de hoy, es una de las librerías más populares y mejor mantenidas del momento.

[7] https://keras.io/
[8] https://www.tensorflow.org/

3.4. Algoritmo de Reinforcement Learning

Una vez explicados como funciona el entorno en el que se va a construir el modelo y cómo está definida la estructura de la red neuronal a utilizar, se va a proceder a explicar el papel que juega el RL en el diseño del algoritmo final generado. Como bien se ha explicado en la Sección 1.3.1, existen dos formas distintas de resolver los MDPs: 1) mediante *programación lineal*; 2) mediante *programación dinámica*. En el caso presentado en esta tesis nos encontramos con un MDP, en el que los estados no son conocidos y en el que el espacio de acciones del agente es discreto. Por lo tanto, se ha seguido una aproximación por la cual se ha buscado generar una política de decisión óptima π^* donde $\pi = P(a|s)$ que permita al agente maximizar las recompensas futuras acumuladas. Por ello, se ha hecho uso del algoritmo de aprendizaje *Q-Learning* [Watkins, 1989] que es un algoritmo de control TD basado en una politica offline (*off-policy*).

El objetivo principal de *Q-Learning* es generar una política óptima π^* que genere un agente inteligente capaz de seleccionar las mejores acciones a en cada instante temporal t maximizando la recompensa futura acumulada. Para ello, se busca aproximar la función de acción-valor $Q^*(s,a)$ óptima mediante la fórmula presentada en la Ecuación 3.1 que consiste en la suma de las recompensas r_t aplicándoles un factor de descuento γ en cada paso temporal t.

$$Q^*(s,a) = \max_{\pi} \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ...|s_t = s, a_t = a, \pi] \tag{3.1}$$

Esta ecuación asegura una convergencia siempre y cuando se le aplique un problema basado en una ecuación lineal, lo cual no es el caso de las redes neuronales. Más adelante, se podrá visualizar cómo se resuelve esta problemática. La razón por la cual se ha escogido este algoritmo, en detrimento de otros, es porque el algoritmo *Q-Learning* es más robusto frente a otros algoritmos de RL (como puede ser AC). Además, la probabilidad de convergencia es más alta que otros algoritmos siempre y cuando el espacio de acciones de agente sea discreto. Ésto es debido a que el algoritmo selecciona la acción que maximice la función acción-valor, discriminando el resto de acciones. La razón de esta selección es porque según los postulados de las ecuaciones de Bellman, la política óptima π^* se genera si se satisface la siguiente condición presentada en la Ecuación 3.2, donde la política óptima aparece siempre y cuando se seleccionen aquellas acciones que maximizan la función de valor-acción esperada.

$$Q_{\pi^*}(s,a) = \mathbb{E}_{s'}[r + \gamma \max_{a'} Q^*(s',a')|s,a] \tag{3.2}$$

3.5. Algoritmo Deep Q-Network

La idea básica de los algoritmos de RL, es aproximar la función de acción-valor haciendo uso de la ecuación de Bellman de forma iterativa de tal manera que $Q_i \rightarrow Q^*$. El problema es que en la práctica ésto no es computacionalmente posible. Por ende, lo que se realiza es un cálculo de una función de aproximación que estime dicha función de acción-valor $Q(s,a|\theta) \rightarrow Q^*(s,a)$. Generalmente, dicha aproximación es una función de aproximación lineal, pero en el caso que nos ocupa esto no es posible porque las redes neuronales no proporcionan una función de aproximación lineal. Para solventar esto, Google inventó una función de aproximación neuronal llamada

Q-Network, que ajusta los pesos θ de la red neuronal de forma iterativa para reducir el error medio cuadrático en la ecuación de Bellman.

El objetivo principal del uso de esta función *Q-Network* [Mnih y col., 2015], es la minimización de una función *loss* $L(\theta)$ de la red neuronal que indica cuán buena o mala ha sido la acción tomada por el agente. *Q-Network* se utiliza para calcular la función de la red neuronal de forma iterativa en cada paso temporal t. Esto ajustará los pesos de las neuronas θ para poder incidir sobre la política π de decisión (recordemos que $\pi = P(a|s)$). La Ecuación 3.3 describe la función $L(\theta)$ que busca ser minimizada iterativamente mediante la modificación de los parámetros θ y θ' que representan la estimación actual de *Q-Network* y *Target Network*. Esta última consiste en una copia de *Q-Network*, que se realiza cada t pasos y se utiliza para estimar el *Q-Value* en la siguiente iteración y reducir las correlaciones en las entre las actualizaciones consecutivas de la red.

$$L(\theta) = E \bullet \{ (\overbrace{r + \gamma \max_{a'} Q(s', a'|\theta')}^{\text{target}} - \underbrace{Q(s, a|\theta)}_{\text{output}})^2 \} \tag{3.3}$$

Para mejorar la estabilidad del cálculo de la función $L(\theta)$ se utiliza una lista D llamada *replay memory* que contiene las experiencias pasadas del agente. Cada tupla de dicha lista tiene la forma $e = (s_t, a_t, r_t, s_{t+1}, done)$. La lista D es muestreada aleatoriamente de forma uniforme obteniendo una lista pequeña de experiencias (*minibatch*) que serán usadas de forma encadenada para calcular $L(\theta)$.

Otra mejora llevada a cabo ha sido el cambio del error medio cuadrático por otra función que mejore el cálculo de los errores cometidos por la aproximación presentada en la Ecuación 3.3. Dicho cambio es la utilización del llamado *hubber loss* [Huber, 1964]. *Hubber loss* hace uso del error medio cuadrático cuando los valores de error producidos son muy pequeños y hace uso del error absoluto medio cuando los valores de error son muy grandes. La Ecuación 3.4, presenta la formalización matemática de esta función *loss*, donde a son los residuales entre la diferencia del resultado de la red neuronal y el resultado esperado, esto es $(a = y - f(x))$ o en el caso de esta tesis $(y = \text{target} - \text{output})$. El valor δ un coeficiente que determina el punto de inflexión entre el uso de error medio cuadrático o del error absoluto medio. *Q-Learning* muestra resultados efectivos si los valores resultantes calculados están en el intervalo $[-1, 1]$ por lo tanto, para que *hubber loss* funcione de forma correcta se tiene que utilizar $\delta = 1$.

$$L_\delta(a) = \begin{cases} \frac{1}{2}a^2 & \text{si } |a| \leq \delta, \\ \delta(|a| - \frac{1}{2}\delta), & \text{en otro caso.} \end{cases} \tag{3.4}$$

La razón del uso de esta optimización en contra de la presentada anteriormente en la Ecuación 3.3 es debida a: 1) se necesita una función *loss* que tenga en cuenta tanto los errores grandes como los pequeños, ya que en DQN un error pequeño puede desestabilizar el entrenamiento del agente, el error medio cuadrático se preocupa mucho en minimizar los errores grandes, pero no los pequeños; 2) solventa el problema de hacer un *clipping* de las recompensas entre los intervalos $[-1, 1]$. Para este último punto, es importante saber que sin el uso de esta función *loss* el agente no podría llegar a distinguir si una recompensa es mejor que otra. Por ejemplo, en el caso del juego *SpaceInvaders*, cada nave destruida devuelve al agente una recompensa de 10 puntos. Pero, a veces ocurre que de forma aleatoria aparece una nave especial de color morado, que si el agente destruye, obtendrá una recompensa de

TABLA 3.4: Hiperparámetros utilizados en el algoritmo DQN

Hyperparámetro	Valor	Descripción
Tamaño estado	$[105, 80, 4]$	Tensor de entrada de la red neuronal
Actualización red objetivo C	10000	Frecuencia de actualización de la red objetivo $\hat{Q}$
Memoria de repetición	1000000	Transiciones previas que puede recordar el programa.
Tamaño minibatch	32	Casos de entrenamiento usados para actualizar la red mediante SGD
Ratio de exploración	1000000	Número total de frames en lo que se enfría linealmente ϵ
ϵ_0	0.1	Valor inicial de la exploración $e-greedy$
ϵ_{t-1}	0,0001	Valor final de la exploración $e-greedy$
γ	0.99	Factor descuento utilizado en la actualización *Q-Learning*
Observación	50000	Numero total de frames en los cuales se lanzan acciones estocásticas.
Episodios Máximos	42000	Número total de episodios jugados en la experimentación
Pasos máximos	20000000	Número máximo de timesteps t que se ejecutan en los experimentos
Ratio aprendizaje	0,00025	Ratio aprendizaje de RMSprop.
Momento del gradiente ρ	0.95	Valor del momento de gradiente de RMSprop.
Gradiente cuadrado mínimo ϵ	0.01	Valor del gradiente cuadrado mínimo de RMSprop

500 puntos, siendo sin duda un premio mucho mayor de lo normal y haciendo que la recompensa final mejore sensiblemente. Si no hacemos uso de *hubber loss* el agente no verá diferencia alguna entre destruir una nave normal y la nave especial, por lo que no intentará crear una estrategia por la cual sepa priorizar qué nave debe de destruir en cada momento, ya que considerará que la nave morada vale lo mismo que los demás y adoptará una estrategia más conservadora.

Por último, para optimizar el cálculo de los pesos a partir del resultado obtenido por $L(\theta)$ se ha hecho uso de RMSprop[9] como algoritmo de optimización. Existen otras variantes como ADAM, [Kingma y Ba, 2014] que dan buenos resultados en problemas basados en DRL pero en esta arquitectura y con las optimizaciones mencionadas, RMSprop cumple a la perfección.

3.5.1. Pseudo-código del algoritmo DQN

El Algoritmo 1 detalla el pseudo-código seguido del algoritmo DQN utilizado en la experimentación. El código fuente de dicho algoritmo se puede consultar en **Github** [10]. Algunas líneas de asignación se han omitido para evitar hacer demasiado confuso el algoritmo.

3.5.2. Tabla de hiperparámetros utilizado en el algoritmo DQN

La Tabla 3.4 contiene los hiperparámetros utilizados en el algoritmo DQN explicado en la Sección 3.5.1. Los parámetros aquí mostrados han sido utilizados en todos los experimentos llevados a cabo, sin realizar alteraciones para intentar ajustar y optimizar los aprendizajes en cada juego. Es importante observar los valores de ϵ_0 y ϵ_{t-1} que son de 0,1 y 0,0001 respectivamente. Esto indica un factor de confianza alto durante la fase de exploración haciendo que $\epsilon - greedy$ no incida en exceso a la hora de aleatorizar las decisiones que tiene que tomar el agente. La razón de dotar un factor de confianza alto en el proceso de exploración es debido a que, si el agente tiene un conocimiento previo de un entorno anterior, debería ser capaz de desenvolverse perfectamente ya desde las primeras fases. Si el factor de confianza es muy bajo, es muy probable que el algoritmo $\epsilon - greedy$ perturbe completamente el conocimiento previo, invalidando completamente la hipótesis que se desea presentar.

[9] http://www.cs.toronto.edu/ tijmen/csc321/slides/lecture_slides_lec6.pdf
[10] https://github.com/rubenmulero/generalization_atari_dqn

Algorithm 1 Algoritmo DQN implementado

procedure DQN
  $env \leftarrow$ instanciación del entorno a ejecutar
  action_size $\leftarrow$ espacio total de acciones disponibles en el entorno
  $Q \leftarrow$ inicialización de la red neuronal con pesos aleatorios
  $\hat{Q} \leftarrow Q$ ▷ El target de la función de valor para calcular el SGD
  $D \leftarrow$ inicialización de la memoria de repetición de tamañoN
  $T \leftarrow$ pasos máximos del experimento
  $E \leftarrow$ episodios máximos del experimento
  $C \leftarrow$ pasos para actualizar la red neuronal objetivo
  $t \leftarrow 0$
  for episodio $= 0, E$ **do**
    $\varphi_t \leftarrow env.reset()$ ▷ tamaño: $210x160x3$ en RGB
    $x_t \leftarrow$ preporcesar_imagen (φ_t) ▷ tamaño: $105x80x1$ en B/W
    if episodio $== 0$ **then** ▷ El estado es 4 veces x_t
      $s_t \leftarrow [x_t, x_t, x_t, x_t]$ ▷ tamaño: $105x80x4$
    else ▷ El estado se genera añadiendo la nueva imagen
      $s_t \leftarrow [x_{t-3}, x_{t-2}, x_{t-1}, x_t]$ ▷ tamaño: $105x80x4$
    while *true* **do**
      if $rand <= \epsilon$ **then**
        $a_t \leftarrow$ *acción aleatoria*
      else
        $a_t \leftarrow \arg\max_a Q(s_t, a|\theta)$
      $\varphi_{t+1}, r_t, done \leftarrow env.step(a_t)$ ▷ tamaño: $210x160x3$ en RGB
      $x_{t+1} \leftarrow$ preprocesar_imagen (φ_{t+1}) ▷ tamaño: $105x80x1$ en B/W
      $s_{t+1} \leftarrow [x_{t-2}, x_{t-1}, x_t, x_{t+1}]$ ▷ tamaño: $105x80x4$
      $D \leftarrow [s_t, a_t, r_t, s_{t+1}, done]$
      if $t >$ obsevación **then**
        minibach$(s_j, a_j, r_j, s_{j+1}) \leftarrow$ muestreo aleatorio de D
        $y_j = \begin{cases} r_j & \text{si el episodio terminal en el paso } j+1 \\ r_j + \gamma \, \text{máx}_{a'} \, \hat{Q}(s_{j+1}, a'|\theta') & \text{en otro caso} \end{cases}$
        Ejecutar SGD $\begin{cases} \frac{1}{2}(y_j - Q(s_j, a_j|\theta))^2 & \text{si } |(y_j - Q(s_j, a_j|\theta)| \leq 1, \\ (|(y_j - Q(s_j, a_j|\theta)| - \frac{1}{2}), & \text{en otro caso.} \end{cases}$
        $\epsilon \leftarrow \epsilon -$ parametro de reducción ▷ reducción del valor de ϵ
      $s_t \leftarrow s_{t+1}$
      $t \leftarrow t + 1$
      if t mód $C == 0$ **then**
        $\hat{Q} \leftarrow Q$ ▷ Actualizamos el target
      if $t >= T$ **then**
        $raisefinalexperimento$

3.5.3. Funcionamiento del algoritmo DQN

El algoritmo DQN consiste en un proceso iterativo para crear una política π_* que permita generar un agente inteligente mediante el uso de una red neuronal. Durante las primeras fases de la experimentación, el algoritmo instancia una serie de hiperparámetros que serán utilizados en toda la vida del experimento. Cuando se asigna la variable *env* lo que se hace es instanciar el método *.make()* de *OpenAIGym* para definir qué juego queremos usar para el experimento. Con esta instanciación se ejecuta en segundo plano el emulador y se realizan las conexiones necesarias para que el programa pueda enviar las acciones pertinentes y recibir imágenes y recompensas.

Cuando se ejecuta *env.reset()* el emulador devuelve un tensor de $\varphi_t = 210x160x3$ que representa una imagen del juego inicializado. Dicha imagen es preprocesada reduciéndola a un tensor de $x_t = 105x80x1$ que consiste en una imagen reducida en blanco y negro (1 canal con valores normalizados $[0, 1]$). La Figura 3.4 contiene las diferencias entre una imagen original y una imagen preprocesada. Esta operación se lleva a cabo para reducir costes computacionales, ya que procesar tantas imágenes a pleno color requiere de mucha memoria, tanto gráfica como aleatoria.

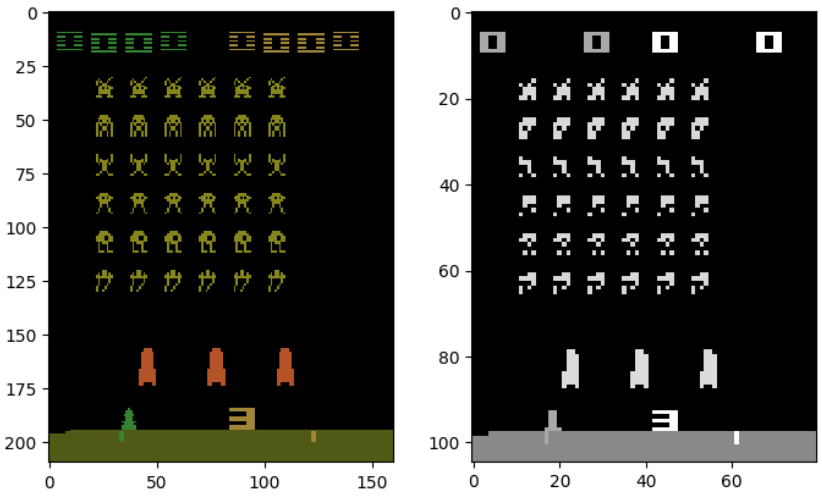

FIGURA 3.4: Preprocesado de imagenes; (a) izquierda: imagen original; (b) derecha: imagen transformada

Cuando la imagen es preprocesada, se genera el estado s_t apilando un total de cuatro imágenes preprocesadas, que forman un tensor de $105x80x4$ donde 4 contiene valores normalizados entre $[0, 1]$. La razón por la cual se apilan las imágenes es porque de esta forma, la CNN es capaz de "ver" la transición existente en un estado concreto. La generación del primer estado de la experimentación s_0 se hace mediante el uso de $4 \cdot x_t$ iguales, más adelante cada estado s_t será formado por una nueva imagen procesada x_{t+1} que se ira añadiendo al estado por la derecha, eliminando la ultima de ellas por la izquierda[11].

Una vez que el estado está formado, el algoritmo selecciona la acción a_t a realizar. En esta parte se utiliza $\epsilon - greedy$ para seleccionar una acción aleatoria entre el espacio de acciones disponibles u obtener una acción a través del resultado obtenido por la red neuronal dado el estado s_t que hemos creado antes. La comparativa se realiza haciendo un *random* y comparándolo con el valor actual que tenga la variable ϵ. La salida de la red neuronal produce un vector de probabilidades del tamaño igual al

[11] https://pythontic.com/containers/deque/popleft

espacio de acciones del entorno; usando el arg max se selecciona de forma voraz la mejor acción disponible. Al final de este proceso, a_t contendrá un número entero que definirá que acción tomar (ej: 0 izquierda, 1 derecha). El valor que tiene que tomar ϵ tiene que ser un número entre 0 y 1 siendo un número más bajo, una situación más optimista para el algoritmo.

Con la acción ya obtenida, se ejecuta el método *env.step(a_t)* que envía la acción al emulador. Este método devuelve el resultado de dicha acción mediante una tupla que contiene,una nueva imagen φ_{t+1}, una recompensa asociada a la acción realizada r_t y un valor booleano que indica si el agente se encuentra en estado terminal o no (por tanto si ha perdido o no en el episodio actual).

En cada iteración t las acciones escogidas y los resultados obtenidos al ejecutarlas en el emulador se guardan en el *replay memory D* que se irá convirtiendo poco a poco en el set de entrenamiento para la red neuronal. Dentro de la lista D se guarda la tupla que contienen los valores $(s_t, a_t, r_t, s_{t+1}, done)$ y que más adelante serán muestreados de manera aleatoria para ejecutar el SGD. Para mejorar el muestreo se establece una variable llamada *observación* que evita que durante una serie de t pasos se pueda ejecutar el SGD. Esta variable cumple dos funciones importantes: 1) mejora la aleatoriedad del muestreo evitando variables repetidas o seguidas; 2) asegura que la lista D contenga las suficientes experiencias guardadas para extraer de ella un muestreo de calidad y decorrelado.

En este punto, el sistema extrae una muestra de experiencias pasadas de la lista D y las guarda en la variable *minibach* de tamaño j. Lo que hace el código a partir de esa extracción es computar el valor y_j por cada elemento j del *minibach* teniendo en cuenta dos situaciones: 1) si el valor booleano $done_j$ tiene un valor *true* entonces se asigna directamente r_j sin hacer más operaciones porque se entiende que la partida ha terminado y no hay necesidad de calcular el *future discounted reward*: 2) si el valor booleano $done_j$ tiene un valor de $false$ entonces se calculará el *target* indicado en la Ecuación 3.3. Este valor calculado y_j representará el valor que "supuestamente" debería devolver la red neuronal para ese estado concreto (como una muestra de validación en el aprendizaje supervisado) y se utilizará en el siguiente paso para calcular el SGD. Nótese que en este paso se hace uso de la copia de la red $\hat{Q}$. Esto se hace para evitar que la red intente predecir su propia predicción y se pueda mejorar los resultados finales obtenidos.

Cuando se han computado todos los y_j, se realiza el SGD, pasando a la red los estados y acciones (s_j, a_j) y los cálculos de y_j. De esta manera, la red podrá ajustar sus pesos internos y mejorar los resultados en las siguientes iteraciones. Resumidamente, se está realizando una operación similar al aprendizaje supervisado, donde el conjunto de entrenamiento son los estados y las acciones y el conjunto de validación son los y_j que han sido calculados mediante una aproximación obtenida del *Q-Learning*. Ésto le dirá a la red *"cuánta recompensa creo que puedes obtener con esta acción en este estado"* y le asignara una posible puntuación futura que intentará mejorar en las sucesivas iteraciones.

Capítulo 4

Proceso experimental

4.1. Introducción

En este capítulo se va a presentar el proceso experimental seguido para formalizar la hipótesis nula H_0 postulada en la Sección 1.5. Se van a exponer los pasos que se van a seguir a la hora de su validación, indicando los objetivos necesarios para cumplir los subjetivos expuestos.

Este capítulo está dividido en las siguientes secciones: 1) Métricas a estudiar: explica cuáles han sido las métricas consideradas para estudiar los resultados obtenidos en la experimentación; 2) Fases experimentación: describe qué se ha realizado en la experimentación y en qué fases se ha dividido; 3) Plataforma experimentación: muestra la configuración hardware y software utilizadas en la experimentación para hacerse una idea general de qué tipo de máquina y software ha sido utilizada para llevar a cabo la experimentación; 4) Representación de los resultados: describe cómo van a ser representados los datos y métricas obtenidas en la experimentación realizada.

4.2. Métricas a estudiar

Para validar los sub-objetivos planteados en H_0, la experimentación busca realizar un estudio sobre ciertas métricas que dejaran patente si el conocimiento transferido de una experiencia anterior está ayudando o no a mejorar los entrenamientos sucesivos de nuevos agentes en distintos entornos de aprendizaje. Por lo tanto, en la fase experimental se ha buscado estudiar lo siguiente:

1. **Analizar las recompensas obtenidas:** si las recompensas medias tanto en la fase de exploración como en la de entrenamiento completo son mayores, entonces esto significa que el conocimiento previo transferido al agente está ayudando a una convergencia más rápida. Esto demuestra, que el agente no empieza desde un punto aleatorio porque el conocimiento transferido está ayudando a encontrar una estrategia óptima desde un principio.

2. **Analizar la recompensa acumulada:** si la recompensa total obtenida por el agente es mayor, significa que en la mayoría de las partidas, está consiguiendo de media mejores recompensas, por lo que se puede decir que el conocimiento transferido está ayudando al agente a conseguir mejores resultados.

3. **Analizar la duración de los episodios:** si la duración de los episodios tanto en la fase de exploración como en la de entrenamiento completo son mayores, entonces se puede afirmar que el conocimiento transferido está ayudando al agente a mejorar sus resultados. Esto es debido a que el agente puede no obtener las mejores recompensas, pero al durar mas tiempo los episodios, el agente

TABLA 4.1: Métricas registradas en el proceso experimental.

Nombre métrica	Descripción
Timesteps	Número total de pasos de la experimentación
Episodes	Número de episodios que ha conseguido jugar el agente
Max Reward (**MR**)	Recompensa máxima obtenida en un episodio
Total Reward Exploration (**TRE**)	Recompensa total acumulada durante la fase de exploración
Mean Episode Reward (**MER**)	Media de recompensas obtenida en exploración
Mean Episode Duration Exploration (**MEDE**)	Tiempo medio duración episodios en fase exploración
Mean Episode Duration (**MED**)	Tiempo medio duración episodios
Cumulative Reward (**CR**)	Recompensa acumulada total

está siendo capaz de aprender a **no perder**, lo cual quiere decir que el conocimiento transferido le está ayudando a desarrollar una estrategia que al menos en un principio, le permite partir de un punto desde el cual tarda más tiempo en llegar a un estado terminal dentro de un episodio.

Durante el proceso experimental, el agente entrenado pasa por tres fases: *observe, exploration y training*. Se entiende como *observe* el proceso por el cual el agente lanza acciones aleatorias sin realizar entrenamiento alguno. Se entiende como *exploration* el proceso por el cual el agente aplica la política $e - greedy$. Se entiende como *training* el proceso por el cual el agente entrena el modelo. El punto 1 tiene en cuenta la fase de exploration y training, el 2 las tres fases completas, el 3 la fase exploration y training.

La Tabla 4.1 expone las métricas que van a ser recogidas durante la ejecución del proceso experimental. Las métricas recogidas en **Timesteps** y **Episodes** representan el número total de pasos y episodios jugados en todas las fases de la experimentación (observación, exploración y entrenamiento). Las métricas **Total Reward Exploration** y **Mean Episode Duration Exploration** son recogidas únicamente en la fase de exploración. Por último, las métricas **Mean Episode Reward**, **Mean Episode duration** y **Cumulative Reward** son recogidas en la fase de exploración y entrenamiento. La razón detrás de esto es la siguiente: 1) las dos primeras métricas representan la duración del experimento, cuántos pasos temporales se han dado y cuántos episodios ha conseguido jugar el agente; 2) las métricas de exploración se centran en saber si el agente ha mejorado o no en el proceso de exploración de una estrategia a entrenar; 3) las métricas que representan la exploración más el entrenamiento, se centran en saber si el agente ha mejorado en todo el proceso experimental válido. La razón por la que no se incluye la observación (*observe*) es porque el proceso de observación busca crear una memoria inicial mediante la selección de acciones puramente estocásticas y no entrena la red neuronal. La red empieza a ser entrenada cuando el proceso experimental entra en la fase de exploración.

4.3. Fases de la experimentación

Para demostrar si existe o no transferencia de conocimiento entre los distintos video-juegos, se ha dividido la fase experimental en dos partes. La primera, llamada *Fase de experimentación 1*, busca comparar dos video-juegos similares. Estos juegos tienen un objetivo similar, que es la destrucción de unos enemigos en un espacio dimensional parecido (misma región de pantalla), un espacio de acciones similar y por último, un sistema similar de recompensas. La segunda, llamada *Fase de experimentación 2*, busca comparar los dos juegos utilizados en en la *Fase de experimentación*

TABLA 4.2: Proceso experimentación 1. **Task1** = *SpaceInvaders*, **Task2** = *DemonAttack*

ID	Game	Feature Extractor	Action Selector
SI.1	SpaceInvaders	Pre-trained Task2	Random Init
SI.2	SpaceInvaders	Random Init	Pre-trained Task2
SI.3	SpaceInvaders	Pre-trained Task2	Pre-trained Task2
DA.1	DemonAttack	Pre-trained Task1	Random Init
DA.2	DemonAttack	Random Init	Pre-trained Task1
DA.3	DemonAttack	Pre-trained Task1	Pre-trained Task1

1 y compararla con otro video-juego que tiene un espacio de acciones similar pero tiene un objetivo y un sistema de movimiento completamente diferentes.

4.3.1. Fase de experimentación 1

En esta fase se ha llevado a cabo la comparación de la transferencia de conocimiento entre los video-juegos *SpaceInvaders* y *DemonAttack*. El objetivo primordial de esta experimentación ha sido demostrar los sub-objetivos SO1 y SO2. Para llevarla a cabo, lo que se ha realizado es el entrenamiento de un agente inteligente para cada juego sin partir de conocimiento previo alguno. Con esto, se ha conseguido generar un agente inteligente de control para saber cuánto conocimiento se puede transferir o no. Una vez entrenado el agente, que llamaremos **agente de control (AC.SI y AC.DA)**, se ha llevado a cabo el proceso experimental por el cual se ha validado si existe o no transferencia de conocimiento, intercambiando los pesos entre los agentes de control de cada juego y cargando parte o todos los pesos que han generado los entrenamientos de las redes neuronales.

La Tabla 4.2 detalla el proceso experimental seguido. La columna *ID* contiene un nombre identificativo del experimento a realizar. La columna *Game* indica qué entorno de aprendizaje se ha utilizado en la experimentación. La columna *Feature Extractor* indica si se ha hecho uso o no los pesos de las CNN para el entrenamiento del agente. Por último, la columna *Action Selector* indica si se ha hecho uso o no los pesos de las redes Densas para el entrenamiento del agente. Como se puede visualizar, las celdas con valores llamados **random init** significan que no se ha usado el conocimiento previo del otro video-juego y que esa parte de la red se ha ejecutado con pesos aleatorios. Las celdas con valores **Pre-trained Task X** representan el uso de parte de los pesos del Task X donde *X* es el número que representa el origen de los pesos utilizados.

En total se han hecho por cada video-juego 3 experimentaciones: 1) usar los pesos de la CNN de otro juego; 2) usar los pesos de las redes Densas de otro juego; 3) usar todos los pesos del otro juego. De esta forma se podrá visualizar qué cambios se producen o no durante el aprendizaje.

4.3.2. Fase de experimentación 2

En esta fase se ha llevado la comparación de la transferencia de conocimiento entre los video-juegos *SpaceInvaders*, *DemonAttack* sobre *Qbert*. El objetivo primordial de esta experimentación ha sido demostrar los sub-objetivos SO3 y SO3. Para llevarla cabo, se ha realizado primero el entrenamiento de un **agente control (AC.Qb)**

TABLA 4.3: Proceso experimentación 2. **Task1** = *SpaceInvaders*, **Task2** = *DemonAttack*

ID	Game	Feature Extractor	Action Selector
Qb.1	Qbert	Pre-trained Task2	Random Init
Qb.2	Qbert	Random Init	Pre-trained Task2
Qb.3	Qbert	Pre-trained Task2	Pre-trained Task2
Qb.4	Qbert	Pre-trained Task1	Random Init
Qb.5	Qbert	Random Init	Pre-trained Task1
Qb.6	Qbert	Pre-trained Task1	Pre-trained Task1

TABLA 4.4: Especificaciones hardware de las plataformas de experimentación.

Nombre	CPU	Memory	HDD	Nvidia Card
Máquina 1	Intel i5-6600K @ 3.50GHz	64Gb	Seagate ST4000NM0033 7200RPM 4T	GeForce GTX 1080
Máquina 2	Intel i7-6700K @ 4.00GHz	64Gb	Seagate ST8000VN0022 7200RPM 8T	GeForce GTX 1080

para el juego *Qbert* sin hacer uso de experiencia previa alguna. A continuación se ha hecho uso de la experiencia original previa de los juegos *SpaceInvaders* y *DemonAttack* (agentes de control) para saber si el juego *Qbert* es capaz de aprovechar o no la transferencia adquirida.

La Tabla 4.3 detalla el proceso experimental seguido, que consta de los mismos campos que en el experimento anterior. En total se han hecho los mismos experimentos que la fase experimental anterior.

4.4. Plataforma de experimentación

Para realizar las experimentaciones propuestas, se va hacer uso de dos máquinas diferentes preparadas para desarrollar y entrenar aplicaciones basadas en DRL. La Tabla 4.4 contiene las especificaciones hardware de las plataformas utilizadas en la experimentación. La Tabla 4.5 contiene las especificaciones software que indican la versión y el nombre del software utilizado en la experimentación.

La Máquina 1 ha sido utilizada para desarrollar el agente de control del juego *DemonAttack* y usar su conocimiento guardado para lanzar las experimentaciones con los otros juegos. La Máquina 2 ha sido utilizada para desarrollar el agente de control del juego *SpaceInvaders* y usar su conocimiento guardado para lanzar las experimentaciones con los otros juegos. Por concluir, ambas máquinas se han usado para entrenar el agente de control del juego *Qbert*. Esto último se ha realizado para validar que los resultados obtenidos más adelante sean mejores o peores.

TABLA 4.5: Especificaciones software de las plataformas de experimentación.

Nombre	Python version	Keras version	TensorFlow version	Nvidia Driver version	CUDA version
Máquina 1	2.7.12	2.1.3	1.4.1	375.66	8.0.44
Máquina 2	2.7.12	2.1.3	1.4.1	375.26	8.0.61

4.5. Representación de los resultados

Una vez ejecutados los experimentos, los resultados son mostrados de dos maneras diferentes: 1) mediante el uso de gráficas que muestren las variaciones de los resultados obtenidos entre los distintos entrenamientos; 2) mediante el uso de tablas estadísticas que muestren las diferencias numéricas entre los distintos resultados obtenidos. Con esta representación se podrá discutir si el conocimiento utilizado por las experiencias de otros juegos mejora o no los resultados obtenidos y se podrá aceptar o rechazar la hipótesis nula H_0 presentada.

Capítulo 5

Resultados experimentales

5.1. Introducción

En este capítulo se van a mostrar los resultados obtenidos en el proceso experimental definido en el Capítulo 4. Dichos resultados validarán o rechazarán la hipótesis formal H_0 presentada y abrirán una serie de líneas de discusión para valorar los resultados obtenidos. Este capítulo estará dividido en las secciones: 1) Resultados experimentales: fase experimentación 1; 2) Resultados experimentales: fase experimentación 2; 3) Discusión de los resultados obtenidos.

5.2. Resultados experimentales: fase experimentación 1

En la primera fase de experimentación se ha probado si existe transferencia de conocimiento entre el juego *SpaceInvaders* y *DemonAttack*. La Sección 4.3.1 contiene información más detallada del proceso experimental seguido y la Tabla 4.2 contiene información detallada sobre las métricas obtenidas.

5.2.1. Métricas obtenidas

La Tabla 5.1 contiene los resultados experimentales obtenidos en ambos juegos implicados.

5.2.2. Gráficos obtenidos

En las siguientes lineas se van a mostrar los gráficos obtenidos a partir de las métricas recogidas en la Tabla 5.1. Para representar mejor los datos en las gráficas, se ha considerado mostrar los resultados cada 50 Episodios completos o *epoch*, donde 1 $epoch = 50$ episodios. La elección de la representación basada en *epoch*, se considera debido a que los resultados obtenidos en cada episodio no son deterministas y por lo tanto, haciendo una media de dichos resultados se consigue una mejor interpretación del proceso de aprendizaje del agente. La leyenda de los gráficos contiene los siguientes valores: 1) **Original**: valores originales obtenidos con todos los pesos aleatorios (el descrito **agente de control**); 2) **CNN**: valores obtenidos haciendo uso de los pesos de las CNN del otro juego; 3) **Dense**: valores obtenidos haciendo uso de los pesos de las Densas del otro juego; 4) **Full**: valores obtenidos haciendo uso completo de los pesos del otro juego (CNN y Densas).

Gráficas para experimentaciones SI.1, SI.2, SI.3

Las siguientes gráficas muestran los resultados de las experimentaciones **SI.1**, **SI.2** y **SI.3** que representan el entrenamiento de un agente inteligente que sepa

TABLA 5.1: Métricas registradas en el proceso experimental 1. **AC.SI** = Agente de Control Space Invaders. **AC.DA** = Agente de Control Demon Attack.

ID	Timesteps	Episodes	MR [a]	TRE [b]	MER [c]	MEDE [d]	MED [e]	CR [f]
SI.1	2e+7	21663	1205	236740	297,3077	735,7873	923,9382	6419765
SI.2	2e+7	21370	1210	243070	326,3414	751,9632	936,1881	6954335
SI.3	2e+7	**21914**	1245	229455	254,9030	744,3162	913,0189	5569885
AC.SI	2e+7	19140	**1490**	**269775**	**410,2084**	**835,2264**	**1045,936**	**7824315**
DA.1	2e+7	18412	5130	**163220**	524,6326	1190,4270	1086,2480	9659535
DA.2	2e+7	12365	4770	102665	**789,3449**	1737,1990	**1617,4690**	9760250
DA.3	2e+7	16619	5080	84510	524,1651	**2929,9080**	1203,4420	8711100
AC.DA	2e+7	16963	**6900**	136800	695,0035	1050,94	1176,361	**11788650**

[a] **M**ax **R**eward
[b] **T**otal **R**eward **E**xploration
[c] **M**ean **E**pisode **R**eward
[d] **M**ean **E**pisode **D**uration **E**xploration
[e] **M**ean **E**pisode **D**uration
[f] **C**umulative **R**eward

aprender a jugar al juego *SpaceInvaders* haciendo uso del conocimiento guardado del juego *DemonAttack*

La Figura 5.1, muestra la media de las recompensas obtenidas en cada *epoch*. El eje de abscisas muestra los *epoch* y el eje de ordenadas muestra las recompensas medias por cada *epoch* (media de 50 episodios). A más recompensa media obtenida, mejor estrategia ha aprendido el agente.

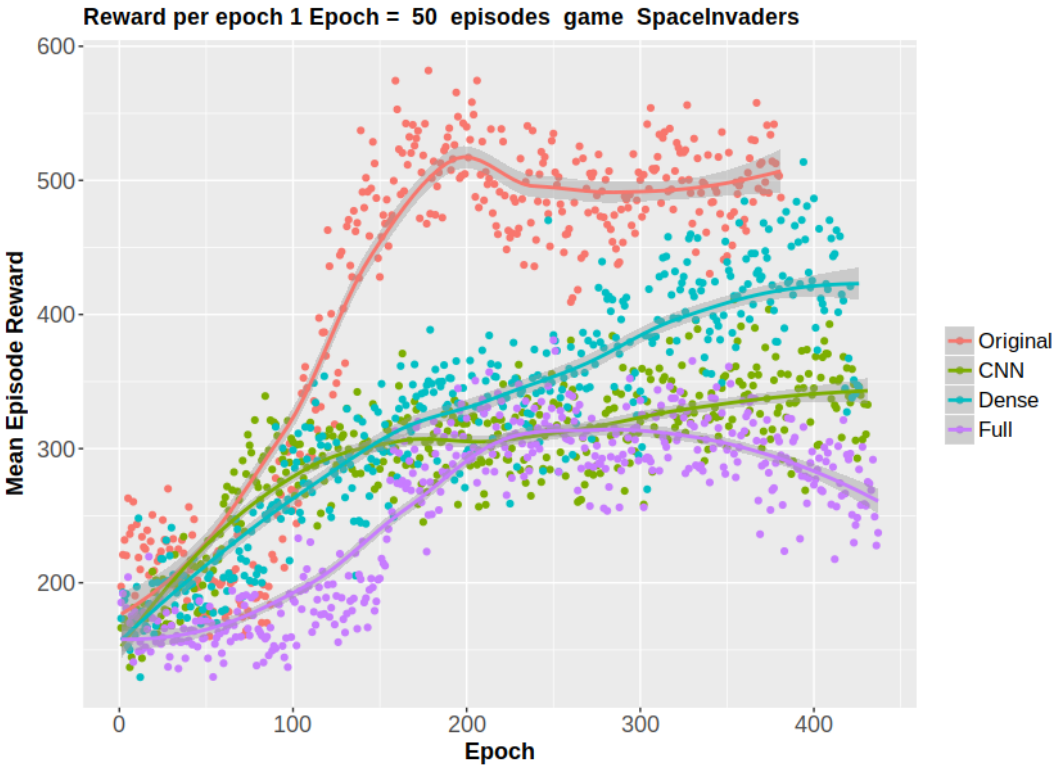

FIGURA 5.1: Recompensas medias obtenidas en los episodios de las experimentaciones SI.1, SI.2 y SI.3

La Figura 5.2, contiene los pasos temporales medios en cada episodio. El eje de abscisas muestra los *epoch* y el eje de ordenadas muestra los pasos temporales medios por cada *epoch*. A más pasos temporales, mejor estrategia ha aprendido el agente ya que está aprendiendo a no perder dentro del juego.

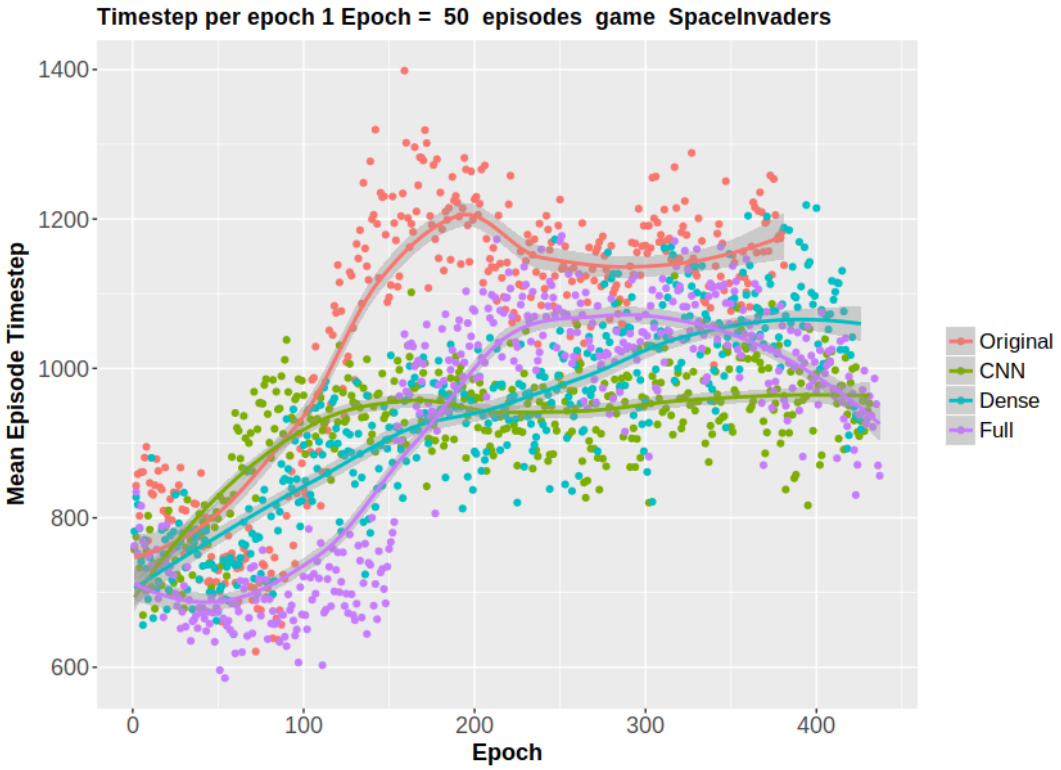

FIGURA 5.2: Pasos medios en los episodios de las experimentaciones SI.1, SI.2 y SI.3

La Figura 5.3, contiene las recompensas acumuladas en todo el proceso experimental. El eje abscisas muestra los Episodios jugados y el eje de ordenadas muestra la recompensa acumulada en cada Episodio jugado. A más recompensa acumulada, mejores resultados ha obtenido el agente en los Episodios jugados y por lo tanto, mejor estrategia ha sido capaz de aprender para jugar.

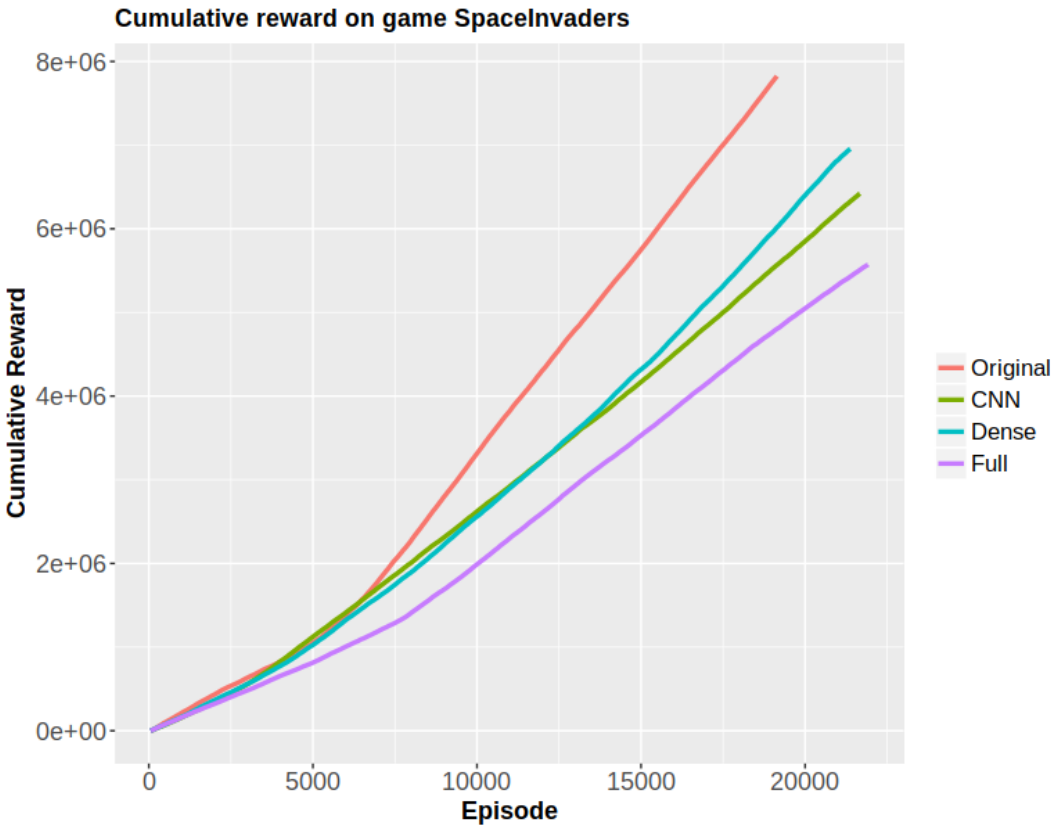

FIGURA 5.3: Recompensa acumulada de las experimentaciones SI.1, SI.2 y SI.3

Gráficas para experimentaciones DA.1, DA.2, DA.3

Las siguientes gráficas muestran los resultados de las experimentaciones **DA.1**, **DA.2** y **DA.3** que representan el entrenamiento de un agente inteligente que sepa aprender a jugar al juego *DemonAttack* haciendo uso del conocimiento guardado del juego *SpaceInvaders*.

La Figura 5.4, muestra la media de las recompensas obtenidos en cada *epoch*. El eje de abscisas muestra los *epoch* y el eje de ordenadas muestra las recompensas medias por cada *epoch* (media de 50 episodios). A más recompensa media obtenida, mejor estrategia ha aprendido el agente.

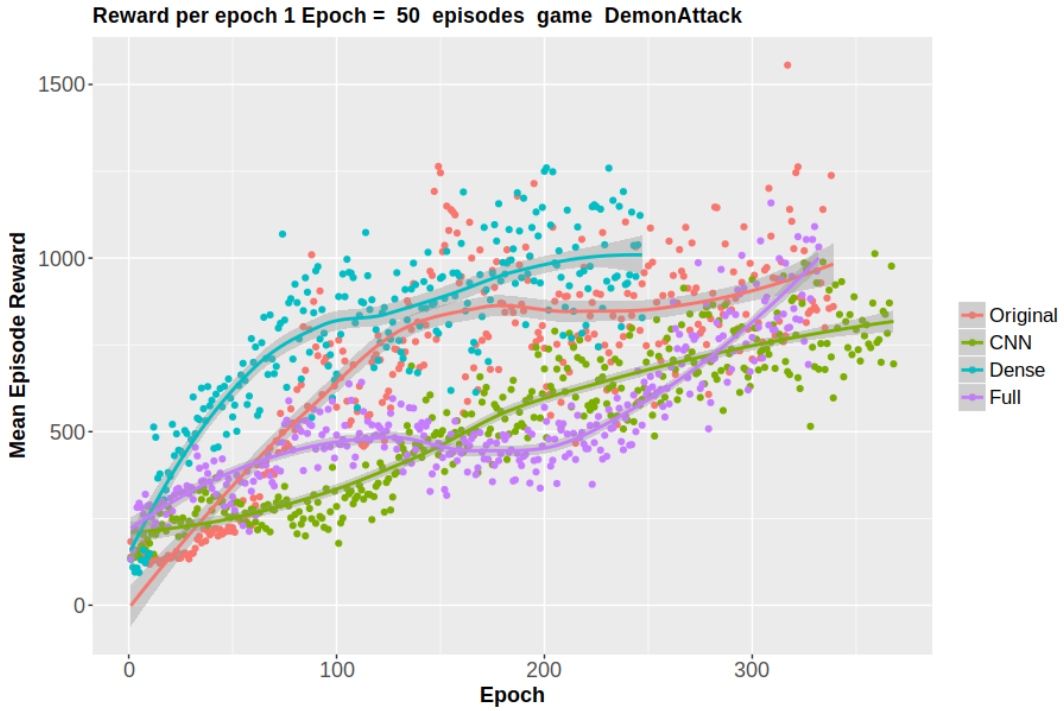

FIGURA 5.4: Recompensas medias obtenidas en los episodios de las experimentaciones DA.1, DA.2 y DA.3

La Figura 5.5, contiene los pasos temporales medios en cada episodio. El eje de abscisas muestra los *epoch* y el eje de ordenadas muestra los pasos temporales medios por cada *epoch*. A más pasos temporales, mejor estrategia ha aprendido el agente ya que está aprendiendo a no perder dentro del juego.

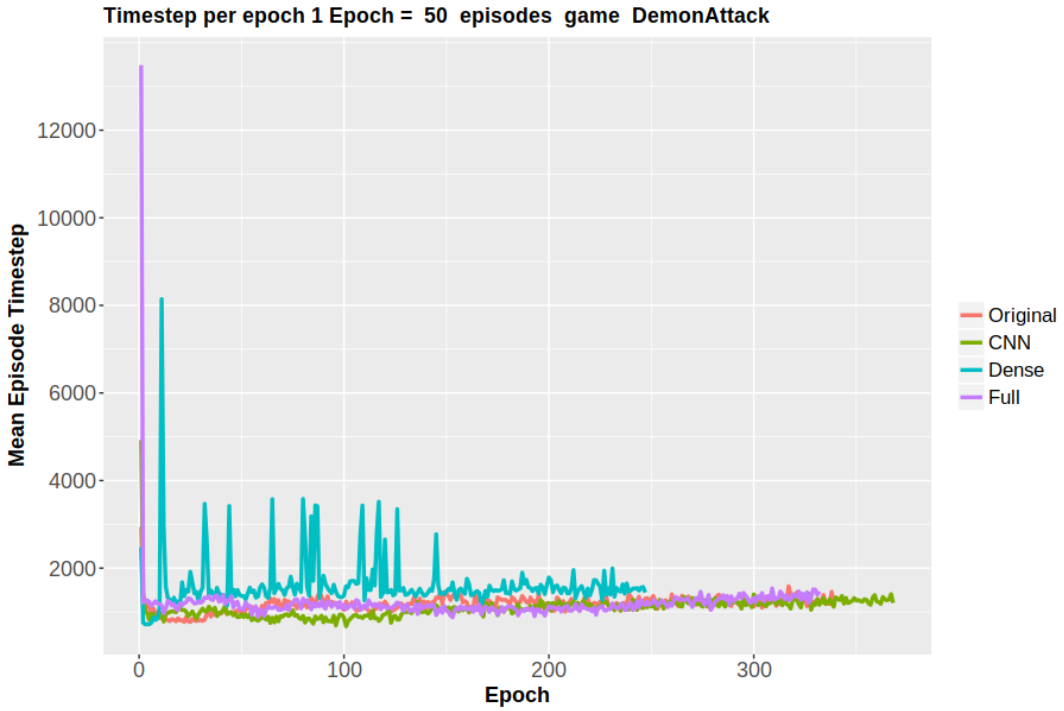

FIGURA 5.5: Pasos medios en los episodios de las experimentaciones DA.1, DA.2 y DA.3

La Figura 5.6, contiene las recompensas acumuladas en todo el proceso experimental. El eje de abscisas muestra los Episodios jugados y el eje de ordenadas muestra la recompensa acumulada en cada Episodio jugado. A más recompensa acumulada, mejores resultados ha obtenido el agente en los Episodios jugados y por lo tanto, mejor estrategia ha sido capaz de aprender para jugar.

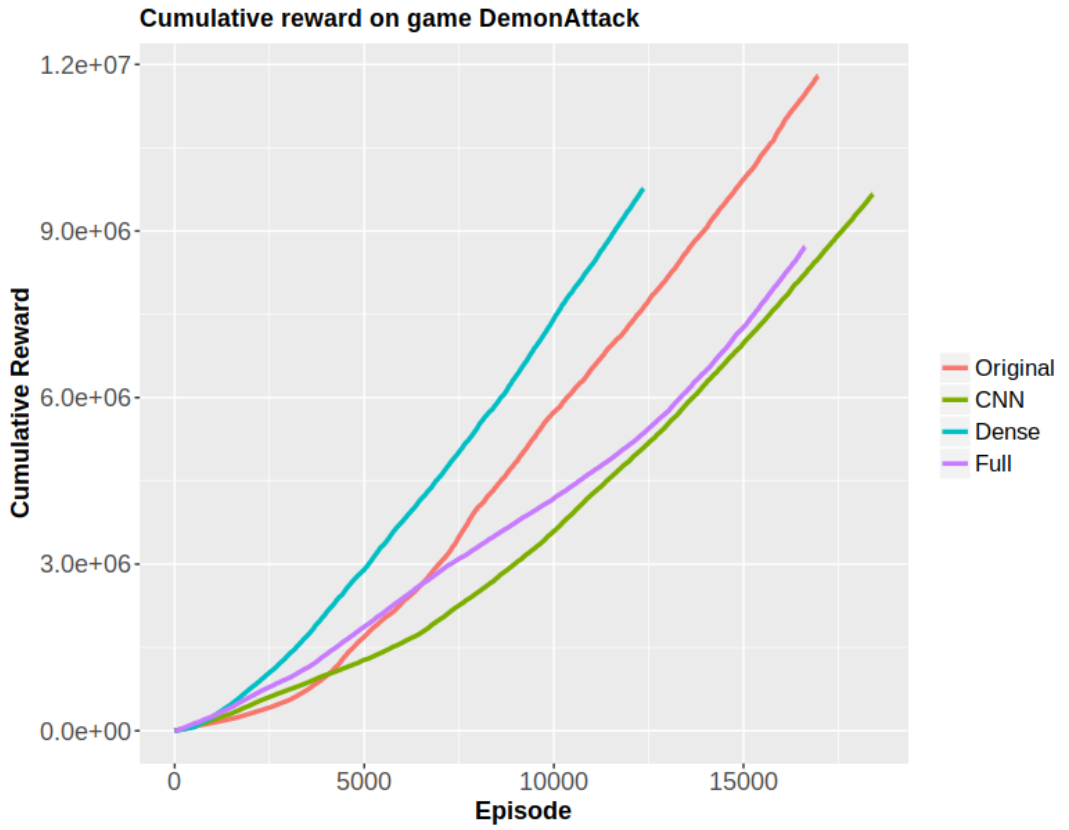

FIGURA 5.6: Recompensa acumulada de las experimentaciones DA.1, DA.2 y DA.3

5.3. Resultados experimentales: fase experimentación 2

5.3.1. Métricas obtenidas

La Tabla 5.2 contiene los resultados obtenidos en ambos juegos implicados.

TABLA 5.2: Métricas registradas en el proceso experimental 2. **AC.Qb** = Agente de Control Qbert.

ID	Timesteps	Episodes	MR [a]	TRE [b]	MER [c]	MEDE [d]	MED [e]	CR [f]
Qb.1	2e+7	25810	5275	863825	902,6799	497,662	777,0843	23174500
Qb.2	2e+7	30958	5075	782150	866,6756	481,3293	646,6453	26738675
Qb.3	2e+7	**31302**	5500	791125	1061,845	440,0079	640,1227	33093450
Qb.4	2e+7	30657	5375	699900	795,7163	484,0552	654,4935	24255025
Qb.5	2e+7	22189	**8775**	**1031725**	**3283,325**	453,8374	903,4926	**72499100**
Qb.6	2e+7	29223	5700	668200	1010,744	402,9508	687,2207	29341900
AC.Qb	2e+7	21848	5875	570450	990,1488	**505,6555**	**917,419**	21531775

[a] **M**ax **R**eward
[b] **T**otal **R**eward **E**xploration
[c] **M**ean **E**pisode **R**eward
[d] **M**ean **E**pisode **D**uration **E**xploration
[e] **M**ean **E**pisode **D**uration
[f] **C**umulative **R**eward

5.3.2. Gráficos de los resultados obtenidos

Siguiendo el mismo proceso que la Fase Experimental 1, se va a proceder a mostrar las gráficas obtenidas a partir de las métricas de la Tabla 5.2. Las consideraciones y leyendas aplicadas anteriormente son las mismas para los resultados aquí mostrados.

Gráficas para experimentaciones Qb.1, Qb.2, Qb.3

Las siguientes gráficas muestran los resultados de las experimentaciones **Qb.1**, **Qb.2** y **Qb.3** que representan el entrenamiento de un agente inteligente que sepa aprender a jugar al juego *Qbert* haciendo uso del conocimiento guardado del juego *DemonAttack*.

La Figura 5.7, muestra la media de las recompensas obtenidos en cada *epoch*. El eje de abscisas muestra los *epoch* y el eje de ordenadas muestra las recompensas medias por cada *epoch* (media de 50 episodios). A más recompensa media obtenida, mejor estrategia ha aprendido el agente.

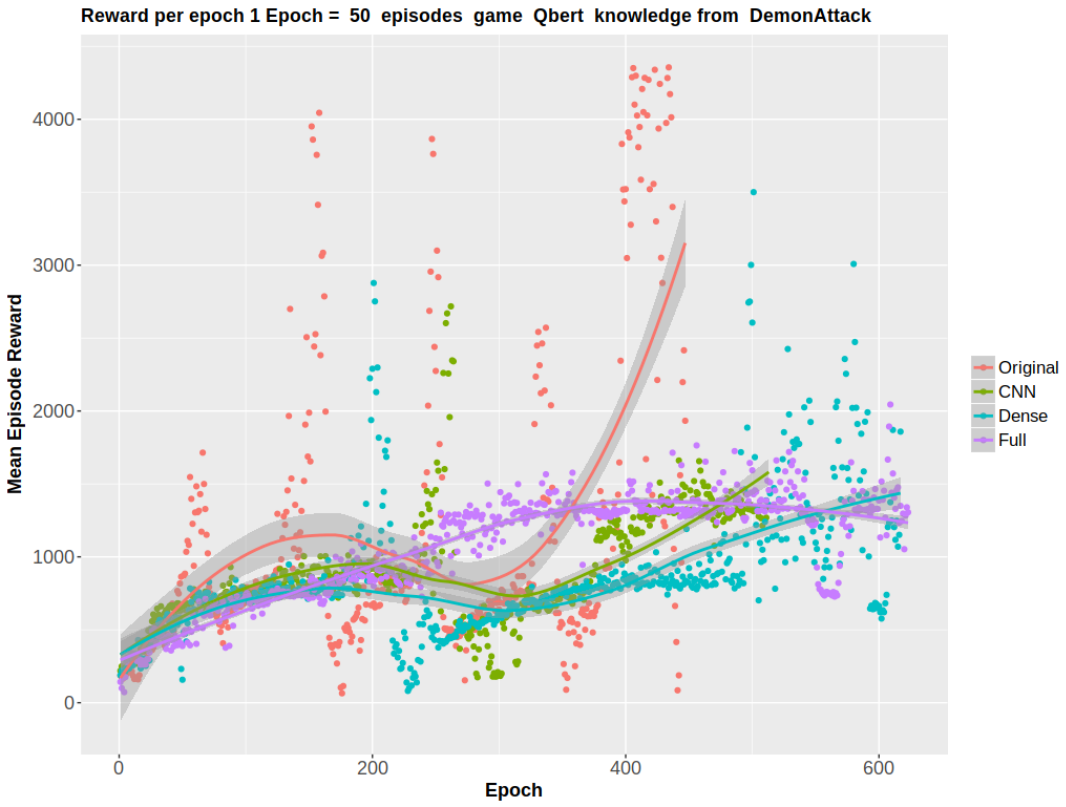

FIGURA 5.7: Recompensas medias obtenidas en los episodios de las experimentaciones Qb.1, Qb.2 y Qb.3

La Figura 5.8, contiene los pasos temporales medios en cada episodio. El eje de abscisas muestra los *epoch* y el eje de ordenadas muestra los pasos temporales medios por cada *epoch*. A más pasos temporales, mejor estrategia ha aprendido el agente ya que está aprendiendo a no perder dentro del juego.

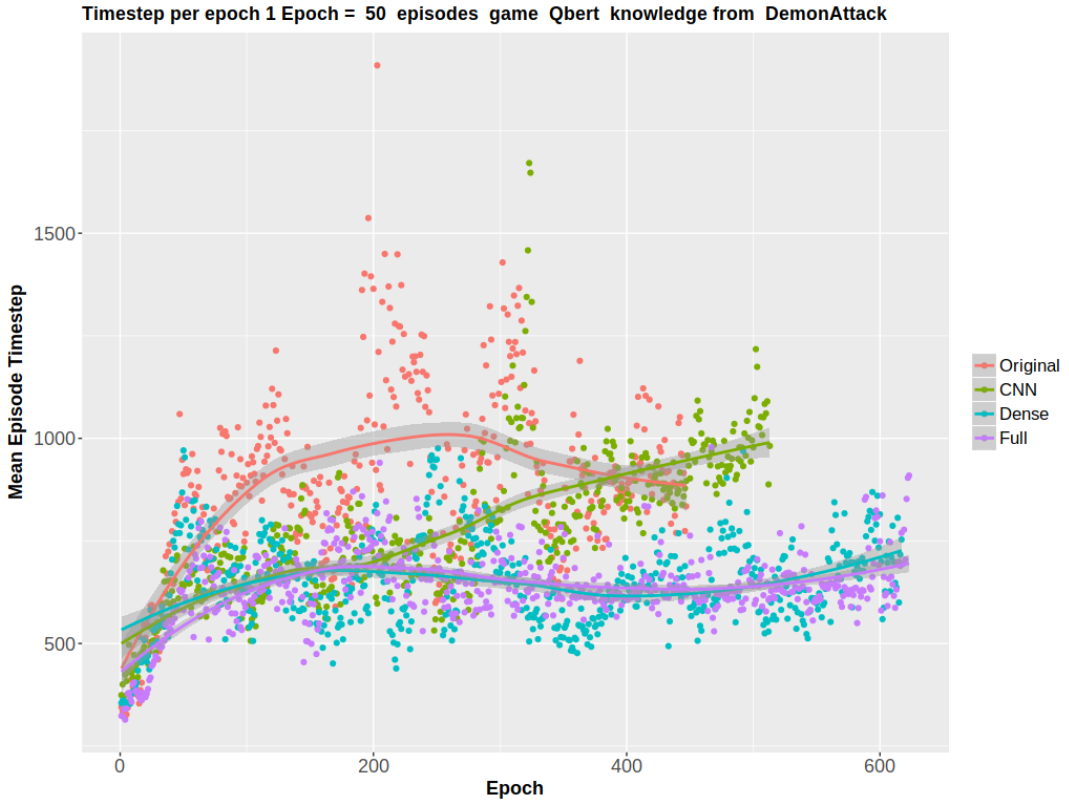

FIGURA 5.8: Pasos medios en los episodios de las experimentaciones Qb.1, Qb.2 y Qb.3

La Figura 5.9, contiene las recompensas acumuladas en todo el proceso experimental. El eje de abscisas muestra los Episodios jugados y el eje de ordenadas muestra la recompensa acumulada en cada Episodio jugado. A más recompensa acumulada, mejores resultados ha obtenido el agente en los Episodios jugados y por lo tanto, mejor estrategia ha sido capaz de aprender para jugar.

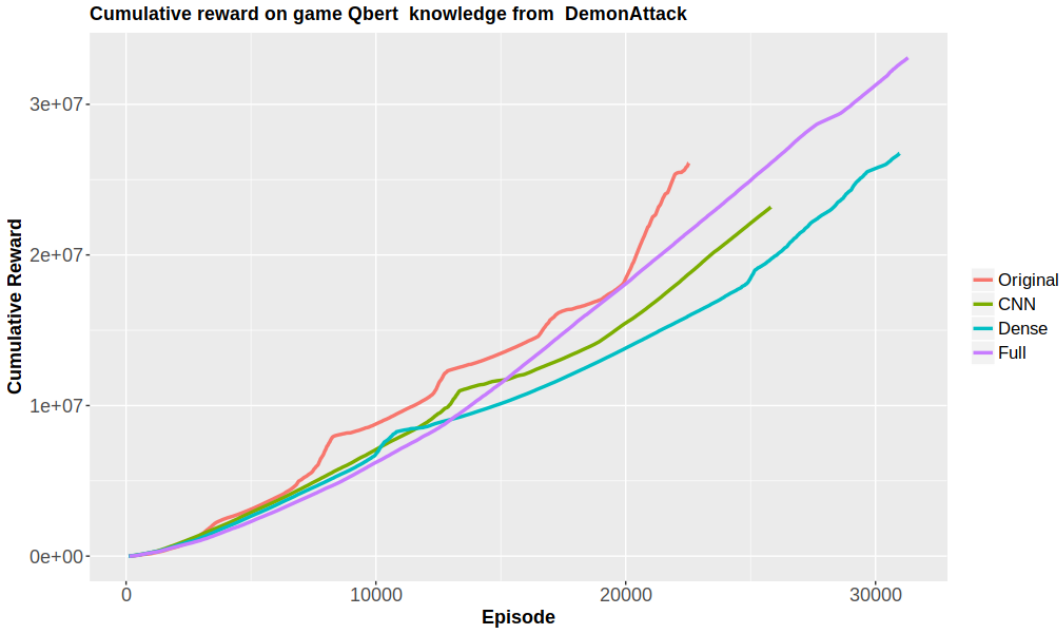

FIGURA 5.9: Recompensa acumulada de las experimentaciones Qb.1, Qb.2 y Qb.3

Gráficas para experimentaciones Qb.4, Qb.5 y Qb.6

Las siguientes gráficas muestran los resultados de las experimentaciones **Qb.4**, **Qb.5** y **Qb.6** que representan el entrenamiento de un agente inteligente que sepa aprender a jugar al juego *Qbert* haciendo uso del conocimiento guardado del juego *SpaceInvaders*.

La Figura 5.10, muestra la media de las recompensas obtenidos en cada *epoch*. El eje de abscisas muestra los *epoch* y el eje de ordenadas muestra las recompensas medias por cada *epoch* (media de 50 episodios). A más recompensa media obtenida, mejor estrategia ha aprendido el agente.

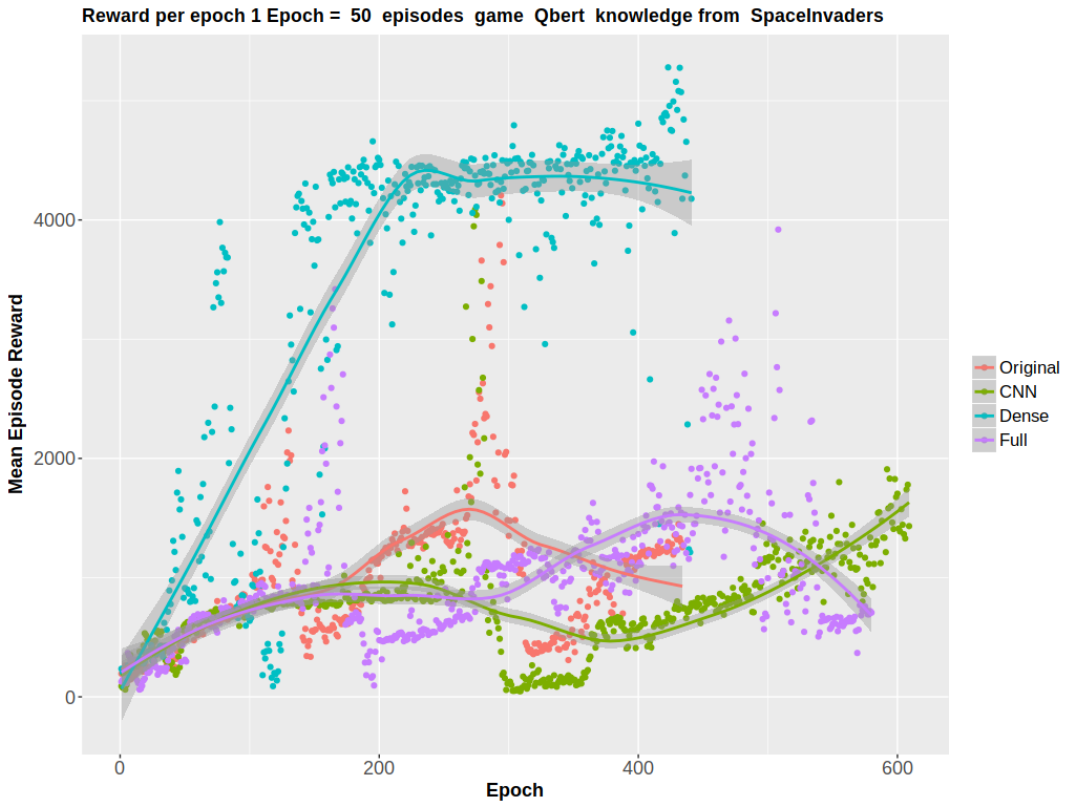

FIGURA 5.10: Recompensas medias obtenidas en los episodios de las experimentaciones Qb.4, Qb.5 y Qb.6

La Figura 5.11, contiene los pasos temporales medios en cada episodio. El eje de abscisas muestra los *epoch* y el eje de ordenadas muestra los pasos temporales medios por cada *epoch*. A más pasos temporales, mejor estrategia ha aprendido el agente ya que está aprendiendo a no perder dentro del juego.

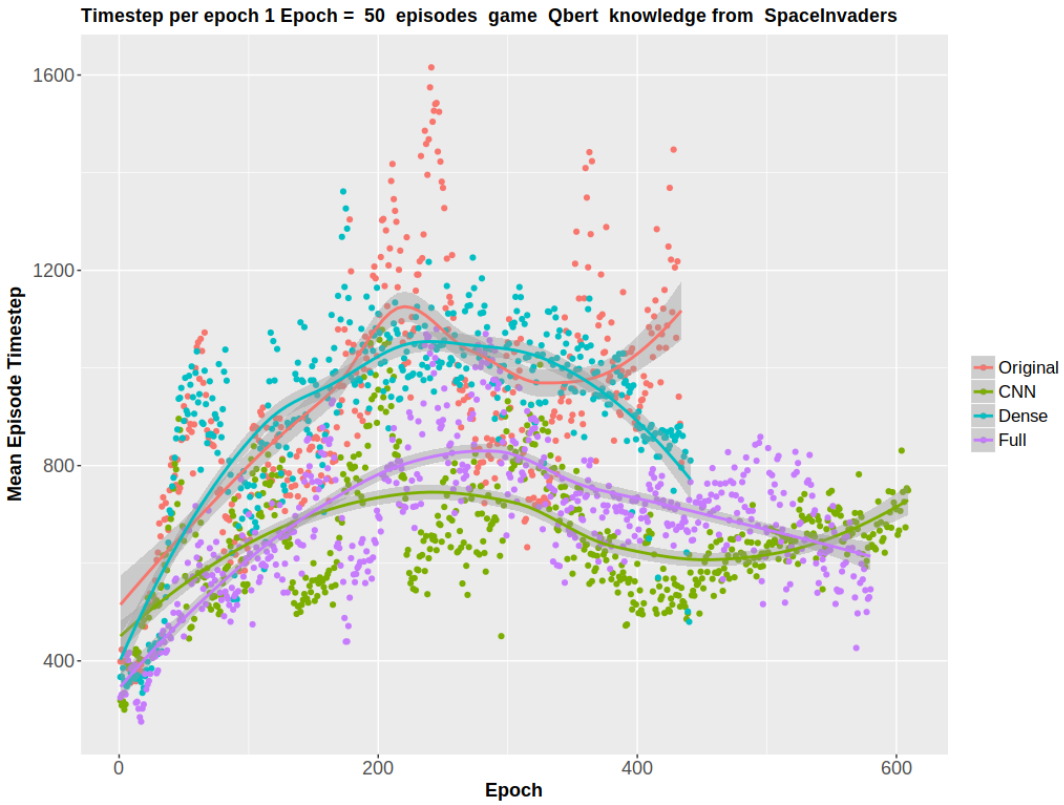

FIGURA 5.11: Pasos medios en los episodios de las experimentaciones Qb.4, Qb.5 y Qb.6

La Figura 5.12, contiene las recompensas acumuladas en todo el proceso experimental. El eje de abscisas muestra los Episodios jugados y el eje de ordenadas muestra la recompensa acumulada en cada Episodio jugado. A más recompensa acumulada, mejores resultados ha obtenido el agente en los Episodios jugados y por lo tanto, mejor estrategia ha sido capaz de aprender para jugar.

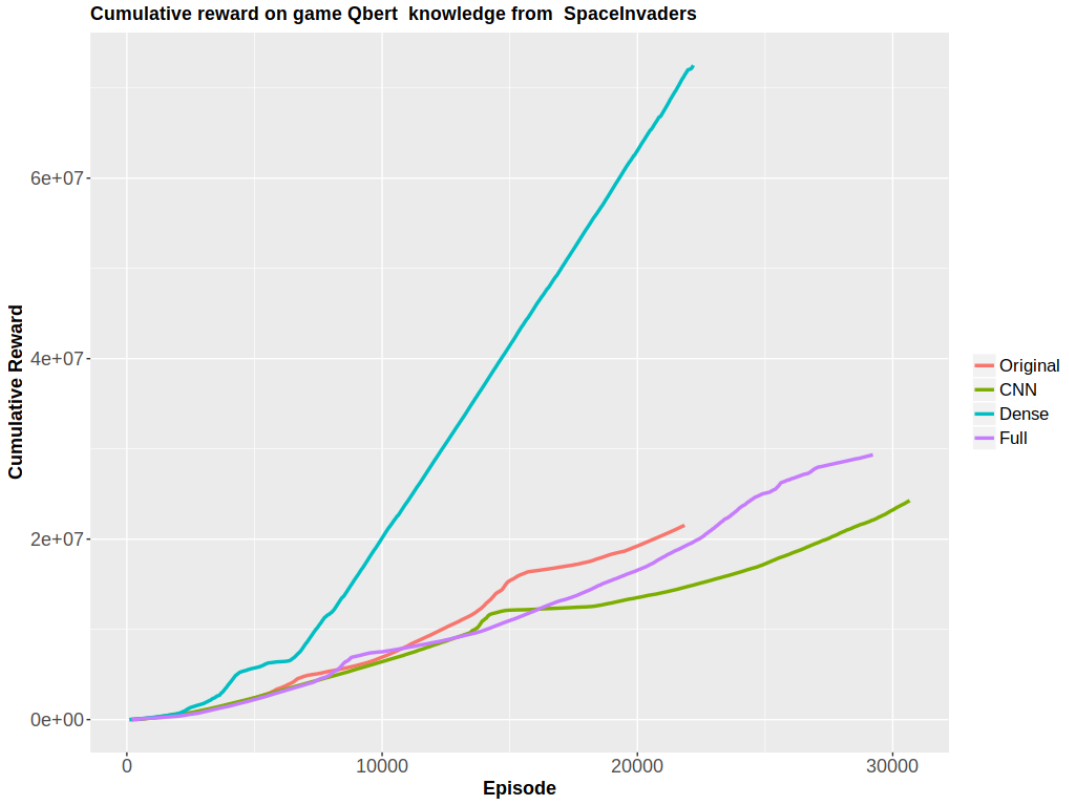

FIGURA 5.12: Recompensa acumulada de las experimentaciones Qb.4, Qb.5 y Qb.6

5.4. Discusión de los resultados obtenidos

Una vez mostrados los resultados obtenidos en las experimentación, se va a proceder a discutir los resultados obtenidos en ambas fases de la experimentación. Esta discusión será divida en dos subapartados para separar los dos procesos experimentales llevados a cabo.

Antes de comenzar con la discusión, se recomienda encarecidamente recordar cómo se han entrenado los agentes inteligentes. Si se revisa el texto en la Sección 3.5.2, se puede recordar que el factor de confianza durante la fase de exploración del agente ha sido alto, dejando claro que esto es debido a la importancia de no perturbar el conocimiento previo adquirido ya que se entiende que si un agente recibe conocimiento previo, este ya debería ser capaz por definición de poder tomar acciones inteligentes desde los primeros pasos temporales.

5.4.1. Experimentación 1

SpaceInvaders

Los resultados obtenidos en el juego *SpaceInvaders* dejan patente que la transferencia de conocimiento desde el juego *DemonAttack* no está ayudando a crear un agente inteligente más eficiente y en un tiempo menor. El rendimiento total obtenido no ha mejorado en ninguna de las experimentaciones realizadas. La Tabla 5.1 (Experimentaciones, SI.1, SI.2, SI.3 y AC.SI), muestra cómo el agente de control obtiene mejores métricas en prácticamente todas las pruebas realizadas. Las Figuras 5.1, 5.2,

5.3 muestran cómo, en todas ellas, el agente de control es quien mejor se comporta y mejores resultados finales obtiene. Repasando las **características visuales** expuestas en la Sección 3.2.2 se puede observar que ambos juegos guardan ciertas similitudes gráficas por las que la parte visual de extracción de la red neuronal debería ser capaz de identificar fácilmente alguno de los elementos del nuevos juego. Sin embargo, con los resultados expuestos se puede afirmar que esto no está ocurriendo. La adaptabilidad de las CNN en el nuevo entorno no está mejorando el rendimiento final del agente. La razón por la que esto ocurre, es debido a que aunque ambos juegos son muy parecidos, la estrategia que adopta el agente para poder ganar en ellos es ligeramente distinta. La Tabla 3.2 muestra como el juego *DemonAtatck* tiene ciertas disimilaridades en cuanto al tiempo total de un episodio, el tiempo para obtener un reward positivo, y el sistema de recompensas del juego. Estas diferencias hacen que la estrategia adoptada por un agente entrenado con el conocimiento del *DemonAtatck* sea, en general, más conservadora. De hecho, en las pruebas visuales realizadas, el agente de control (**AC.DA**) entrenado en el juego *DemonAttack* adopta una estrategia muy conservadora por la cual mantiene la nave desplazada a la derecha del entorno del juego durante el transcurso del episodio y "espera" a que los enemigos se acerquen para poder destruirlos. Cuando el agente ve que en un tiempo determinado no alcanza a ningún enemigo (no está obteniendo recompensas), entonces decide moverse para buscarlos y destruirlos. Adoptar esta estrategia a un juego como *SpaceInvaders* no es una estrategia óptima dadas las características del juego, ya que existe un límite de tiempo por el cual el episodio entra directamente en un estado terminal.

En la experimentación **SI.1**, se puede visualizar que los resultados obtenidos por el agente con las CNN pre-entrenadas no consiguen mejores resultados que el agente de control. La Figura 5.2 muestra cómo la duración de los episodios es inicialmente muy pobre, sobre todo en la fase de exploración. Dicha duración va aumentando poco a poco hasta alcanzar un punto de estabilidad por el cual los episodios duran de media el mismo tiempo hasta el fin de la experimentación. Los resultados no parecen tener sentido, porque según hemos comentado en las **características visuales** de ambos juegos, existe cierto parecido entre ambos en algunos elementos visuales. El problema es que conforme el agente sale de la fase de exploración y avanza en el experimento, el conocimiento adquirido de las CNN entorpece la búsqueda de una estrategia óptima ya que si visualizamos los valores que toma la gráfica para las CNN, se puede apreciar que el agente se atasca en un mínimo local y deja de explorar nuevos estados que podrían ser más eficientes (punto de estabilidad). Esto significa que los valores de las CNN del juego *DemonAttack* propician que el agente no explore de forma adecuada nuevos estados y este genera una política conservadora para maximizar la recompensa. Si visualizamos la Figura 5.1 para las CNN, se puede observar que el agente optimiza la estrategia que ha elegido pero no consigue los mejores resultados posibles por lo que la transferencia de conocimiento no ayuda a una mejora significativa. El agente consigue en cada episodio una duración similar en pasos temporales con unas recompensas medias cada vez mejores, pero no es capaz de alcanzar el mismo nivel de convergencia que hace el agente de control, dejando claro un comportamiento conservador y seguro en vez de un comportamiento exploratorio y confiado. Por último, la Figura 5.3, muestra cómo el conocimiento de las CNN tampoco arroja una mejora de las recompensas acumuladas. Sintetizando, se puede decir que los resultados obtenidos indican que, aunque ambos juegos tienen características visuales similares entre sí, las estrategias a adoptar son distintas y el conocer previamente algunos elementos visuales entorpece la adaptabilidad del agente a nuevos entornos.

En la experimentación **SI.2**, la transferencia del conocimiento de las redes Densas muestran una explotación efectiva del conocimiento. Ahora bien, dicha explotación no supera de forma general los resultados obtenidos por el agente de control. El agente, experimenta una tasa de resultados que se representan en la Figura 5.1 como una función monótona creciente por la cual si la duración del experimento fuese mayor, terminaría convergiendo en algún paso temporal t y obteniendo resultados similares al agente de control. Los resultados obtenidos, se traducen en que el agente "ya tiene una idea" de cómo funcionan los controles de la consola Atari, pero no es capaz de asociar perfectamente bajo qué estados específicos necesita decidir qué acción debe tomar (le falta la interpretación visual que debe adaptar). El ajuste que realiza el agente, es la evaluación sobre si la "memoria" que ya se tenia sobre el juego similar, está ayudando a obtener resultados efectivos en este nuevo juego. En resumen, aunque el agente utiliza un conocimiento basado en una resolución conservadora (irse a la derecha y disparar), la experiencia transferida de las redes Densas ayuda a adoptar una estrategia que no resulta ser puramente estocástica y que consigue unos resultados apreciables. La adaptabilidad de un juego a otro es bastante similar a lo que ocurre en la vida real, una analogía directa sería pasar de un lenguaje de programación a otro, no es lo mismo adaptarse de un lenguaje de programación como C a Java, que aprender C desde el principio. Dependiendo de cada persona (en este caso el agente), la adaptación puede ser más o menos costosa. Por último, en la Figura 5.3, se puede ver cómo la recompensa acumulada en la experimentación se acerca bastante a la recompensa acumulada por el agente de control. Esto nos dice que, el agente entrenado es muy conservador y que además los resultados obtenidos son constantes y crecientes con una varianza inferior que en las otras experimentaciones.

Por último, en la experimentación **SI.3**, se obtienen los peores resultados en términos de recompensa acumulada y media. Las Figuras 5.1 y 5.3, muestran cómo las recompensas obtenidas en toda la experimentación no mejoran, dejando claro que el conocimiento completo que contiene la estrategia del *DemonAttack* no es efectivo para el *SpaceInvaders*. Ahora bien, en la Figura 5.2 se puede apreciar cómo la duración de los episodios es superior a los otros dos casos de transferencia de conocimiento (solo Densas o solo CNN). El agente utiliza el conocimiento adquirido para no perder las partidas, por lo tanto, aunque no está maximizando las recompensas obtenidas, la experiencia completa le está ayudando a jugar el juego con la estrategia adquirida de otro. Dicha estrategia no es muy efectiva, porque como se ha descrito anteriormente es demasiado conservadora, pero le está sirviendo para "sobrevivir" en el juego y hacer episodios más largos. La interpretación que sugieren estos resultados, es que la suma de la estrategia conservadora más el reconocimiento previo de algunos elementos de las imágenes hacen que el agente perfile una estrategia por la cual no sepa exactamente qué elementos dan una recompensa factible dentro del entorno, pero sí que sepa qué elementos son los que le hacen perder en el juego. El agente consigue aprender los elementos que le ayudan a sobrevivir, pero descarta los elementos que le hacen ganar porque el conocimiento visual y operativo que tiene está centrado en elementos visuales diferentes. Si recordamos las características visuales y se revisa la Figura 3.2 se podrá apreciar que los enemigos a destruir son diferentes en un juego y otro, y que por lo tanto, el agente no es capaz de identificar correctamente este hecho como para poder mejorar en los entrenamientos. Por último, examinando la curva que describe el agente en la Figura 5.2, muestra cómo este llega a conseguir una estrategia por la cual no muere con tanta facilidad. Esto reafirma las argumentaciones dadas acerca de cómo el agente está aprendiendo a sobrevivir dentro del juego.

DemonAttack

Los resultados obtenidos en el juego *DemonAttack* revelan que la transferencia de conocimiento desde el juego *SpaceInvaders* genera un impacto positivo en el proceso de aprendizaje del agente. Sin embargo, aunque dicho proceso muestra diferencias significativas respecto al agente de control, al final de las experimentaciones es el agente de control quien obtiene mejores resultados acumulados. En la Sección 3.2.2 ya se ha podido observar las similitudes existentes entre un juego y otro. En este caso, el juego *SpaceInvaders* identifica más elementos en pantalla que el juego *DemonAttack* lo cual propicia a una adaptabilidad más estable a la hora de que el agente se de cuenta que faltan elementos en pantalla (faltan enemigos). La Tabla 5.1 y las Figuras 5.4, 5.5 y 5.6 muestran las diferentes variaciones que sufre el agente en cada tipo de experimentación. La transferencia de conocimiento no genera el mejor escenario posible pero si se estudian las Figuras y las métricas proporcionadas por los resultados experimentales, se puede observar que en ciertas fases la transferencia de conocimiento ha sido clave a la hora de obtener una mejora en el proceso de aprendizaje. La razón por la que el agente de control termina teniendo un comportamiento más eficiente, es debido a las característica del juego y a la adopción de la estrategia por parte del agente. La Tabla 3.2, muestra cómo el juego *SpaceInvaders* es un juego más rápido y dinámico debido a que la tasa de duración de los episodios y la facilidad en conseguir puntuación es más baja, generando agentes inteligentes que tienden a responder rápidamente a cambios significativos dentro de una partida. El juego *DemonAttack* presenta una modelo más lento debido a que las recompensas obtenidas van en aumento conforme el agente va paulatinamente eliminado enemigos; este comportamiento no se produce en el juego *SpaceInvaders*.

En la experimentación **DA.1** se obtienen resultados inferiores en duración y recompensa comparados con el agente de control. Ahora bien, aunque los resultados sean inferiores, estos son mucho más estables mostrando una varianza baja. En esta experimentación se repite el mismo caso producido en la experimentación **SI.1** por la cual el agente obtiene un mejor rendimiento en las fases exploratorias pero, más adelante, dicho rendimiento se ve mermado. Al ser ambos juegos gráficamente similares, la argumentación es exactamente que en la experimentación **SI.1** por la cual, aunque el agente sabe reconocer algunas formas ya de entrada en el juego, la calidad de exploración en fases exploratorias es menor y el agente no termina por aprender una política óptima. De hecho, si visualizamos las Figuras 5.4 y 5.5 podemos ver cómo las líneas de las CNN describen funciones monótonas crecientes que fluctúan poco y mantienen una estabilidad completa. En la Figura 5.6, se puede observar que la recompensa acumulada no es nada positiva en comparación al agente de control; se han necesitado bastantes episodios para obtener una recompensa "aceptable". Por lo tanto, y resumiendo esta experimentación, se puede afirmar que el agente se ha quedado bloqueado en un mínimo local y su tasa de aprendizaje se ve mermada por el conocimiento visual previo.

En la experimentación **DA.2**, los resultados obtenidos son muy positivos. La media de las recompensas obtenidas en los episodios han sido ligeramente mayores comparado con el agente de control y bastante mejores comparado con las experimentaciones **DA.1** y **DA.3**. Las Figura 5.4 demuestra que en un menor número de episodios, el agente es capaz de alcanzar unos resultados medios muy altos en toda la fase experimental. Si se observan las métricas expuestas en la Tabla 5.1, se puede ver que esta experimentación arroja los mejores resultados en las recompensas medias y en la duración de episodios. La Figura 5.5, muestra un incremento en la duración de cada episodio, siendo estos poco a poco más cortos y aumentando la

puntuación final obtenida (lo cual indica que los enemigos van siendo más complicados a medida que surgen nuevas oleadas, lo que hace más complejo puntuar). Sin embargo, si se estudia la recompensa acumulada en la Figura 5.6, se puede visualizar que en la experimentación completa, el agente de control es quien obtiene una mayor recompensa total acumulada. Esto indica, que el agente obtenido al transferir el conocimiento de las redes Densas del juego *SpaceInvaders*, es un agente que obtiene recompensas muy dispares y con una alta varianza, lo cual impacta de forma negativa en los resultados totales. En contra, si visualizamos el agente de control, podremos ver que este muestra una función que describe un crecimiento más homogéneo y con poca varianza, obteniendo de forma acumulada mejores resultados. El razonamiento de éstos resultados deja patente que el conocimiento de las redes Densas del juego *SpaceInvaders*, es un conocimiento basado en "buscar y destruir", el cual espera una recompensa rápida y en poco tiempo. Aunque dicha estrategia es útil, en un juego como el *DemonAttack* no resulta ser la más efectiva, debido a que la IA del juego responde a los disparos del jugador esquivando de forma eficaz cualquier intento de agresión, por lo tanto, ésto hace que los resultados a obtener en cada episodio, puedan llegar a ser dispares. La duración de los episodios tan extensa demuestra que el agente no es capaz de destruir de forma eficaz a los enemigos; a diferencia del juego *SpaceInvaders*, el juego *DemonAttack* tiene una duración media de episodio mayor debido a que no existe un límite de tiempo como tal por episodio, dando lugar a episodios más largos de lo habitual aunque las recompensas obtenidas no sean las mejores. La conclusión final que se llega en la discusión de esta experimentación, es que la transferencia de conocimiento genera un agente con una estrategia eficiente para poder jugar al *DemonAttack*, pero en el fondo no termina siendo la mejor estrategia posible para obtener la máxima puntuación.

Por último, en la experimentación **DA.3**, los resultados obtenidos son bastante pobres. Si se visualizan las Figuras 5.4 y 5.6 se puede observar que los resultados obtenidos por el agente son inferiores respecto a las otras experimentaciones realizadas. Las recompensas medias obtenidas y la recompensa acumulada total durante la experimentación es inferior a las otras experimentaciones realizadas para este juego. Ésto indica que la estrategia completa del juego *SpaceInvaders* es incompatible con la jugabilidad ofrecida en el *DemonAttack*. La razón es compresible si visualizamos las diferencias entre ambos juegos que se reflejan en la Tabla 3.2. En dicha tabla se puede observar que en el juego *SpaceInvaders* se premia una estrategia rápida, ya que la duración de los episodios es menor y la capacidad de conseguir una recompensa es más inmediata que en el *DemonAtatck*. Además en el juego *SpaceInvaders* es más sencillo obtener recompensas más altas, debido al diseño del sistema de recompensas que tiene el juego por el cual el reward completo se encuentra en el mismo nivel y no en distintos niveles como sucede en el *DemonAttack*. También si se tiene en cuenta las **caracteristicas visuales** de cada juego y se visualiza la Figura 3.2, se puede apreciar que el transferir todo el conocimiento de un juego a otro, hace que al agente le cueste bastante adaptarse al nuevo entorno. Esto indica que, en algunas situaciones, el conocimiento previo entorpece el proceso de aprendizaje debido a que al agente le cuesta adaptarse de forma completa a todos los cambios que el nuevo juego le sugiere (como serían los cambios gráficos y los cambios de jugabilidad).

Además de estos resultados obtenidos, también es interesante visualizar el comportamiento que describe la Figura 5.5, en la cual se puede observar cómo en las fase de exploración la duración de los episodios es sensiblemente superior que el resto de experimentaciones. Esto, junto a las métricas que se muestran en la Tabla 5.1, demuestra que el agente está obteniendo recompensas muy bajas y episodios muy largos (tal y como ocurría en la experimentación anterior), y por lo tanto, se

puede afirmar que: 1) el agente ha aprendido a no perder y ha adoptado una estrategia segura que le ayuda a no morir fácilmente; 2) el agente no sabe exactamente cómo matar a los enemigos, sabe esquivarlos pero no sabe qué estrategia le va hacer mejorar su tasa de recompensa. En este punto, la diferencia fundamental entre esta experimentación y la experimentación **DA.2** es que el agente ya tiene experiencia previa de los controles, y por lo tanto, se adapta más rápidamente al juego haciendo que en pocos episodios la duración de estos sea inferior.

5.4.2. Experimentación 2

Qbert y el conocimiento de DemonAttaack

Los resultados experimentales obtenidos a la hora de entrenar un agente inteligente para el juego *Qbert* haciendo uso del conocimiento del juego *DemonAttack* revelan ciertas incógnitas sobre la transferencia de conocimiento utilizada. La Tabla 5.2 demuestra que la transferencia completa de conocimiento logra alcanzar los mejores resultados, debido a que tanto las recompensas medias y las acumuladas están siendo mayores que en el resto de los casos estudiados (como se verá a continuación). Se aprecia cómo la duración total de los episodios no es superior comparada con el agente de control, pero la estrategia adoptada consigue mejores recompensas que en el resto de las experimentaciones. Observando las **características visuales**, se aprecia que ambos video-juegos tienen varias diferencias visuales, pero la adaptabilidad que ofrecen las CNN muestra resultados mas concluyentes cuando se pasa de un entorno con pocos elementos visuales a un entorno con muchos elementos visuales. Por lo tanto, el conocimiento conservador del juego *DemonAttack* y la adaptabilidad a un entorno con más elementos visuales propician a que el agente sepa encontrar caminos para conseguir una estrategia más óptima que la alcanzada por el agente de control. Si se comparan las características de ambos video-juegos, en la Tabla 3.2 se observa cómo el sistema de recompensas de ambos juegos es prácticamente similar. Ahora bien, si entramos en más detalle, se puede ver que el grado de libertad de ambos juegos es distinto y que la duración terminal y el reward esperado también lo son, ésto hace que la experiencia que proporciona *DemonAttack* genere un agente inteligente que no sepa las limitaciones concretas de escenario que existe en *Qbert*, haciendo que los episodios sean cortos y poco extensibles.

QB.1, los resultados obtenidos son poco óptimos y muy dispares. En la fase de exploración, el agente consigue obtener un conjunto de recompensas estables y positivas como se puede apreciar en la Figura 5.7. El problema, es que el nivel de recompensas decrece conforme el experimento avanza. Esto significa que el agente se adapta de forma eficaz al nuevo escenario al que jugar, pero la falta de conocimiento previo sobre los controles, le impide saber exactamente qué decisiones tiene que tomar en cada estado de forma correcta para ganar recompensas positivas. Es importante recordar que a diferencia de los video-juegos *SpaceInvaders* y *DemonAttack*, *Qbert* es un juego en el que el agente entra en un estado terminal si este se sale de los límites del juego. La falta de conocimiento previo acerca de los controles del juego hace que el agente entre más fácilmente en un estado terminal y no consiga desarrollar una estrategia para obtener una recompensa máxima. Obviamente, a medida que el agente se adapte a los controles y a los gráficos, podrá mejorar sus recompensas y tiempos en cada episodio. De hecho, se puede apreciar en la figura cómo en las últimas fases del experimento, el agente repunta en recompensas y mejora las puntuaciones obtenidas. Además, si se observa la Figura 5.8 se aprecia cómo la duración de los episodios no decrece durante la experimentación, aunque la varianza es alta,

el agente consigue poco a poco adaptarse al nuevo juego y a entender cómo funcionan los controles frente a los elementos visuales existentes. Por último, si se estudia la Figura 5.9, se observa cómo esta experimentación no consigue la mejor recompensa acumulada, pero si muestra como los episodios son más extensos en cuanto a duración temporal. Esto ratifica lo anteriormente descrito sobre que el agente se adapta visualmente al nuevo entorno pero no termina de saber cómo conseguir recompensas de forma efectiva. El agente ha adoptado una estrategia por la cual no pierde fácilmente, pero le falta la experiencia sobre cómo funcionan los controles le limita a la hora de obtener recompensas.

En la experimentación **QB.2**, los resultados obtenidos son algo más pobres que los obtenidos en la experimentación anterior. En el proceso de exploración el agente tiene un rendimiento pobre con respecto a la experimentación **QB.1**. Ésto es debido a que el agente carece del apartado visual que gozaba la experimentación anterior, haciendo que aunque el agente tenga un conocimiento de los controles del juego, este no sepa explotarlos de forma adecuada al no saber asociar dichos controles con los elementos gráficos. Si se observa el rendimiento general de la experimentación en las Figuras 5.7 y 5.8 se puede identificar cómo el rendimiento de la experimentación fluctúa de forma positiva y negativa en todo momento. Las recompensas positivas y negativas tienen una varianza muy alta y el agente no llega a converger en una estrategia óptima para poder conseguir una puntuación que sea sensiblemente superior al agente de control. De hecho, si se observan los pasos temporales de los episodios, se puede ver cómo la nube de puntos genera una onda de episodios con muchos y pocos pasos, dando a entender que el agente no es capaz de optimizar la estrategia para conseguir más puntos. Estos resultados dan a entender que la experiencia de un juego como *DemonAtatck* no ayuda al agente a aprender a identificar los elementos del juego que le hacen ganar puntos y asociarlos correctamente con los elementos gráficos del juego. El agente intenta continuamente optimizar su estrategia haciendo que este termine entrando en un bucle, por el cual no consigue distinguir la mejor estrategia a adoptar con los elementos existentes en el juego. No obstante, la Figura 5.9 arroja un resultado inquietante por el cual las recompensas acumuladas superan los resultados obtenidos por el agente de control. Esto abre un debate entorno a la eficiencia del agente de control a la hora de obtener recompensas. Si revisamos de nuevo la Figura 5.7 se puede observar que el agente de control es quien más pasos temporales tiene de media en la experimentación frente al resto; ahora bien, las recompensas obtenidas son muy dispares si las comparamos con la experimentación actual. Esto quiere decir que, el agente de control intenta probar cosas nuevas (explorar opciones para mejorar) mientras que el agente entrenado con las redes Densas del juego *DemonAttack* entra en un bucle más cíclico que le hace ser más "estable" (arriesga menos para explorar nuevas opciones que el agente de control), haciendo que a la larga las recompensas finales sean más altas.

En la experimentación **QB.3**, los resultados obtenidos son bastante positivos. El conocimiento completo obtenido del juego *DemonAttack*, genera un agente que obtiene mejores resultados que el agente de control. Esto es debido a que el agente inteligente que se genera sigue una estrategia muy conservadora y no arriesga en tomar decisiones nuevas. Las Figuras 5.7 y 5.8 reflejan unos resultados muy compactos y con poca varianza. En la Figura 5.7 se puede apreciar cómo el agente entrenado consigue de forma continuada recompensas similares con una cierta mejora cuando el experimento va avanzando. El agente adopta una estrategia que va poco a poco optimizando hasta alcanzar un mínimo local. Con esta estrategia adoptada por el agente se puede identificar un punto de cualidades positivas y un punto de cualidades negativas. El punto positivo del agente entrenado es que los resultados

obtenidos en los episodios son muy similares, evitando episodios con recompensas muy altas o muy bajas y sugiriendo una estrategia estable y funcional. El punto negativo del agente entrenado es que dicho agente no intenta generar nuevas opciones que le ayuden a maximizar la recompensa posible. Por lo tanto, el conocimiento adquirido por el juego *DemonAttack* está mermando la capacidad de aprendizaje del agente y le impide explorar nuevas opciones que puedan mejorar su resultados totales (está limitando la capacidad de descubrir nuevos estados posibles). La Figura 5.8, muestra cómo los pasos temporales de los episodios son homogéneos y con poca varianza, afianzando el razonamiento que el agente adopta su estrategia conservadora. Este agente entrenado genera una recompensa acumulada mayor que el agente de control debido a esa estabilidad que muestra en los resultados obtenidos; ello se puede ver reflejado en la Figura 5.9 que muestra cómo la recompensa acumulada por el agente es mayor en esta experimentación que en el resto.

Qbert y el conocimiento de SpaceInvaders

Los resultados experimentales en el entrenamiento de un agente inteligente para el juego *Qbert* haciendo uso del conocimiento del juego *SpaceInvaders*, son científicamente relevantes en el estudio del TL. La Tabla 5.2 muestra las diferencias obtenidas entre el agente de control y el resto de las experimentaciones realizadas. El conocimiento transferido ha propiciado una mejora significativa en el juego. Los datos obtenidos concluyen que la estrategia adoptada en el juego *SpaceInvaders* ayuda a generar un agente inteligente que obtiene una máxima puntuación cuando se transfiere únicamente el conocimiento de las redes Densas. Si se revisan las **características visuales** expuestas en la Sección 3.2.2 se puede apreciar que las CNN amoldan los enemigos que aparecen en la parte superior de la imagen del juego *SpaceInvaders*, en los cuadros que conforman el juego *Qbert*. Además, si se revisa la Tabla 3.2, se puede distinguir que ambos juegos tienen ciertas similitudes en cuanto a duración terminal en pasos temporales (t) y facilidad de conseguir un reward positivo en poco tiempo. Esto nos indica que ambos juegos van a hacer que el agente inteligente consiga recompensas positivas en pocos pasos temporales creando un agente más rápido y dinámico a la hora de buscar puntuaciones.

En la experimentación **Qb.4**, los resultados obtenidos demuestran que la transferencia de conocimiento de las CNN no genera un agente inteligente que obtenga unos resultados aceptables. La Figura 5.10 muestra cómo las recompensas del agente son muy bajas, pero los resultados obtenidos son muy estables y con poca varianza, el agente no es capaz de asociar el conocimiento recibido con el sistema de juego de *Qbert* y falla estrepitosamente a la hora de conseguir asignar exactamente qué controles necesita utilizar para poder obtener recompensas válidas. Además, la Figura 5.11, muestra cómo la duración de los episodios es muy irregular; el agente no consigue encontrar cuál es la manera optima de asociar los elementos de la imagen con los controles necesarios para obtener puntuaciones positivas. Estos dos resultados indican que el conocimiento de las CNN del juego *SpaceInvaders* sufre una falta de adaptabilidad cuando el agente tiene que jugar al *Qbert*. Si a esta falta de adaptabilidad por parte de la parte gráfica se le suma al completo desconocimiento de los controles por parte del agente, entonces se junta una problemática por la cual se obtienen los resultados mostrados. Paradójicamente, aunque los resultados de la experimentación no son los mejores, la Figura 5.12 muestra que los resultados acumulados en total son mejores que los ofrecidos por el agente de control. Esto sucede únicamente porque el modelo obtiene recompensas bajas pero estables sin mucha varianza tal y como se ha explicado en las primeras líneas de este párrafo.

En la experimentación **Qb.5**, los resultados obtenidos muestran una mejora significativa en el proceso de aprendizaje del agente inteligente. En esta experimentación se ha utilizado el conocimiento de las redes Densas del juego *SpaceInvaders*. Si se visualizan los resultados medios obtenidos en la Tabla 5.2, se puede apreciar que desde el comienzo de la experimentación, el agente obtiene resultados con una media más alta que el resto de experimentaciones. A medida que el experimento avanza, el agente va comprendiendo los elementos que le dan una mejor puntuación y este mejora su eficiencia en cada episodio, haciendo que los resultados vayan mejorando hasta alcanzar una convergencia. Además, si se visualiza la Figura 5.11 se puede apreciar que la duración de los episodios entre el agente de control y esta experimentación es parecida, esto nos quiere decir que, en esta experimentación el agente "sabe" que tiene que activar todas las plataformas para ganar la partida mientras que en el agente de control este no termina de entender que tiene que pasar como mínimo una vez por cada casilla y termina en un proceso cíclico en el que técnicamente da vueltas sin un objetivo fijo. El razonamiento detrás de éstos resultados es debido a que el conocimiento de decisión transferido, hace que el agente ya sepa hacer uso de los controles de los cuales dispone. Dichos controles, están preparados para una rápida respuesta en un entorno basado en movimientos rápidos como se ejecutan en el *SpaceInvaders* (como se ha comentado en la Tabla 3.2) para buscar y destruir las naves enemigas. La diferencia está en que el agente tiene que adaptarse a las nuevas reglas del juego para perfilar mejor su jugabilidad. Este ejemplo se podría trasladar a la vida real donde un usuario ya conoce como funciona el mando de una consola y solo necesita aprender a jugar al nuevo juego para empezar a ganar partidas. Ese proceso de adaptación que sufre un usuario común lo sufre también el agente en la parte de la red encargada del reconocimiento gráfico (las CNN).

Por último, en la experimentación **Qb.6**, los resultados obtenidos sugieren una mejora comparado al agente de control y la experimentación **Qb.4**. En la Figura 5.10 se aprecia cómo en las primeras fases de la experimentación los resultados que se obtienen son muy similares a las otras experimentaciones, pero conforme avanza el experimento, el agente encuentra un mínimo local por el cual no optimiza ni mejora los resultados obtenidos. La Figura 5.11 refleja que la duración de los episodios es inferior comparándolas con algunas experimentaciones realizadas y que estos no duran ni mejoran de forma notable durante el transcurso de la experimentación. Las recompensas acumuladas obtenidas son ligeramente superiores a las obtenidas por el agente de control y la experimentación **Qb.4**. Estos resultados indican que el conocimiento visual que se transfiere entorpece al agente inteligente para reconocer los nuevos patrones del juego. Dicho conocimiento transferido por parte de las CNN no ayuda al agente a explorar nuevas formas de entender el nuevo video-juego, y por lo tanto, el periodo de adaptación no ocurre. Básicamente, transferir el conocimiento visual está creando un sesgo dentro del proceso de aprendizaje del agente que limita que este pueda aprender correctamente a diferenciar las nuevas formas, colores etc del nuevo juego al que debe aprender a jugar.

Capítulo 6

Conclusiones y trabajo futuro

6.1. Demostración de la hipótesis presentada

A la vista de la hipótesis y los sub-objetivos presentados en la Sección 1.5, se puede concluir que la hipótesis H_0 queda parcialmente aceptada debido a que es necesario realizar una investigación más exhaustiva y con más ejemplos. En general, los resultados demuestran que la adaptabilidad de los agentes inteligentes pre-entrenados no siempre es posible y se demuestra que los resultados pueden ser mejores o peores dependiendo de la naturaleza de los video-juegos empleados. Por lo tanto, no es posible afirmar que la generalización sea aplicable para todos los casos en los cuales la transferencia de conocimiento mejore el rendimiento de creación de un agente inteligente; pero tampoco se puede rechazar que la transferencia de conocimiento no ayude en ningún caso. Repasando los sub-objetivos presentados se puede comentar lo siguiente:

- **SO1**: Los agentes inteligentes entrenados con CNN pre-entrenadas no mejoran los resultados obtenidos por un agente que comienza con CNN con pesos aleatorios. Aunque gráficamente los juegos muestran ciertas similitudes, la estrategia que tiene que adoptar el agente para conseguir la máxima recompensa es distinta. En general, los agentes inteligentes han demostrado obtener un menor número de recompensas, pero han demostrado mejorar los tiempos en los episodios (más timesteps o t) por lo que han ayudado a que los agentes hayan aprendido a no perder. La adaptabilidad de las CNN depende mucho de cada juego y se tiene que tener en cuenta que también cuenta mucho que el agente sea capaz de aprender la asociación entre lo que "ve" y lo que "ejecuta".
- **SO2**: Los agentes inteligentes entrenados con Densas pre-entrenadas han demostrado resultados muy positivos. En la mayoría de los casos, los agentes generados han conseguido puntuaciones muy altas y han dejado patente que una vez que se transfiere de las redes Densas, el agente "ya conoce los controles". El problema radica en que dependiendo del juego del que provenga el conocimiento, este puede entorpecer el proceso de aprendizaje o mejorarlo pero se puede apreciar que el proceso exploratorio queda algo limitado porque el agente tiene que adaptarse a los nuevos modos del juego. Aunque este sepa cómo jugar, necesita adaptarse a los nuevos gráficos, mecánicas etc.. del nuevo juego, y esta adaptabilidad resulta muy lenta en comparación con un agente que aprende desde el principio. Se considera que ésto es lo más cercano a lo que ocurre en la vida real, las personas humanas somos complicadas a la hora de cambiar un hábito y adaptarnos a otro distintos, dependiendo de cada uno o de qué hábito, nuestra adaptación puede ser mejor o peor.

- **SO3**: Los agentes inteligentes entrenados con CNN pre-entrenadas de otro video-juego distinto, han demostrado una mejora en los resultados obtenidos. En las experimentaciones realizadas la transferencia de conocimiento de las CNN ha mejorado los resultados finales obtenidos. Ahora bien, la mejora no ha sido significativa con respecto a los agentes de control. Es cierto que se puede hablar de que la generalización ha ayudado en la mejora, pero deja abierta la puerta a que el punto de mejora sea completamente dependiente al conocimiento previo que se utilice.
- **SO4**: Los agentes inteligentes entrenados con Densas pre-entrenadas de otro video-juego han demostrado mejorar los resultados obtenidos en algunos casos. Esto es un claro indicativo de que el conocimiento no se puede generalizar siempre ya que, todo depende de la naturaleza del conocimiento adquirido. Por lo tanto, la efectividad del agente es dependiente a la experiencia previa en otro video-juego.

6.2. Trabajo realizado

En esta tesis Fin de Máster se ha buscado satisfacer la hipótesis nula H_0 que dictamina que: *"El conocimiento adquirido por un agente en un juego determinado puede ser reutilizado en otros juegos para entrenar agentes inteligentes de una manera más rápida y que obtengan mejores resultados en menos tiempo"*. Para poder demostrar o rechazar la hipótesis presentada se han definido 4 sub-ojetivos que han buscado estudiar hasta qué punto se puede generalizar el conocimiento de un agente inteligente en un juego determinado. Dichos 4 sub-objetivos han presentado casos para saber: 1) Si las CNN son generalizables entre juegos similares; 2) Si las redes Densas son generalizables entre juegos similares; 3) Si las CNN son generalizables entre juegos distintos; 4) Si las redes Densas son generalizables entre juegos distintos.

El procedimiento que se ha seguido ha comenzado con la presentación formal de los conceptos básicos acerca de la IA. En dichos conceptos, se ha presentado la descripción formal de la IA y qué tipo de variantes existen y han existido centrándose sobre todo en la que atañe a esta tesis que es la *IA computacional*. Una vez se han entendido los conceptos básicos de la IA, se ha hecho una descripción para entender una de las metodologías psicológicas más interesantes para crear una IA: el *aprendizaje por refuerzo*. Se ha detallado cómo funciona (estado, acción, reward....) y cómo se puede modelar matemáticamente (MDP) para representar un entorno real dentro de una computadora. Una vez entendido cómo se modela matemáticamente el aprendizaje por refuerzo, se ha explicado cuáles son las técnicas matemáticas existentes para poder resolver la modelización matemática del aprendizaje por refuerzo, se ha explicado las limitaciones y las ventajas/desventajas de ciertos algoritmos existentes (*Q-Learning*, *MC* ...) así como una de las maldiciones que poseen las técnicas llamada *exploration or exploitation problem*. Entendido el funcionamiento del RL, la tesis ha explicado brevemente el papel que juegan DL para crear, lo que hoy en día se conoce como el DRL que consiste en la suma del DL más el RL para crear agentes inteligentes capaces de aprender a realizar tareas usando únicamente como entrada los píxeles de una pantalla.

Entendidos todos los conceptos necesarios y formando al lector para que este tenga una idea básica, la tesis ha comenzado a explicar los distintos estados del arte de 4 disciplinas que han sido clave para poder realizar la experimentación de la H_0 presentada: 1) **Reinforcement Learning**: explicando sus inicios desde el campo de

la psicología hasta su formalización matemática para ser entendida por una computadora; 2) **Convolutional Neuronal Networks**: explicando qué son las ANN, cómo se han aplicado, qué son las CNN y cuáles han sido los distintos hitos y motivaciones que se han podido conseguir a día de hoy; 3) **Deep Reinforcement Learning**: porqué surge esta rama de la investigación, qué se ha podido conseguir y cómo ha sido posible adaptar los distintos algoritmos de las disciplinas de RL y DL ; 4) **Transfer Learning**: cuáles han sido las motivaciones que han generado la creación de esta línea de investigación y porqué este manuscrito busca afianzar los conocimientos que se han ido consiguiendo. Con la revisión del estado del arte, el lector ha podido comprender la problemática existente a día de hoy en la IA y la motivación y necesidad de validar o rechazar la H_0 presentada.

Una vez que el lector ya sabe cuál es el estado del arte, la tesis ha comenzado a explicar qué tipos de componentes técnicos se han utilizado durante la fase experimental. Se ha explicado qué lenguaje de programación se ha utilizado, siendo en este caso el lenguaje de programación Python. También se ha explicado qué librerías se han utilizado, que en este caso han sido las librerías de manejo de tensores TensorFlow y la API de alto nivel de creación de modelos DL Keras. Se ha explicado cómo funciona el banco de trabajo para desarrollos rápidos basados en RL o DRL llamado *OpenAIGym*. Se ha comentado qué juegos se han utilizado y qué características lógicas y visuales tiene cada uno de ellos. Se ha explicado en profundidad el modelo de red neuronal utilizado para entrenar los agentes inteligentes, describiendo cada tipo de capa, su función de activación y su papel para conseguir que mediante la entrada de una serie de imágenes, la red sea capaz de dar una respuesta válida. Se ha presentado el algoritmo de RL elegido para ser integrado en el algoritmo DRL, exponiendo las razones y motivaciones de su uso y explicando cómo funciona de una forma más completa. Por último, se ha explicado el algoritmo DRL final utilizado en la fase de experimentación, con una descripción exhaustiva de cómo funciona, mostrando un pseudo-código que ayuda a entender su funcionamiento, los hiperparámetros utilizados, la manipulación de imágenes que se realiza etc..

Teniendo claro la hipótesis que se desea demostrar, en qué punto del estado del arte se encuentra ahora mismo esta investigación y cuáles son las herramientas que se disponen para hacer las experimentaciones, se ha explicado cómo se va a realizar el proceso experimental. Se ha explicado qué tipo de métricas se desean obtener y para qué vale cada una de ellas. Se han identificado las dos fases de experimentaciones y los identificadores de los experimentos llevados a cabo en cada una de ellas. Por cada experimento mostrado, se ha desgranado qué tipo de modificaciones se han querido hacer. También se ha explicado la plataforma experimental utilizada; de esta forma es posible reproducir de forma fidedigna el algoritmo presentado para los experimentos. Por último, se ha descrito qué procedimiento se va a utilizar para mostrar los resultados obtenidos por la experimentación descrita.

Una vez entendido el procedimiento experimental, se han mostrado al lector los resultados obtenidos en la investigación realizada. Se ha mostrado por cada fase experimental realizada, una tabla que ha representado las métricas recogidas durante la experimentación y unos gráficos producidos por los resultados obtenidos en la experimentación. Con los datos recogidos se ha realizado una discusión para interpretar y razonar cuáles han sido las razones que han hecho que la transferencia de conocimiento haya podido ser o no generalizable en cada tipo de experimento llevada a cabo, realizando las referencias necesarias a las características lógicas y visuales de cada juego y mostrando una discusión que haga entender al lector por qué se han obtenidos los resultados mostrados.

Por último se ha aceptado parcialmente la H_0 presentada y se ha esgrimido una

serie de razonamientos haciendo uso de los sub-objetivos marcados inicialmente. Con esta aceptación parcial queda la puerta abierta a futuras investigaciones en aras a saber realmente hasta qué punto la transferencia de conocimiento es posible dentro del campo de DRL y cual podrían ser las limitaciones existentes en el campo.

6.3. Lineas futuras

La investigación llevada a cabo ha dejado una puerta abierta sobre la transferencia de conocimiento dentro del campo del DRL. Los resultados obtenidos en el proceso experimental realizado, han mostrado situaciones de mejora y de empeoramiento dependiendo principalmente del tipo de conocimiento transferido. Sin embargo, todavía queda mucha investigación que realizar:

1. **Existen más algoritmos de DRL a probar**: los algoritmos existentes de DRL no se limitan sólo al DQN. Existen variantes mejoradas del DQN (como el doble DQN, Soft-Q-Learning, etc...) y otros algoritmos más avanzados que usan sistemas distribuidos o entropías en ciertos cálculos para entrenar agentes inteligentes. Al ser algoritmos mucho más rápidos y más eficientes se puede acentuar el estudio de la transferencia de conocimiento. Además, dependiendo de cada juego, hay algoritmos que funcionan mejor que otros. Estudiar para cada uno qué algoritmo encaja mejor y qué hiperparámetros son los más adecuados es un trabajo laborioso. Sería interesante poder estudiar qué ocurre con la transferencia de conocimiento en cada tipo de algoritmo utilizado y con distintos hiperparámetros configurados.

2. **Realizar una transferencia selectiva:** en el estudio realizado se ha hecho uso de todas las capas de CNN o Densas del modelo presentado. Sería interesante probar qué pasa si se usara una pequeña parte de la red neuronal o un numero limitado de capas de la red. Ésto generaría otro caso de estudio para probar si pasar el conocimiento selectivo de algunas capas de la red condiciona o no la transferencia de conocimiento. Además, esta transferencia selectiva ayudaría a identificar qué partes de la red son las que contienen el conocimiento que hacen que el agente no consiga aprender de una forma efectiva bajo ciertas circunstancias (qué partes entorpecen el aprendizaje).

3. **Probar contextos diferentes:** el estudio de la transferencia de conocimiento se ha hecho directamente sobre una muestra de juegos muy pequeña, pero el estudio de la transferencia de conocimiento no solo debe de limitarse a video-juegos. Sería interesante probar la transferencia de distintas políticas de otros campos como un brazo robótico, un coche autónomo u otro modelo preentrenado que tenga unas experiencias determinadas y ver qué sucede y estudiar qué elementos condicionan las transferencias de conocimiento o no.

4. **Simular un conocimiento continuo:** la generalización de las redes viene dada por la capacidad de aprovechar el conocimiento previo. Sería interesante saber qué pasaría si se probasen sucesivas transferencias de conocimiento desde distintos entornos para ver hasta qué punto se puede o no generalizar el conocimiento. La idea sería simular el aprendizaje continuo de la misma forma que lo hace un ser humano cuando este aprende cosas de forma continua (cúmulo de experiencias).

Además de los puntos indicados, también es imperativo estudiar nuevas metodologías de experimentación para poder recoger métricas más precisas o tener en cuenta nuevas variables para medir la calidad de las políticas aprendidas. Un ejemplo interesante sería la obtención de la entropía generada en el cálculo del la función $L(\theta)$ para intentar medir si la política que se está generando es la óptima o no siempre y cuando el algoritmo de DRL utilizado le permita calcular de forma óptima. Haciendo uso de la entropía para maximizarla puede ser determinante a la hora de calcular cuan buena es una política construida, sobre todo si la intención fuera transferir distintas políticas desde otras fuentes.

Por último, una de las cuestiones a estudiar adicionalmente a las líneas futuras, es el comportamiento y la capacidad de abstracción que puede tener el DL frente a lo que un humano es capaz de realizar. Autores como Chollet, 2017, describen que los humanos tienen una capacidad de abstracción mayor que un modelo de DL. Dicha capacidad de abstracción permite a un humano adaptarse más rápidamente a situaciones nuevas y desconocidas que un modelo de DL. Un modelo de red neuronal artificial, es entrenado mediante una serie datos que la permiten identificar ciertos elementos que "ha visto" previamente (conjuntos de entrenamiento con x instancias e y propiedades que determinan una clasificación). Si a dicha red se le presentan nuevas problemáticas completamente diferentes, su nivel de adaptabilidad no será tan efectiva como la de un ser humano debido a esa falta de abstracción en la percepción de las nuevas situaciones. Por lo tanto, las redes neuronales usadas en el DL solo son capaces de llegar a la llamada **generalización local**, la cual describe la dificultad que tiene la red en mapear los nuevos valores de entrada con los de salida cuando los datos de entrada son ligeramente distintos a los que ha aprendido durante el entrenamiento. El ideario futuro es conseguir que un modelo de DL sea capaz de llegar a la **generalización extrema** que es la capacidad de adaptación que tienen los seres humanos. Si una red neuronal alcanzara dicha generalización, podría ser capaz de adaptarse a nuevas situaciones haciendo uso de muy pocos datos de entrada o directamente, sin hacer uso de ningún dato de entrada. Resumidamente, se puede decir que de momento, la limitación existente en un campo como la IA radica en que las redes neuronales artificiales son adaptables hasta cierto punto pero aún no son capaces de amoldarse con la misma facilidad y rapidez que un ser humano. Aún queda mucha investigación que realizar para lograr que las redes neuronales artificiales obtengan una capacidad de generalización similar o mayor que la de un ser humano y poder entonces aplicarla en el campo del RL para crear agentes inteligentes que aprendiesen a jugar, por ejemplo, a nuevos video-juegos con un nivel de adaptabilidad igual o superior.

Bibliografía

Babbage, Charles y Francis Baily (1823). «On Mr. Babbage's new machine for calculating and printing mathematical and astronomical tables, from Fr. Baily». En: *Astronomische Nachrichten* 2, pág. 409.

Barreto, Andre y col. (2017). «Successor Features for Transfer in Reinforcement Learning». En: *Advances in Neural Information Processing Systems 30*. Ed. por I. Guyon y col. Curran Associates, Inc., págs. 4055-4065. URL: http://papers.nips.cc/paper/6994-successor-features-for-transfer-in-reinforcement-learning.pdf.

Bellman, Richard (1957). «A Markovian decision process». En: *Journal of Mathematics and Mechanics*, págs. 679-684.

— (2013). *Dynamic programming*. Courier Corporation.

Bellman, Richard E y Stuart E Dreyfus (2015). *Applied dynamic programming*. Vol. 2050. Princeton university press.

Bengio, Yoshua y col. (2009). «Learning deep architectures for AI». En: *Foundations and trends® in Machine Learning* 2.1, págs. 1-127.

Bertsekas, Dimitri P (1976). «Dynamic programming and stochastic control». En: *Mathematics in science and engineering* 125, págs. 222-293.

Bibbona, Enrico, Gianna Panfilo y Patrizia Tavella (2008). «The Ornstein–Uhlenbeck process as a model of a low pass filtered white noise». En: *Metrologia* 45.6, S117.

Brockman, Greg y col. (2016). *OpenAI Gym*. eprint: arXiv:1606.01540.

Chollet, Francois (2017). *Deep Learning with Python*. 1st. Greenwich, CT, USA: Manning Publications Co. ISBN: 1617294438, 9781617294433.

Dayan, Peter (1993). «Improving generalization for temporal difference learning: The successor representation». En: *Neural Computation* 5.4, págs. 613-624.

Dixit, Avinash K (1990). *Optimization in economic theory*. Oxford University Press on Demand.

Dreyfus, Stuart E y Averill M Law (1977). *The art and theory of dynamic programming, volume 130 of Mathematics in Science and Engineering*.

Espeholt, Lasse y col. (2018). «IMPALA: Scalable distributed Deep-RL with importance weighted actor-learner architectures». En: *arXiv preprint arXiv:1802.01561*.

Esteva, Andre y col. (2017). «Dermatologist-level classification of skin cancer with deep neural networks». En: *Nature* 542.7639, pág. 115.

Fukushima, Kunihiko y Sei Miyake (1982). «Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition». En: *Competition and cooperation in neural nets*. Springer, págs. 267-285.

Fulcher, John (2008). «Computational intelligence: an introduction». En: *Computational intelligence: a compendium*. Springer, págs. 3-78.

Hahnloser, Richard HR y H Sebastian Seung (2001). «Permitted and forbidden sets in symmetric threshold-linear networks». En: *Advances in Neural Information Processing Systems*, págs. 217-223.

Hasselt, Hado V. (2010). «Double Q-learning». En: *Advances in Neural Information Processing Systems 23*. Ed. por J. D. Lafferty y col. Curran Associates, Inc., págs. 2613-2621. URL: http://papers.nips.cc/paper/3964-double-q-learning.pdf.

He, Kaiming y col. (2016). «Deep residual learning for image recognition». En: *Proceedings of the IEEE conference on computer vision and pattern recognition*, págs. 770-778.

Hein, Daniel y col. (2017). «Particle swarm optimization for generating interpretable fuzzy reinforcement learning policies». En: *Engineering Applications of Artificial Intelligence* 65, págs. 87-98.

Hochreiter, Sepp y col. (2001). *Gradient flow in recurrent nets: the difficulty of learning long-term dependencies*.

Howard, Ronald A (1964). «Dynamic programming and Markov processes». En:

Huang, Jui-Ting y col. (2013). «Cross-language knowledge transfer using multilingual deep neural network with shared hidden layers». En: *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE, págs. 7304-7308.

Hubel, David H y Torsten N Wiesel (1959). «Receptive fields of single neurones in the cat's striate cortex». En: *The Journal of physiology* 148.3, págs. 574-591.

Huber, Peter J y col. (1964). «Robust estimation of a location parameter». En: *The annals of mathematical statistics* 35.1, págs. 73-101.

Kaelbling, Leslie Pack, Michael L Littman y Andrew W Moore (1996). «Reinforcement learning: A survey». En: *Journal of artificial intelligence research* 4, págs. 237-285.

Karpathy, Andrej y col. (jun. de 2014). «Large-scale Video Classification with Convolutional Neural Networks». En: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Kingma, Diederik P y Jimmy Ba (2014). «Adam: A method for stochastic optimization». En: *arXiv preprint arXiv:1412.6980*.

Konidaris, George y Andrew Barto (2006). «Autonomous shaping: Knowledge transfer in reinforcement learning». En: *Proceedings of the 23rd international conference on Machine learning*. ACM, págs. 489-496.

Krizhevsky, Alex, Ilya Sutskever y Geoffrey E Hinton (2012). «Imagenet classification with deep convolutional neural networks». En: *Advances in neural information processing systems*, págs. 1097-1105.

Lazaric, Alessandro (2012). «Transfer in reinforcement learning: a framework and a survey». En: *Reinforcement Learning*. Springer, págs. 143-173.

LeCun, Yann, Koray Kavukcuoglu, Clément Farabet y col. (2010). «Convolutional networks and applications in vision.» En: *ISCAS*. Vol. 2010, págs. 253-256.

LeCun, Yann y col. (1990). «Handwritten digit recognition with a back-propagation network». En: *Advances in neural information processing systems*, págs. 396-404.

LeCun, Yann y col. (1998). «Gradient-based learning applied to document recognition». En: *Proceedings of the IEEE* 86.11, págs. 2278-2324.

Lillicrap, Timothy P y col. (2015). «Continuous control with deep reinforcement learning». En: *arXiv preprint arXiv:1509.02971*.

Mahadevan, Sridhar y Jonathan Connell (1992). «Automatic programming of behavior-based robots using reinforcement learning». En: *Artificial intelligence* 55.2-3, págs. 311-365.

McCulloch, Warren S y Walter Pitts (1943). «A logical calculus of the ideas immanent in nervous activity». En: *The bulletin of mathematical biophysics* 5.4, págs. 115-133.

Metropolis, Nicholas y Stanislaw Ulam (1949). «The monte carlo method». En: *Journal of the American statistical association* 44.247, págs. 335-341.

Mirowski, Piotr y col. (2018). «Learning to Navigate in Cities Without a Map». En: *arXiv preprint arXiv:1804.00168*.

Mnih, Volodymyr y col. (2015). «Human-level control through deep reinforcement learning». En: *Nature* 518.7540, pág. 529.

Mnih, Volodymyr y col. (2016). «Asynchronous methods for deep reinforcement learning». En: *International Conference on Machine Learning*, págs. 1928-1937.

Mülling, Katharina y col. (2013). «Learning to select and generalize striking movements in robot table tennis». En: *The International Journal of Robotics Research* 32.3, págs. 263-279.

Narasimhan, Karthik, Tejas Kulkarni y Regina Barzilay (2015). «Language understanding for text-based games using deep reinforcement learning». En: *arXiv preprint arXiv:1506.08941*.

Pan, Sinno Jialin, Qiang Yang y col. (2010). «A survey on transfer learning». En: *IEEE Transactions on knowledge and data engineering* 22.10, págs. 1345-1359.

Parisotto, Emilio, Jimmy Lei Ba y Ruslan Salakhutdinov (2015). «Actor-mimic: Deep multitask and transfer reinforcement learning». En: *arXiv preprint arXiv:1511.06342*.

Peng, Xue Bin y col. (2018a). «DeepMimic: Example-Guided Deep Reinforcement Learning of Physics-Based Character Skills». En: *arXiv preprint arXiv:1804.02717*.

— (jul. de 2018b). «DeepMimic: Example-guided Deep Reinforcement Learning of Physics-based Character Skills». En: *ACM Trans. Graph.* 37.4, 143:1-143:14. ISSN: 0730-0301. DOI: 10.1145/3197517.3201311. URL: http://doi.acm.org/10.1145/3197517.3201311.

Puterman, Martin L (2014). *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

Riedmiller, Martin y col. (2009). «Reinforcement learning for robot soccer». En: *Autonomous Robots* 27.1, págs. 55-73.

Rumelhart, David E, Geoffrey E Hinton y Ronald J Williams (1986). «Learning representations by back-propagating errors». En: *nature* 323.6088, pág. 533.

Schaal, Stefan (1997). «Learning from demonstration». En: *Advances in neural information processing systems*, págs. 1040-1046.

Schrijver, Alexander (1998). *Theory of linear and integer programming*. John Wiley & Sons.

Schultz, Wolfram, Peter Dayan y P Read Montague (1997). «A neural substrate of prediction and reward». En: *Science* 275.5306, págs. 1593-1599.

Silver, David y col. (2014). «Deterministic policy gradient algorithms». En: *ICML*.

Silver, David y col. (2016). «Mastering the game of Go with deep neural networks and tree search». En: *nature* 529.7587, pág. 484.

Silver, David y col. (2017). «Mastering chess and shogi by self-play with a general reinforcement learning algorithm». En: *arXiv preprint arXiv:1712.01815*.

Simonyan, Karen y Andrew Zisserman (2014). «Very deep convolutional networks for large-scale image recognition». En: *arXiv preprint arXiv:1409.1556*.

Srivastava, Nitish y col. (2014). «Dropout: a simple way to prevent neural networks from overfitting». En: *The Journal of Machine Learning Research* 15.1, págs. 1929-1958.

Sutton, Richard S, Andrew G Barto y col. (1998). *Reinforcement learning: An introduction*. MIT press.

Szegedy, Christian y col. (2015). «Going deeper with convolutions». En: *Proceedings of the IEEE conference on computer vision and pattern recognition*, págs. 1-9.

Szegedy, Christian y col. (2016). «Rethinking the inception architecture for computer vision». En: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, págs. 2818-2826.

Taylor, Matthew E, Gregory Kuhlmann y Peter Stone (2008). «Autonomous transfer for reinforcement learning». En: *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 1*. International Foundation for Autonomous Agents y Multiagent Systems, págs. 283-290.

Taylor, Matthew E y Peter Stone (2007). «Cross-domain transfer for reinforcement learning». En: *Proceedings of the 24th international conference on Machine learning*. ACM, págs. 879-886.

Taylor, Matthew E y Peter Stone (2009). «Transfer learning for reinforcement learning domains: A survey». En: *Journal of Machine Learning Research* 10.Jul, págs. 1633-1685.

Tesauro, Gerald (1995). «Temporal difference learning and TD-Gammon». En: *Communications of the ACM* 38.3, págs. 58-68.

Thorndike, Edward L (1898). «Animal intelligence: An experimental study of the associative processes in animals.» En: *The Psychological Review: Monograph Supplements* 2.4, pág. i.

Van Hasselt, Hado, Arthur Guez y David Silver (2016). «Deep Reinforcement Learning with Double Q-Learning.» En: *AAAI*. Vol. 16, págs. 2094-2100.

Watkins, Christopher JCH y Peter Dayan (1992). «Q-learning». En: *Machine learning* 8.3-4, págs. 279-292.

Watkins, Christopher John Cornish Hellaby (1989). «Learning from delayed rewards». Tesis doct. King's College, Cambridge.

Werbos, Paul John (1994). *The roots of backpropagation: from ordered derivatives to neural networks and political forecasting*. Vol. 1. John Wiley & Sons.

Yosinski, Jason y col. (2014). «How transferable are features in deep neural networks?» En: *Advances in Neural Information Processing Systems 27*. Ed. por Z. Ghahramani y col. Curran Associates, Inc., págs. 3320-3328. URL: http://papers.nips.cc/paper/5347-how-transferable-are-features-in-deep-neural-networks.pdf.