Aprendizaje por refuerzo en videojuegos clásicos
Reinforcement Learning in classic games

Trabajo de Fin de Grado
Curso 2022–2023

Autor
Álvaro Álvarez Iglesias

Director
Belén Díaz Agudo

Grado en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

Aprendizaje por refuerzo en videojuegos clásicos
Reinforcement Learning in classic games

Trabajo de Fin de Grado en Ingeniería Informática

Autor
Álvaro Álvarez Iglesias

Director
Belén Díaz Agudo

Convocatoria: *Junio 2023*

Grado en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

29 de mayo de 2023

Resumen

Aprendizaje por refuerzo en videojuegos clásicos

El aprendizaje por refuerzo ya ha dominado la mayoría de los juegos arcade dando resultados incluso superiores a los de un jugador humano. Sin embargo, el cálculo de recompensas habitualmente hace uso de valores internos de la memoria RAM. En el caso del videojuego Tetris, podría tratarse del valor con la puntuación del jugador. Este método además de ser dependiente del juego en cuestión, no es comparable con la experiencia que tendría un jugador real.

Además, existen situaciones en las que no es posible guiar el entrenamiento usando recompensas convencionales, como por ejemplo en los juegos de exploración.

En su lugar, en este trabajo se propone hacer uso de las llamadas recompensas intrínsecas para entrenar un agente controlado por una red convolucional profunda usando únicamente la imagen. Las recompensas intrínsecas se suelen usar en conjunto con las clásicas, pero en esta ocasión se usarán de forma íntegra de forma que la implementación es independiente del juego en cuestión y más cercana a una experiencia humana.

Los resultados muestran que la función de recompensa implementada da la suficiente información como para poder entrenar un agente que complete niveles de algunos videojuegos y que sea capaz de navegar los distintos escenarios.

Palabras clave

Aprendizaje por refuerzo, Machine learning, Red neuronal, Python, recompensa intrínseca

Abstract

Reinforcement Learning in classic games

Reinforcement learning has already dominated most arcade games giving results even superior to those of a human player. However, reward calculation usually makes use of internal RAM values. In the case of the video game Tetris, this could be the value of the player's score. This method, besides being dependent on the game in question, is not comparable to the experience that a real player would have.

In addition, there are situations where it is not possible to guide training using conventional rewards, such as in exploration games.

Instead, in this paper we propose to make use of so-called intrinsic rewards to train an agent controlled by a deep convolutional network using only the image. Intrinsic rewards are usually used in conjunction with classical rewards, but here they will be used in their entirety so that the implementation is independent of the game in question and closer to a human experience.

The results show that the implemented reward function gives enough information to be able to train an agent to complete levels of some video games and to be able to navigate the different scenarios.

Keywords

Reinforcement learning, Machine learning, Neural Networks, Python, Intrinsic rewards.

Índice

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

El aprendizaje por refuerzo es una rama del aprendizaje automático centrada en entrenar un agente a tomar las decisiones correctas en un determinado escenario. Esta técnica ha demostrado ser eficaz para resolver problemas que requieren habilidad y adaptación, como los videojuegos.

Los videojuegos son el reto ideal para probar algoritmos y nuevas técnicas no solo de inteligencia artificial, sino de aprendizaje por refuerzo, ya que son un entorno controlado y con multitud de variables y reglas de las que aprender. Además, es fácil interactuar con ellos para extraer información del sistema. La memoria principal o memoria RAM contiene todo tipo de información útil como coordenadas, puntuaciones, contadores, vidas y demás variables que marcan el estado en el que se encuentra el juego en un momento dado.

Sin embargo, usar los datos de la memoria RAM para el entrenamiento no es la manera en la que un humano aprendería a jugar, este saca toda la información de la pantalla. Además de que hay juegos en los que no es posible medir numéricamente un avance, juegos en los que la exploración es un factor importante. Comparemos dos ejemplos concretos en dos videojuegos diferentes:

El videojuego “Tetris” es un juego del género arcade que consiste en apilar una serie de bloques de diferentes formas para formar filas. De este juego es sencillo extraer valores que nos indiquen el desempeño de un jugador, como el número de filas.

Un ejemplo del caso contrario sería un juego como “Pokemon Rojo”, un videojuego

de rol donde, durante la mayor parte del juego, controlas un personaje que explora un mapa predefinido. En este caso, la memoria RAM no contiene valores que, al menos a simple vista, nos aporten información relevante. No hay puntuaciones que marquen un progreso, por lo que medirlo es complicado y en ocasiones incluso subjetivo.

Es por esto que usar la imagen como datos de entrada no solo es más realista, sino que permite hacer un entrenamiento sobre distintos juegos sin necesidad de hacer distinción entre ellos.

Existen varios enfoques para este problema, pero uno razonable es explotar un elemento también encontrado en humanos: las recompensas intrínsecas, en este caso premiando la novedad. Se trata de aplicar un hash a cada uno de los estados, es decir, de las imágenes, y contar cuantas veces se repite ese estado. Esta técnica se llama "Counting after hashing". En resumen, premia el encontrar imágenes en pantalla nuevas.

En la mayoría de juegos, el hecho de encontrar contenido nuevo, nuevos gráficos o mapas es indicativo de que se está haciendo un buen trabajo, surgiendo incluso de este concepto géneros enteros de videojuegos. Por otro lado, no moverte o encontrar en múltiples ocasiones la misma imagen, se asocia a algo malo por el hecho de ser aburrido.

1.1. Motivación

No es raro ver a niños pequeños jugar a videojuegos desde muy temprana edad. Muchas veces todavía no saben leer o entienden bien lo que ocurre en pantalla y, sin embargo, son capaces de completar niveles o incluso el juego entero. Por ejemplo, ¿Cómo puede un niño que no sabe leer completar un juego como Pokémon, donde los diálogos son una parte importante de la jugabilidad?

Este trabajo nace de esta pregunta. Se enfoca en un entrenamiento más cercano al que tendría un humano, es decir, usando únicamente la pantalla, y se centra en un tipo de juegos que a penas están estudiados.

1.2. Objetivos

El objetivo final de este trabajo es desarrollar un algoritmo de aprendizaje automático que tome como input únicamente la imagen y utilice la técnica de ´´Counting after hashing". Por lo tanto, se marcarán una serie de objetivos que conduzcan al objetivo final y se seguirán como plan de trabajo.

1. Crear un entorno de desarrollo con las librerías necesarias, emuladores y control de versiones para poder llevar a cabo el seguimiento.

2. Crear la red neuronal convolucional que se usará de modelo para el agente y preparar el script de entrenamiento.

3. Establecer el sistema de recompensa.

4. Entrenar agentes progresivamente más elaborados sobre el juego principal.

5. Entrenar agentes sobre una serie de juegos secundarios.

6. Analizar los resultados de cada uno de los agentes y compararlos con el comportamiento esperado.

1.3. Estructura

Una vez explicado en líneas generales el tema y sus objetivos, en el capítulo 2 "Estado de la cuestión" se explica la teoría utilizada en este trabajo partiendo de los fundamentos y avanzando a través de los diferentes avances que han llevado a la situación actual También se explican algunos detalles sobre tecnologías utilizadas.

A continuación, en el capítulo 3 y 4, veremos el desarrollo del trabajo en sí, con sus correspondientes explicaciones sobre los métodos usados, decisiones tomadas y sus correspondientes resultados.

Finalmente, en el capítulo 5, se hará una breve conclusión a partir de los resultados obtenidos y se enumerarán las características que uno puede añadir al proyecto para mejorarlo.

Capítulo 2

Estado de la Cuestión

En este capítulo vamos a hablar de los elementos académicos que componen el proyecto, junto con algunos proyectos relevantes que tratan el mismo tema. Este trabajo abarca varios campos importantes de la inteligencia artificial como el aprendizaje profundo o el aprendizaje por refuerzo, en nuestro caso utilizadas en conjunto. Veremos los algoritmos más importantes y las bases de las que salieron, pasando por lo necesario para entender su funcionamiento y su importancia dentro del trabajo.

También se repasarán algunas de las herramientas más importantes en la actualidad para trabajar en el mundo de la inteligencia artificial y cómo se les ha dado uso aquí.

2.1. Inteligencia artificial en videojuegos

La inteligencia artificial en los videojuegos son los mecanismos y algoritmos usados para controlar o generar distintos aspectos del sistema. Desde prácticamente los comienzos de los videojuegos, se han desarrollado inteligencias artificiales para mejorar la experiencia de juego, aumentando la dificultad o dando una experiencia más personalizada al jugador. Cabe destacar que en muchas ocasiones las llamadas ïnteligencias artificiales"no pretenden imitar un comportamiento humano ni inteligente, componiéndose de patrones de respuesta predeterminados.

Uno de los primeros ejemplos de inteligencia artificial fue el videojuego Nim publicado en 1952, basado en el juego matemático homónimo de dos jugadores.

También fue uno de los primeros juegos en ser digitalizado y convertido a videojuego. Cada jugador debe turnarse para retirar uno o más objetos de una de las filas de objetos. El jugador que retira el último objeto pierde la partida.

En la década de los 70, con el éxito de los videojuegos arcade, empezaron a aparecer ejemplos de inteligencia artificial más compleja, siendo la que más influyente Space Invaders en 1978, que incluía diferentes patrones de movimiento y eventos dependiendo de acciones del jugador.

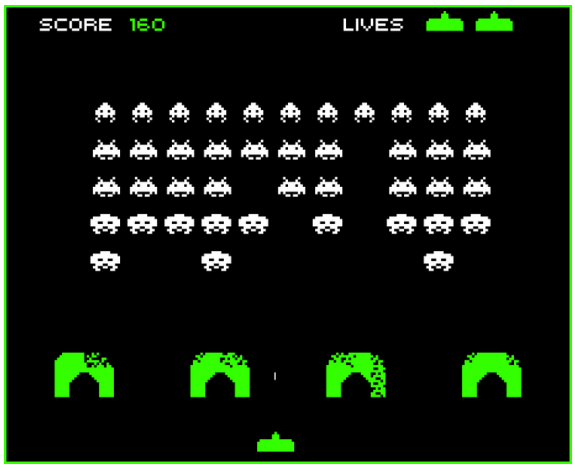

Figura 2.1: Space invaders (smithsonianmag (2018))

Conforme fue aumentando el poder computacional de los equipos de la época y apareciendo nuevos géneros de videojuegos, también se fueron desarrollando técnicas de control más avanzadas. Entre ellas, path-finding, máquinas de estados finitos, árboles de decisión, etc (Ocio Barriales (2014)).

En juegos modernos, se sigue usando los métodos convencionales de inteligencia artificial ya descritos porque suelen seguir siendo efectivos. Sin embargo, en algunos ámbitos donde la habilidad y la intuición son factores importantes en la jugabilidad como los juegos competitivos, estos se quedan atrás y no cumplen su objetivo de dar un reto al jugador.

Es aquí donde entra el machine learning, que ha demostrado ser muy efectivo en algunos juegos clásicos como Go o el ajedrez, o videojuegos como “Startcraft”. Estos juegos requieren, una vez más, de habilidad, intuición, toma de decisiones y estrategia. Aun así, se ha podido entrenar sistemas que aprendiendo de los patrones de comportamiento humano son capaces de ganar a los mejores jugadores humanos.

2.2. Machine learning

El machine learning es una de las ramas de la inteligencia artificial, cuyo objetivo es usar una serie de algoritmos para aprender de una serie de datos de entrada y, a partir de esos datos, hacer predicciones de nuevos datos nunca vistos sin necesidad de ser programadas directamente.

Se usan una serie de datos de entrenamiento, para construir un modelo que usa como hipótesis acerca del funcionamiento de un sistema determinado, extrayendo patrones sobre los datos que pueden ser imperceptibles para un humano. El machine learning se puede dividir en tres tipos principalmente: aprendizaje supervisado, el aprendizaje no supervisado, y el aprendizaje por refuerzo, del que se hablará en la sección 2.5 (Janiesch et al. (2021)).

2.3. Redes neuronales

Las redes neuronales son un modelo computacional inspirado en la estructura de un cerebro real. Aunque su uso esté enormemente extendido a día de hoy, su invención es ya bastante antigua, pero no se habían podido desarrollar hasta hace relativamente poco por falta de potencia computacional. Está formada por un conjunto de neuronas artificiales conectadas entre sí que procesan y transmiten la información entre ellas (Rosenblatt (1958)).

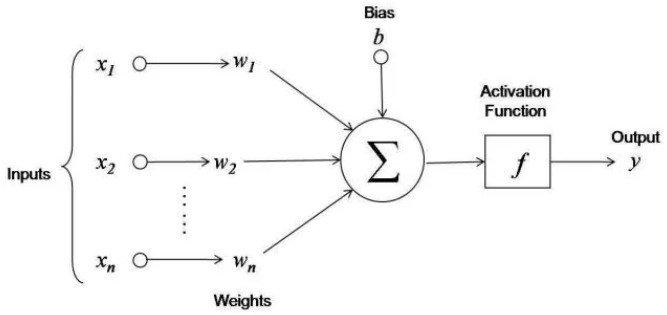

Figura 2.2: Esquema de una neurona (towardsdatascience (2019))

A diferencia de las neuronas reales, procesan la información usando una función matemática llamada función de activación. Existen distintas funciones de activación y cada una tiene efectos determinados sobre el funcionamiento final del modelo.

Algunos ejemplos son la función lineal, logística o tangente hiperbólica. Una vez procesada la información, se transmite a las neuronas a las que esté conectando a través de conexiones. Estas conexiones son el equivalente a los axiomas en una neurona real, y están ponderadas de forma que el valor transmitido se modifica en su proporción. Estos se llaman pesos.

El preceptrón, una de las primeras redes neuronales creadas y una de las más simples posibles, consta de una capa de entrada y una de salida. Este tipo de red neuronal tiene la desventaja de que solo puede operar en problemas linealmente separables (Mcculloch y Pitts (1943)). Este problema se solucionó con los perceptrones multicapa, donde las neuronas se ordenan por capas con distintas funciones, como la capa de input, output, y una o varias capas ocultas (Popescu et al. (2009)).

Cuando la red neuronal consta de más de una capa oculta, pasamos a hablar de Deep Learning o aprendizaje profundo. Esta es una nueva rama del aprendizaje automático que presenta varias ventajas frente a las anteriores. Poseen una mayor capacidad de generalización. Es decir, muestran un mayor desempeño a la hora de hacer predicciones sobre datos que no se usaron en la etapa de entrenamiento. También son capaces de aprender características más abstractas, útil a la hora de reconocer nuevos patrones (Janiesch et al. (2021)).

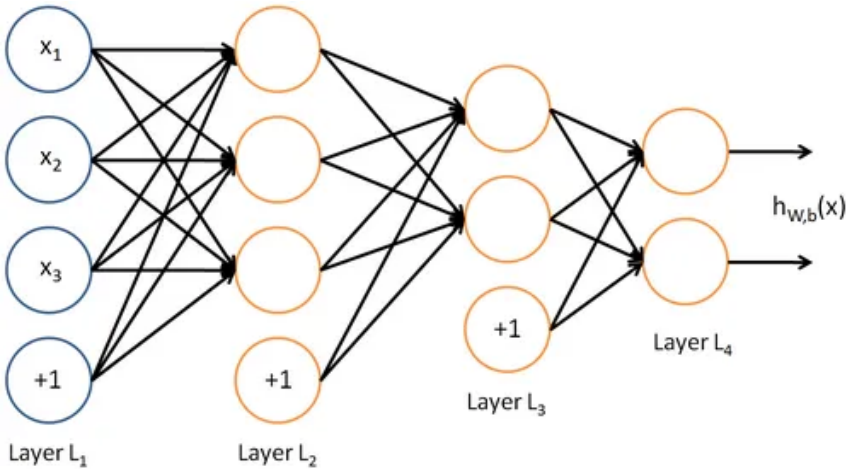

Figura 2.3: Esquema de un perceptrón multicapa (towardsdatascience (2019))

2.4. Redes neuronales convolucionales

Las redes neuronales convolucionales son un tipo de arquitectura de red neuronal especializadas en procesar datos multidimensionales como imágenes o vídeo (O'Shea

y Nash (2015)). Son ampliamente utilizadas a día de hoy en tareas como el reconocimiento facial o la detección de objetos puesto que han demostrado un muy buen desempeño (Lawrence et al. (1997)) (Pathak et al. (2018)).

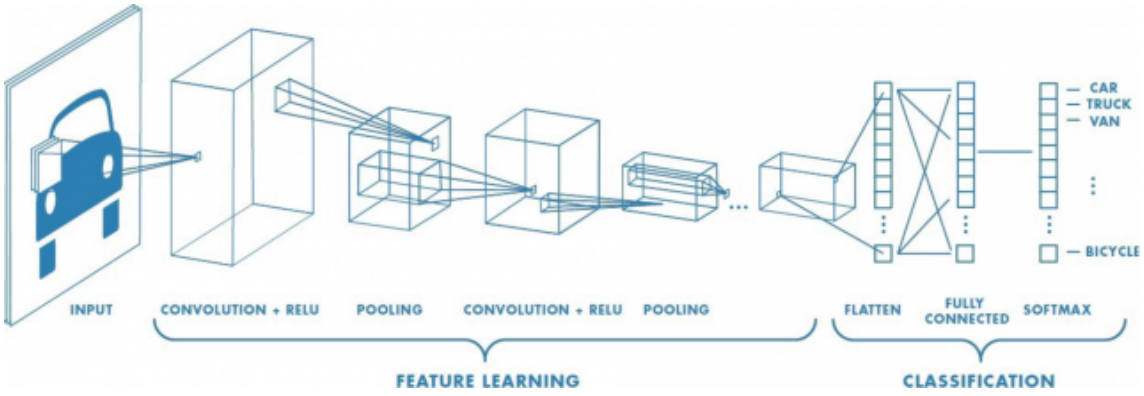

Figura 2.4: Esquema de red neuronal convolucional (Kumar (2021))

Al igual que las redes neuronales se basan en el concepto biológico de un cerebro, las redes convolucionales se inspiran en el funcionamiento de la corteza visual.

Son un tipo de red neuronal multicapa. Mientras que los perceptrones multicapa tienen todas las neuronas de una capa conectadas con todas las neuronas de la capa siguiente, las convolucionales están conectadas de una manera concreta para producir las convoluciones.

Una convolución es una operación matemática que permite la combinación de dos conjuntos de datos. En el caso de estas redes neuronales, se trataría de una porción del input y un filtro o "núcleo". La red hace esta combinación recorriendo elemento por elemento el input y realizando una multiplicación matricial, produciendo un mapa de características como resultado.(Brownlee (2020))

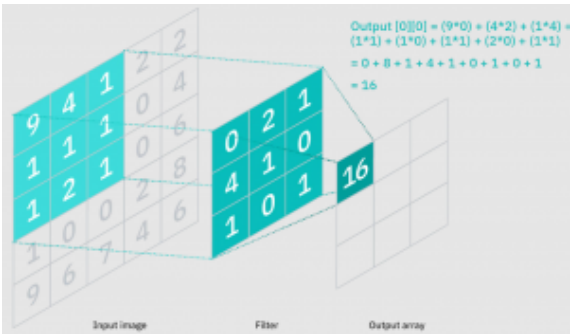

Figura 2.5: Ejemplo de una operación de convolución (Kumar (2021))

Habitualmente, la función de activación utilizada y la usada en este trabajo, es

la lineal rectificada (ReLu). Se ha convertido en la función de activación por defecto para muchos tipos de redes neuronales ya que es más fácil de entrenar que otras alternativas y habitualmente obtiene mejores resultados (Brownlee (2020)).

Finalmente, es habitual que las últimas capas de una red convolucional sean capas completamente conectadas. Primero se realiza una operación de aplanado y se alimenta a las siguientes capas como un vector unidimensional. Estas capas dan a la red la capacidad de entender realmente los patrones observados por las capas anteriores y de realizar la categorización o la operación deseada.

En este proyecto, la red neuronal a entrenar se trata de una red convolucional, cuya estructura completa se detallará en la sección 3.4. La capacidad de detectar patrones visuales hace este tipo de redes ideal para analizar la imagen de lo que sería la pantalla de cada videojuego, y tomar decisiones en base a ellos. Esta red se entrenará usando la técnica de aprendizaje por refuerzo.

2.5. Aprendizaje por refuerzo

El aprendizaje por refuerzo es una de las técnicas que componen el machine learning. Se basa en determinar cuál de las posibles acciones es la mejor en cada situación para maximizar la acumulación de recompensas obtenidas. Las recompensas vienen dadas por lo que se denomina función de recompensa.

Los problemas de aprendizaje por refuerzo se suelen afrontar como un proceso de decisión de Markov. Los MDP (Procesos de Decisión de Markov) son la forma idealizada matemáticamente del problema de aprendizaje por refuerzo, para el cual se podría encontrar un enunciado teórico preciso que pueda describirla, en otras palabras los MDP describen formalmente el medio ambiente en el cual se desarrolla el RL (van Otterlo y Wiering (2012)).

Un problema MDP está definido por una tupla de 4 elementos (S, A, Pa, Ra) donde:

1. S es el conjunto de todos los posibles estados s.

2. A es el conjunto de todas las posibles acciones. De la misma manera, A(s) es el conjunto de posibles acciones a realizar desde el estado s.

3. Dado s y s', Pa(s,s') se define como la probabilidad de que la acción a estando en el estado s conduzca al estado s'

4. Dado s y s', Ra(s, s') se define como la recompensa inmediata recibida al realizar la acción a para transicionar entre s y s'.

En el caso del aprendizaje por refuerzo, se trata de un caso concreto de MDP en el que Ra y Pa son desconocidos antes de empezar.

Los problemas de aprendizaje por refuerzo se afrontan con una simulación del entorno, que se ejecuta en pasos discretos. La simulación también se divide en episodios de una duración determinada, en la que se reinicia la simulación al final de cada uno. El objetivo último es maximizar la recompensa obtenida en cada episodio.

En una explicación menos formal, un problema de aprendizaje por refuerzo consta de los siguientes elementos: agente, entorno, estado y recompensa. El agente toma como input el estado para tomar la decisión de qué acción tomar. La acción modifica el entorno, y a partir del entorno se genera un nuevo estado y la recompensa asignada a la anterior acción.

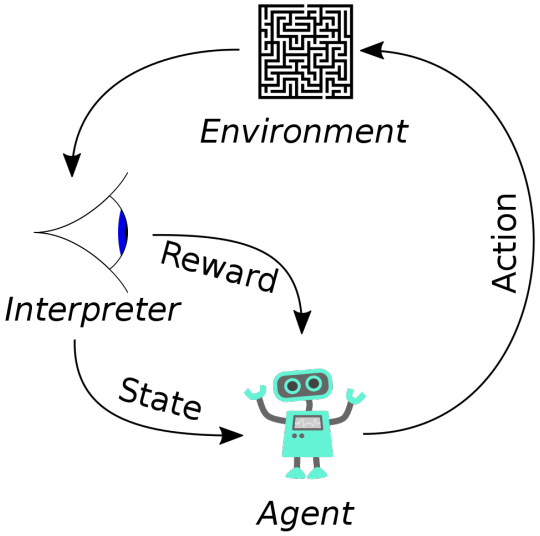

Figura 2.6: Esquema de los elementos de un problema de aprendizaje automático y su relación (Wikipedia (2023a))

Un estado puede tener multitud de formas, aunque tiene que tener un formato interpretable por el agente. Son un conjunto de variables que describen el entorno y es el diseñador del sistema el que elige qué variables son relevantes.

Si por ejemplo el sistema estuviese controlando un brazo robot, uso habitual en el

aprendizaje por refuerzo, el estado podría consistir en las variables que describen los diferentes motores del brazo. En el caso de un videojuego, el estado puede consistir en las diferentes variables del juego como coordenadas, puntuaciones, y localización de obstáculos. Otra alternativa y la utilizada en este trabajo es usar la lectura de la pantalla como estado.

Otro elemento a elección del diseñador del sistema es la función de recompensa, la encargada de evaluar los efectos que las acciones tienen sobre el entorno. En videojuegos, un ejemplo sencillo de recompensa sería la diferencia de puntuación. Un aumento de puntuación es buena señal, mientras que un descenso, mala (Sutton et al. (1999)).

El algoritmo básico y primero en aparecer que aplicaba este concepto se denomina Q-learning.

2.5.1. Q-learning

El Q-learning, introducido en Watkins y Dayan (1992) implementa la idea mencionada anteriormente en una tabla llamada tabla Q. Es un algoritmo muy simple ya que no necesita un modelo del entorno. La tabla se construye a base de tomar una acción determinada y observar la recompensa obtenida, dejándola apuntada para poder tenerla en consideración para tomar la siguiente acción. Q hace referencia a la función Q(s,a) que relaciona un valor de recompensa (Q) a los pares de estado (s) y acción (a)

Este funcionamiento se puede describir como un problema de programación dinámica. Existe una tabla de S filas y A columnas, donde S es el número posible de estados y A es el número posible de acciones a tomar. Esta tabla, que representa la recompensa esperada y está inicializada a un valor elegido por el programador.

La tabla se va rellenando siguiendo una simulación prácticamente igual a la descrita en la sección anterior. Está dividida en episodios, cada uno de una duración determinada y medidos en cuantos de tiempo discretos. En el ámbito de videojuegos estos pueden ser un número determinado de fotogramas. En el caso de un brazo robot, lecturas de los sensores.

Al inicio de cada episodio, la simulación se debe resetear a un estado inicial determinado. Este no tiene por qué ser siempre el mismo, pero es importante que

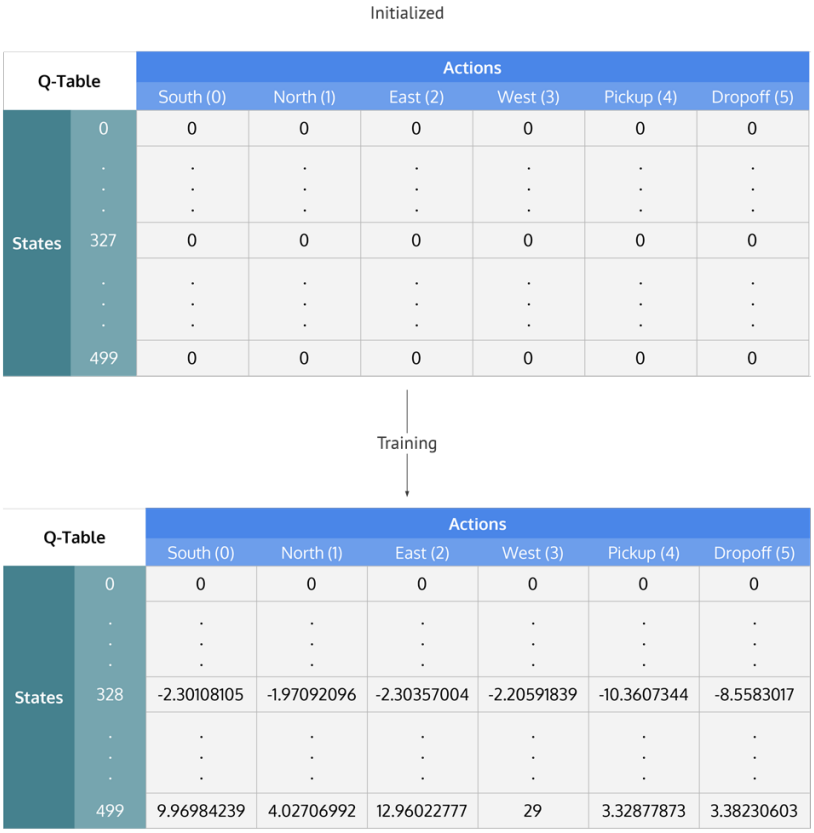

Figura 2.7: Ejemplo de tabla Q antes y después de un entrenamiento (Laskin (2022))

todos los episodios empiecen en un estado conocido. Entonces, se selecciona una acción de entre las posibles siguiendo algún criterio concreto o política como por ejemplo seleccionar la que tenga el valor Q más elevado. Esto produce un nuevo estado, y una recompensa, que se usa para actualizar la tabla Q siguiendo la siguiente fórmula:

$$\underbrace{\text{New}Q(s,a)}_{\text{New Q-Value}} = Q(s,a) + \underbrace{\alpha}_{\text{Learning rate}}[\underbrace{R(s,a)}_{\text{Reward}} + \underbrace{\gamma}_{\text{Discount rate}} \overbrace{\max Q'(s',a')}^{\substack{\text{Maximum predicted reward, given} \\ \text{new state and all possible actions}}} - Q(s,a)]$$

donde:

1. R(s,a) corresponde a la recompensa obtenida al realizar la acción a estando en el estado s.

2. La variable α es la tasa de aprendizaje. Sirve para que los cambios a los valores de la tabla sean más graduales.

3. La variable γ es la tasa de descuento de recompensas futuras. Un valor bajo hará que solo se fije en recompensas inmediatas, mientras que un valor alto hará lo contrario.

4. La expresión $maxQ'(s', a')$ hace referencia a la recompensa máxima posible tomando en consideración todas las posibles acciones en el siguiente estado.

Una vez actualizada la tabla Q con el nuevo valor, se vuelve a tomar otra acción, que producirá otro estado y otra recompensa, y así sucesivamente. Una vez llegado al límite de duración del episodio se resetea. El entrenamiento puede acabar una vez alcanzados un número concreto de episodios, al alcanzar algún objetivo como un estado concreto, o cualquier otro criterio.

Este algoritmo, aunque el primero de su tipo, tiene muchas limitaciones. La más evidente es el tamaño de la tabla. En escenarios donde haya un número reducido de estados es viable, pero en otros con mayor dimensionalidad como pantallas o con elementos continuos y no discretos no es viable ya que la tabla se vuelve demasiado grande y la memoria pasa a ser un problema.

Para abordar este problema, (Mnih et al. (2013)) propusieron el algoritmo de Deep Q-Network o DQN, que sustituye la tabla Q con una red neuronal profunda. En el siguiente punto lo explicaremos-

2.5.2. Deep Q-Network

El siguiente algoritmo, Deep Q-Network o DQN, es una variante del algoritmo anterior. Cuando la representación de un estado está formado por demasiadas variables o dimensiones, usar una tabla se presenta inviable. Por ejemplo, en una pantalla de 100 x 100 píxeles que puedan estar encendidos o apagados, resultaría en 10^8 posibles estados. DQN soluciona este problema sustituyendo la tabla Q por una red neuronal profunda que toma como entrada la representación del estado actual y produce el valor Q como salida para cada una de las acciones posibles.

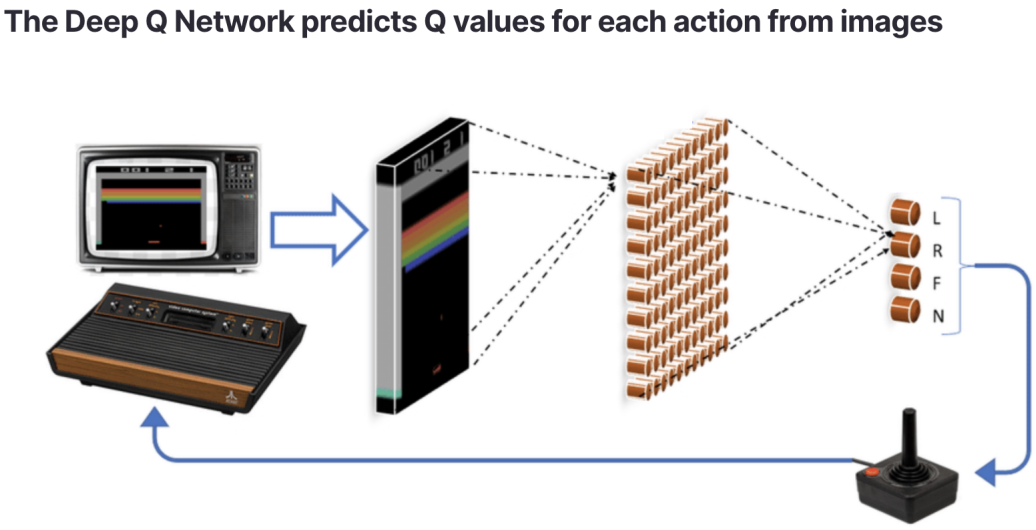

Figura 2.8: Deep Q-Network con capa convolutiva para controlar un juego de Atari (Wikipedia (2023b))

Una implementación directa, es decir, sustituir únicamente la tabla Q por una red neuronal profunda, provoca inestabilidad y no da buen resultado. Para entrenar la red neuronal, es necesario tener un conjunto de datos lo más equilibrado posible. Se mantiene un búffer, llamado búffer de memoria, con la información sobre los estados encontrados y las acciones tomadas. Del búffer se toman muestras aleatorias para hacer el entrenamiento de la red.

Esta técnica se llama “experience replay” y ayuda a evitar correlaciones temporales entre muestras que de otra forma estarían próximas entre sí. Gracias a esta técnica la red converge más rápido y tiene un entrenamiento más estable.

Otra técnica que contribuye a la estabilidad del entrenamiento en el algoritmo DQN consiste en no usar una red neuronal sino dos. Una de ellas se congela y se usa para tomar las decisiones durante la simulación. El entrenamiento se realiza sobre la otra red y ambas se sincronizan pasadas un número determinado de iteraciones.

La introducción de este algoritmo en el paper "Playing Atari with Deep Reinforcement Learning"(Mnih et al. (2013)) supuso una mejora importante en la capacidad de juego de los computadores, no solo mejorando el estado del arte del momento, sino, por primera vez, tener un algoritmo que era capaz de jugar mejor que un humano a múltiples juegos.

2.5.3. Proximal Policy Optimization

Por último, en el apartado de aprendizaje por refuerzo, hablaremos del Proximal Policy Optimization o PPO, desarrollado por OpenAI en 2017 (Schulman et al. (2017)). Está basado en otro algoritmo del mismo autor, el TRPO o "Trust Region Policy Optimization" Schulman et al. (2015) muy similar pero siendo el nuevo más fácil de implementar y más eficiente. Ambos son métodos iterativos que modifican el aproximador de la política haciendo uso del descenso de gradiente con el objetivo de producir cambios más poco a poco en cada iteración en comparación a otros algoritmos.

El principal problema del algoritmo TRPO es no solo que es bastante más difícil de implementar a nivel de código, sino que también es computacionalmente más pesado, algo poco conveniente en un método iterativo que se espera que pase por cientos de miles de iteraciones. Hace uso de la inversa de la derivada de segundo orden para calcular el ascenso de gradiente.

El algoritmo PPO en su lugar muestra resultados incluso mejores usando solo la primera derivada al mismo tiempo que realiza un mejor trabajo a la hora de modificar la política, de manera más consistente y segura. Esto lo hace gracias a varias restricciones, entre ellas la del clipaje.

El clipaje consiste en que en cada iteración de modificación de política, existe una cota que marca el cambio máximo permitido entre la versión anterior de la política y la nueva.

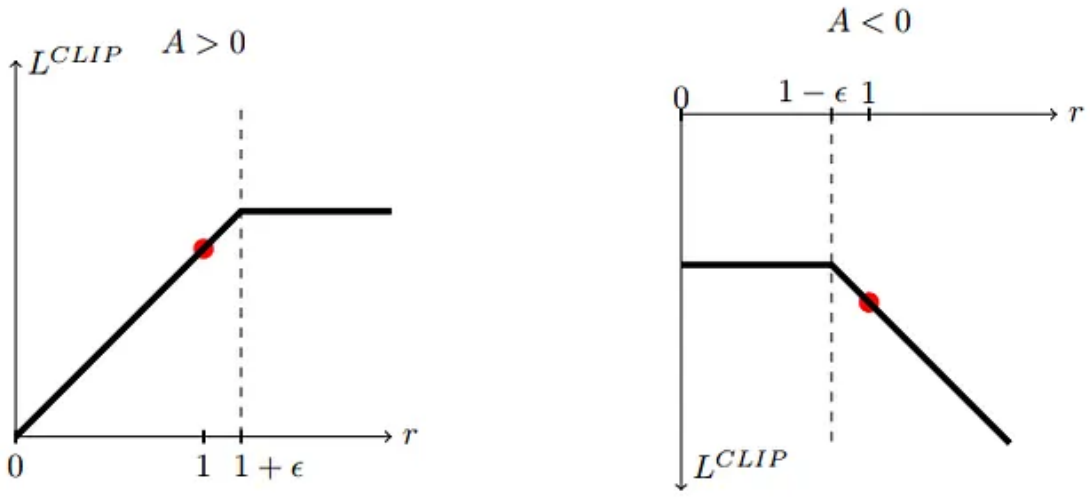

Figura 2.9: Ejemplo de la función de clipaje (Schulman et al. (2017))

Hace uso, al igual que otros algoritmos parecidos, de un sistema de "Crítico-Actor". El crítico es el encargado de estimar el valor de recompensa futura esperado para cada estado, y es este valor el que se usa para actualizar la política del actor. Mientras, el actor es el que toma la decisión sobre qué acción tomar.

A diferencia de DQN, este es un método estocástico que explora el espacio de políticas de acción. Es decir, en lugar de predecir una recompensa esperada para cada par estado-acción, asigna una probabilidad a cada una de las acciones posibles, donde la acción con más probabilidad es la que se espera que devuelva mayor recompensa.

2.5.4. El problema de la exploración

Un problema conocido en el campo del aprendizaje por refuerzo, es el de exploración frente a explotación. Durante el entrenamiento, ¿debería el agente elegir la acción que más recompensa espera generar, o quizás probar con opciones menos exploradas y que por tanto tienen la capacidad de convertirse en el nuevo óptimo? Un mal equilibrio entre ambas puede producir que el agente nunca aprenda.

Existen varios criterios para encontrar un término medio entre exploración y explotación. Algunas de las más importantes serían:

- Epsilon-greedy: Es la más común. Se basa en elegir con una probabilidad de 1-epsilon la mejor opción, y con una probabilidad de epsilon una aleatoria. De esta manera, cuando se toma la opción aleatoria se permite que se exploren estados que de otra manera ya se habrían descartado.

- Softmax: Se asigna una probabilidad de seleccionar cada acción posible en función de su valor Q predicho. Esto favorece explorar más frecuentemente estados no teóricamente óptimos frente a epsilon-greedy. Esta es la que vamos a usar ya que la anterior no es compatible con el algoritmo de PPO.

- Política aleatoria: No da un buen resultado pero se utiliza para comparar con otras políticas.

Aunque la aleatoriedad y la exploración son partes fundamentales del aprendizaje por refuerzo, para que la exploración de estados se traduzca en un aprendizaje, es necesario que el agente reciba una recompensa ya sea positiva o negativa. Existen casos en los que este no es el caso, donde el problema a resolver es difícil de evaluar

o devuelve una recompensa neutra en la mayoría de casos.

Podemos inspirarnos en la psicología humana para paliar este problema usando un tipo especial de recompensas, las recompensas intrínsecas.

2.5.5. Recompensas explícitas e intrínsecas

Las recompensas miden el desempeño del agente al realizar una tarea, pero se pueden elegir de distintas maneras y estas pueden presentar ventajas y desventajas. Podemos distinguir entre dos grupos importantes: explícitas e intrínsecas. Esta distinción se puede hacer también fuera del mundo de la informática.

La primera, las recompensas explícitas, son recompensas fácilmente identificables dentro de su ámbito. En los videojuegos, podría tratarse de la puntuación acumulada, mientras que en un ámbito laboral, se trataría del sueldo.

Por otro lado, las recompensas intrínsecas, son más sutiles y están de alguna manera inherentemente integradas en la tarea. Por ejemplo, en videojuegos, la curiosidad puede actuar como recompensa. Descubrir zonas ocultas o nuevos niveles suele ser un gran motivador para los jugadores. En el ámbito académico, puede ser el propio gusto por aprender.

De este enfoque inspirado en la psicología humana (Oudeyer y Kaplan (2007)) podemos sacar ideas, para enriquecer las funciones de recompensa en problemas de aprendizaje por refuerzo.

> "Al principio, un bebé está muy interesado en las partes de su cuerpo y empieza a jugar con las manos y los pies. A medida que el bebé crece y adquiere experiencia, las partes del cuerpo le resultan menos excitantes y se centra más en los objetos de su entorno. También queremos que un agente sea curioso, por lo que elige las acciones que le llevan a estados en los que es más inusual" (Daaboul (2020)).

Las recompensas intrínsecas pueden ser especialmente útiles para resolver el problema que surge en ámbitos donde las recompensas explícitas son escasas. Este escenario es llamado problema de "exploración difícil".

2.6. Hard-exploration problem

Un problema relacionado con el problema de explotación frente a exploración es el llamado “hard-exploration problem” o problema de exploración difícil en español. Este problema se refiere a los escenarios en los que existen pocas recompensas, o estas son confusas, contradictorias o impredecibles (Weng (2020)).

Un ejemplo concreto de este problema es el videojuego "Montezuma’s revenge"que, a diferencia de la mayoría de juegos de la consola Atari, aún no se ha conseguido crear un modelo que juegue de manera consistente. Esto es así porque se trata de un juego de exploración, del sub-género metroidvania, donde es necesario obtener objetos o habilidades para poder acceder a zonas del mapa.

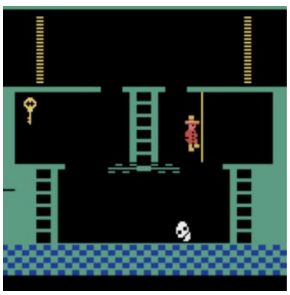

Figura 2.10: Fotograma del videojuego Montezuma’s revenge (elaboración propia, 2023)

Para solucionar este problema está la posibilidad de enriquecer las funciones de recompensa con recompensas intrínsecas. Existen varias maneras de calcular estas recompensas: modelos predictivos basados en recompensar el conocimiento del modelo sobre el entorno(Oudeyer et al. (2007)), y modelos que favorecen la novedad, que es en la que este trabajo se centrará.

La manera más fácil de explotar la recompensa intrínseca de la novedad es llevar cuenta de los estados ya visitados para dar un extra cuando se encuentre uno nuevo. Esto sin embargo es un enfoque demasiado simple puesto que no funcionaría con inputs de alta dimensionalidad o con entradas continuas.

Una aproximación al problema que sí surtió efecto fue la aproximación por Tang et al. (2016) en “Exploration: A Study of Count-Based Exploration for Deep Reinforcement Learning” donde prueba una nueva técnica que hace posible la discretización y por tanto cuenta de los estados. Lo hace utilizando un tipo de función Hash especial llamada LSH o local-sensitive hashing en inglés. Este hash agrupa automáticamente

los estados similares entre sí a hashes concretos, y se aplica

$$r^i(s) = N(hash(S))^{-1/2}$$

para calcular la recompensa, donde N es la cuenta del número de ocurrencias y s es el estado.

2.7. Frameworks y herramientas

Habiendo explicado ya la parte más teórica, pasamos ahora a explicar las tecnologías relevantes usadas en el desarrollo del proyecto, con una pequeña explicación y para qué se han usado. Muchas de ellas son herramientas muy extendidas por lo que no se entrará en profundidad en la mayoría de ellas.

2.7.1. Python

Para realizar el trabajo se ha decidió usar Python. Python es un lenguaje de programación interpretado, multiparadigma y de alto nivel que se ha convertido en el lenguaje por excelencia para proyectos de inteligencia artificial, ciencia de datos, machine learning, big data... por la gran cantidad de bibliotecas y herramientas disponibles.

Algunas de las bibliotecas más populares para desarrollar proyectos de ML, son Tensorflow y Pytorch. Ambos tienen sus propias ventajas y desventajas. En el caso de este proyecto, se comenzó usando Tensorflow en las primeras pruebas, pero por comodidad y problemas a la hora de guardar los modelos, rápidamente se cambió a Pytorch.

La gestión de los paquetes y librerías de este proyecto se ha realizado con Miniconda. Es una distribución del sistema de gestión de paquetes de Conda. Se usa para crear los llamados .entornos virtuales", que consisten en instalaciones aisladas de python con su propio intérprete, librerías y paquetes. Esto permite tener varios proyectos en el mismo ordenador sin que interfieran entre sí.

Miniconda también es útil para más tarde guardar la arquitectura de uno de los entornos virtuales para que otro desarrollador pueda recrearlo de cero con mayor

facilidad en su propio equipo.

2.7.2. Pytorch

Pytorch (pytorch.org) es uno de los paquetes para el desarrollo de modelos de redes neuronales más populares en la actualidad. Es además de código abierto. Es una herramienta intuitiva y fácil de aprender, con gran cantidad de recursos online que tomar como apoyo.

Ofrece la capacidad de crear los modelos de redes de manera aditiva, añadiendo capa sobre capa, y automáticamente se encarga del funcionamiento interno, entre las que se encuentran:

1. Construir grafos dinámicos con la estructura de la red neuronal
2. Cálculo automático de los gradientes en la fase de backpropagation para entrenar la red neuronal
3. Recogida de métricas

También está preparado para trabajar con GPUs para acelerar enormemente el entrenamiento ya que de otra manera tardaría demasiado.

2.7.3. Numpy

Numpy (numpy.org) es otra de las bibliotecas indispensables de python. Extiende inmensamente la capacidad nativa del lenguaje y sus listas con matrices numpy. Contiene una serie de herramientas para crearlas, modificarlas y realizar cálculos sobre ellas de manera muy eficiente. Al ser una herramienta tan popular, está perfectamente integrada con otras librerías, como pytorch.

Ha sido utilizada para guardar en formato matricial los inputs de las redes neuronales, así como otro tipo de imágenes

2.7.4. Matplotlib

Matplotlib (matplotlib.org) es usada para la visualización de datos en multitud de formas: gráficos, historiogramas, diagramas de dispersión... Es otra de las librerías más usadas en el campo de la ciencia de datos por su facilidad de uso y personalización. También tiene buena integración con otras librerías como Numpy.

En el caso de este proyecto, se ha usado para representar unas gráficas relativamente simples de la función de recompensa y las recompensas obtenidas.

2.7.5. Imagehash

Imagehash (github.com/JohannesBuchner/imagehash) es ya una librería más concreta respecto a este proyecto. Es una serie herramientas para calcular el hash de una imagen.

En otros contextos, las funciones hash están preparadas para producir resultados muy dispares ante cualquier variación del input. Sin embargo, este conjunto de hashes pretenden lo contrario: que imágenes similares resulten en un hash similar. Es, por ejemplo, el sistema usado por Google Imágenes (Cloud (2017)) pero también tiene usos como la detección de fraude o imágenes ilegales.

Algunos de los usos posibles para esta librería es la creación de una base de datos de imágenes eficiente que busque en función de los hashes en lugar de la imagen completa. También es útil para la comparación de imágenes y calcular un grado de similitud o clasificar imágenes.

2.7.6. Gym

Es una librería desarrollada por OpenAI para desarrollar, evaluar e interactuar con algoritmos de aprendizaje por refuerzo. Gym (gymlibrary.dev) puede crear una serie de entornos o -enviroments- que hacen de caja de arena para que el agente a entrenar interactúe con ellos. También contiene algunos ya creados para problemas clásicos como el control de partes robóticas o algunos juegos.

Una vez creado el entorno, se permite al usuario obtener información sobre este como por ejemplo en el caso de un videojuego, la pantalla o la memoria RAM.

Existe una librería que hace de complemento a Gym llamada Retro, que expande las capacidades de Gym para interactuar con videojuegos retro, así como poder cargar las imágenes de juego de múltiples plataformas.

2.7.7. Github

Se trata de una plataforma de control de versiones. Aunque gran parte de su funcionalidad se basa en el desarrollo colaborativo entre equipos de trabajo, al realizarse este trabajo individualmente, se ha usado con el único motivo de guardar un backup del trabajo y facilitar el reanudar el trabajo desde otro equipo.

2.7.8. Visual Studio Code

Visual Studio Code es un IDE gratuito y de código abierto y es popular por la gran cantidad de lenguajes de programación que soporta, entre los que se encuentra Python. Esto lo hace gracias a un enfoque de extensiones que añaden características y funcionalidades nuevas como depuración en vivo, autocompletado, atajos de teclado o modificar la interfaz.

Es el IDE que se ha usado para programar en este proyecto junto con las extensiones típicas de python.

2.7.9. Memoria

La Memoria se ha generado usando una plantilla de LaTeX. LaTeX es un lenguaje de marcado de texto muy usado en el ámbito académico por su alta calidad y consistencia de presentación a la hora de crear artículos, documentos científicos, tesis etc. Además permite describir de manera precisa elementos como tablas y fórmulas matemáticas que tan importante son en este ámbito. Se pueden estructurar los proyectos en diferentes archivos y especificar la manera en la que se monta el archivo final, facilitando la organización e incluso la colaboración entre integrantes de un mismo grupo

LaTeX es tan solo un lenguaje y necesita de un editor. La elección ha sido Overleaf por las ventajas que ofrece. Es un editor en línea por lo que no hace falta instalar

nada. Aunque este trabajo sea individual, cuenta adicionalmente con soporte para edición múltiple por parte de varios usuarios, junto con guardado y control de versiones automático.

2.7.10. Equipo

Para este tipo de proyectos de machine learning es importante tener un equipo suficientemente potente para poder realizar los entrenamientos a un ritmo razonable y acelerar las iteraciones. Una solución popular es google Colab, que ofrece computación en la nube gratuita y con acceso a aceleradores (GPUs) muy potentes.

Sin embargo, las libretas de google colab tienen el entorno gráfico desactivado, lo que da problemas a la hora de renderizar algunos elementos de la simulación y dificulta mucho comprobar su desempeño real. Por este motivo, y dado que yo personalmente dispongo de un computador relativamente potente, se ha decidido realizar los entrenamientos de manera local. El equipo en el que se ha entrenado tiene las siguiente especificaciones: Ryzen 7 5800x, 16GB RAM, NVIDIA GTX 1070.

Capítulo 3

Agente entrenado con recompensas intrínsecas

La clave principal de este trabajo es desarrollar un agente que juegue al videojuego “Super Mario Land 2” a partir de únicamente observaciones de la pantalla y sin hacer uso de valores de la memoria RAM, utilizando las llamadas recompensas intrínsecas.

Un enfoque, sería detectar patrones determinados en la pantalla y tomarlos como recompensas explícitas. Por ejemplo, detectar el marcador de puntuación, leerlo, y tomar el valor. Sin embargo, esto se podría decir que es una implementación "poco inteligente", ya que además sería dependiente del juego.

Otro enfoque sería tomar las muestras de la pantalla como estados y, usando la técnica de çounting after hashing", medir la novedad de las imágenes. Se trataría de usar la recompensa intrínseca de explorar como incentivo para el agente.

Este enfoque es adecuado para juegos en los que explorar es indicativo de que estés jugando bien. Cuantas más imágenes diferentes te encuentras, mejor lo estará haciendo y si no está cambiando, significa que te has atascado. Hay juegos donde esto no ocurre como por ejemplo juegos arcade que ocurren siempre en la misma pantalla.

El videojuego “Super Mario Land 2” es uno de los juegos en los que encontrarte nuevos escenarios está relacionado con jugar bien, y por este motivo se ha elegido para realizar la mayoría de pruebas. Además, este juego, al ser originario de la consola Game Boy, funciona a una resolución baja y funciona con solo un canal de

color, es decir, escala de grises, que ayudará a que la red neuronal sea más pequeña y por lo tanto más fácil de entrenar. Concretamente, la consola devuelve una imagen de 160 x 144 píxeles y trata los colores como 2 bits, dando un total de 4 posibles colores.

Super Mario Land 2 es un videojuego de scroll lateral. Es decir, controlas un personaje que se puede mover hacia los lados y saltar, cuyo objetivo es navegar un nivel hasta llegar al final. El final del nivel en los videojuegos de Mario normalmente está a la derecha de todo del mapa, pero en este juego en concreto hay un desarrollo menos lineal, habiendo niveles que hay que navegar hacia arriba o hacia abajo.

Figura 3.1: Portada Super Mario Land 2 (elaboración propia, 2023)

Los controles son muy sencillos. La consola cuenta con 4 botones de dirección, botón A, botón B, botón Start y botón Select. De los 4 botones de dirección, los dos de los lados mueven el personaje a cada lado, y el botón A lo hace saltar. También se pueden hacer combinaciones de botones. Esta información es importante de detallar ya que se debe establecer qué botones el agente puede presionar y cuáles no.

Aunque el formato que tanto el videojuego como la plataforma retro gym nos proporciona sea muy conveniente, debe pasar un preprocesamiento previo para hacer el entrenamiento más eficiente como veremos en la siguiente sección.

3.1. Preprocesamiento

A partir de la imagen se realizan una serie de transformaciones por motivos tanto de eficiencia, añadir información, y realismo en comparación con un jugador humano. Este proyecto maneja dos sistemas en paralelo, la función de recompensa y la red neuronal. Ambos sistemas tienen un preprocesado muy similar, pero no exactamente igual.

En el caso de la red neuronal, la primera transformación que se realiza es apilar 4 fotogramas en paralelo. Esto se realiza para incluir una cierta noción temporal. Un único fotograma no es suficiente para saber la dirección y velocidad de los elementos en pantalla, información crucial en un videojuego en el que existen enemigos y elementos que esquivar. Este hecho se puede observar con facilidad en la figura X.

A continuación se redimensionan a un tamaño concreto, en este caso 84 x 84. Al ser más pequeño, facilita el entrenamiento de la red, pero con un tamaño suficiente como para mantener la información visual necesaria para jugar.

Falta normalizar los canales de color a un solo canal con valores entre 0 y 255. Aunque el sistema funcione en escala de grises, la librería de Gym devuelve los datos en formato RGB por el hecho de estar pensada para funcionar con más variedad de consolas.

Finalmente, se elimina uno de cada 4 fotogramas. Esto es así puesto que la red se ejecuta una de cada 4 veces para acelerar el entrenamiento y la ejecución. Además tomar una decisión o lanzar un input diferente en cada uno de los fotogramas no es realista si se compara con un jugador normal.

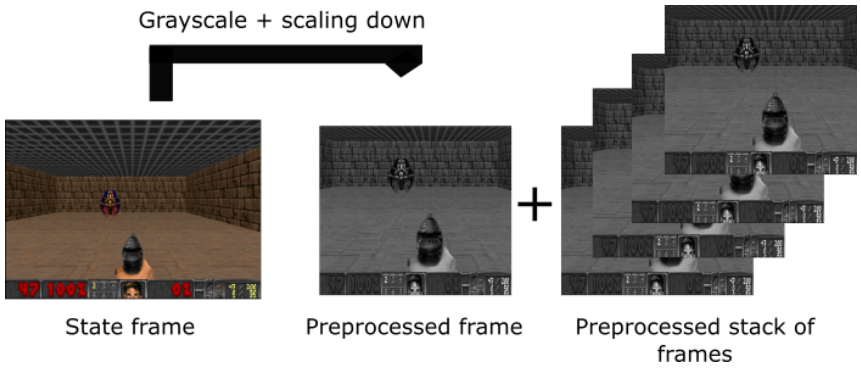

Figura 3.2: Ejemplo de preprocesamiento (Wong (2023))

En el caso de la función de recompensa, el preprocesado es el mismo exceptuando

el salto de fotogramas, ya que provocaría un funcionamiento erróneo al no recordar correctamente todos los eventos y estados.

3.2. Función de recompensa

En el caso de este proyecto, la función de recompensa no tratará de seleccionar un valor concreto, sino de calcularlo en función de estados pasados dentro del mismo episodio de entrenamiento. Por lo tanto, es necesario crear un sistema que almacene la información necesaria para el cálculo.

El preprocesamiento es el explicado en el apartado anterior, pero a continuación se usa una función hash sobre la imagen usando la librería Imagehash. El hash es importante dado que el objetivo es detectar nuevos estados, pero dado un cierto margen de variabilidad. En condiciones normales, si dos imágenes son iguales pero cambia uno solo de los píxeles, se detectaría como imagen nueva. Esto haría inviable la estrategia y por lo tanto es necesario discretizado la dimensión a un tamaño mucho más pequeño.

A continuación, se lleva la cuenta de cuantas ocurrencias han habido de ese hash durante este episodio. La cuenta de todos comienza en 0. Se toma la cantidad de veces que ha aparecido ese hash y se calcula la recompensa con la siguiente fórmula:

$$r^i(s) = N(Hash(s))^{-1/2}$$

. De esta forma se da más recompensa cuantas menos veces haya salido ese hash.

Saber la efectividad de una función de recompensa es complicado sin pasar por un entrenamiento completo que puede durar decenas de horas. Por lo tanto se ha tomado la siguiente estrategia: Se han grabado dos partidas manualmente. Una jugando lo mejor posible, es decir, completando el nivel lo más rápido posible, esquivando enemigos, etc. y otra jugando "mal". Jugar mal es algo difícil de definir, pero en general quedarte quieto o ser golpeado por enemigos se puede asociar a jugar mal.

Una vez grabados ambos vídeos, se ha desarrollado un script que recoge los frames del vídeo y los evalúa con la misma función de recompensa que en el entrenamiento. La recompensa se acumula y se calcula la recompensa media por frame, ya que el valor absoluto no es representativo por la diferencia de longitud entre los vídeos. Una buena función de recompensa asignará una recompensa media alta al vídeo con

la partida buena y una recompensa media baja a la partida mala.

Ahora es posible buscar hiperparámetros como el tamaño y tipo de hash que mejor entrenamiento proporcionará en principio. El tamaño de hash marcará la sensibilidad de la función a estados nuevos. A mayor tamaño, más sensibilidad. También existen distintos tipos de hash que tiene cada uno sus propias características. Si probamos el script para encontrar los mejores valores encontramos los siguientes resultados:

bits	**Partida buena**	**Partida mala**	**Diferencia**
2	0.0564	0.04183	0.0146
4	0.4898	0.3309	0.1589
8	0.8558	0.811	0.0448

Tabla 3.1: phash comparando ambos vídeos para distintos tamaños de hash (elaboración propia, 2023)

bits	**Partida buena**	**Partida mala**	**Diferencia**
2	0.1334	0.1033	0.0301
4	0.6556	0.5401	0.1155
8	0.8743	0.8273	0.047

Tabla 3.2: dhash comparando ambos vídeos para distintos tamaños de hash (elaboración propia, 2023)

bits	**Partida buena**	**Partida mala**	**Diferencia**
2	0.0791	0.0581	0.021
4	0.4136	0.298	0.1156
8	0.8743	0.8273	0.0554

Tabla 3.3: whash comparando ambos vídeos para distintos tamaños de hash (elaboración propia, 2023)

Se puede observar que en general 4 bits es el número ideal, ya que es el que arroja mayor diferencia entre ambos vídeos. En cuanto al tipo de hash, el phash es el que mejor resultado obtiene, aunque también es con diferencia el más lento de todos (content blockchain (2019))

3.3. pHash

El pHash es un tipo de hash perceptual y uno de los hashes incluidos en la librería ImageHash de python. Es un tipo de hash que está construido para ser resistente a ataques de rotación, cambio de color, o diferentes tipos de compresión, lo que lo hace ideal para nuestro objetivo.

El proceso para producir este tipo de hash es el siguiente:

1. Reducir escala a un tamaño concreto. Normalmente 32x32.
2. Reducir el color a escala de grises.
3. Computar la transformada de coseno discreta para reducir las frecuencias más altas.
4. Recortar el resultado de la transformada a una ventana de 8x8 que contenga la información de las frecuencias bajas.
5. Calcular el valor medio.
6. Formar el hash bit a bit, marcando 1 en los casos donde la transformada devuelva un valor mayor que la media y 0 en el caso contrario.

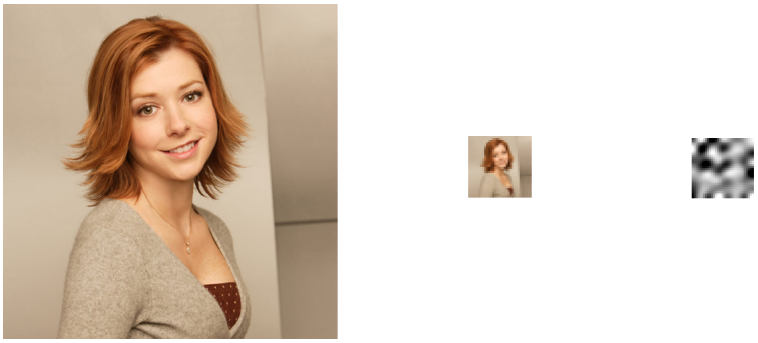

Figura 3.3: Ejemplo de aplicar el pHash a una imagen y su resultado (Krawetz (2011))

La transformada de coseno discreta (o DCT por sus siglas en inglés) es una técnica matemática que sirve para transformar una señal al dominio de la frecuencia. Es especialmente útil en el ámbito de imagen y vídeo ya que estos normalmente acumulan la mayoría de la información en frecuencias bajas, pudiendo desechar las frecuencias altas como información redundante y reduciendo la cantidad de información a almacenar.

3.4. Arquitectura

A continuación se explicará la estructura de la red neuronal convolucional usada en este trabajo. La arquitectura elegida es la misma estructura que usaron en Mnih et al. (2013), que ya ha demostrado funcionar muy bien para este mismo caso de uso. El diagrama de la red es el siguiente:

```
nn.Conv2d(in_channels=4, out_channels=32, kernel_size=8, stride=4),
nn.ReLU(),
nn.Conv2d(in_channels=32, out_channels=64, kernel_size=4, stride=2),
nn.ReLU(),
nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, stride=1),
nn.ReLU(),
nn.Flatten(),
nn.Linear(3136, 512),
nn.ReLU(),
nn.Linear(512, env.action_space.n)
```

Figura 3.4: Arquitectura usada (elaboración propia, 2023)

La primera capa se trata de una capa convolucional con una imagen de entrada de 84x84 píxeles y 4 canales, uno para cada frame. Utiliza un kernel de 8x8 para la convolución, produciendo una salida de 32 canales. Está seguida de una capa ReLU (Rectified Linear Unit), que aplica la función de activación del mismo nombre. Muy utilizada en redes neuronales convolucionales por su eficiencia, al ser no lineal, otorga a la red la capacidad de aprender relaciones no lineales entre la entrada y la salida, imprescindible en este tipo de tareas.

Estas dos capas se repiten dos veces más con diferentes tamaños de kernel para poder así detectar características más complejas y abstractas de la imagen de entrada. Hasta este punto, se puede decir que esta primera mitad de la red neuronal es la encargada de realizar el procesamiento visual de la imagen de entrada y de aprender a detectar patrones relevantes.

A continuación, la capa "Flatten", aplana la salida de la capa anterior y la convierte en un vector unidimesional. Esto es un paso necesario para la siguiente capa, pero no realiza ninguna transformación numérica real.

El siguiente paso es una capa "Linear"que es una capa donde todas las neuronas de la capa anterior están conectadas con todas las neuronas de la capa siguiente, y se activa una vez más con una capa ReLU. Se completa la red neuronal con una última capa "Linear". Estas 3 capas son la segunda mitad de la red neuronal y tienen la misión de aprender a relacionar los patrones detectados por la parte anterior con las acciones que devuelvan mayor recompensa.

La última capa tiene una salida de un solo elemento en el caso del crítico, pues su misión es evaluar el frame de entrada, mientras que el actor tendrá una salida para cada una de las posibles acciones que puede realizar.

3.5. Entrenamiento

El entrenamiento consiste en una sucesión de episodios en los que se evalúa al agente y se recoge las recompensas obtenidas, inputs, y acciones tomadas en cada ocasión. Al acabar cada episodio, el script informa de cuál ha sido la recompensa acumulada obtenida. Se espera que a medida que avanzan los episodios, este valor vaya incrementando.

Además, cada 10 episodios, se imprime la media de estos últimos episodios junto con una gráfica donde se puede observar el histórico de la recompensa obtenida desde el comienzo del episodio.

El sistema dispone de una serie de hiperparámetros, los habituales en este tipo de experimentos. Algunos de los más importantes serían:

1. gamma: Es el factor de descuento en relación al peso relativo entre las recompensas inmediatas y las recompensas futuras. Es un valor entre 0 y 1, donde 0 hace que el agente se centre en recompensas inmediatas y el 1 sería lo contrario. El valor elegido es 0.95, que es lo habitual.

2. lambda: Similar a gamma, regula la importancia relativa de recompensas pasadas. También se ha usado 0.95.

3. worker-steps: El número máximo de pasos en la simulación, o lo que es lo mismo, el número máximo de fotogramas.

4. Learning-rate: También llamada tasa de aprendizaje, es la escala a la que se actualizan los pesos de la red neuronal en la etapa de retropropagación. Un valor bajo hace que el modelo pueda tardar demasiado tiempo en converger y aumentará el tiempo de entrenamiento. Por otro lado, un valor muy alto puede llevar a actualizaciones demasiado grandes y que no se encuentre el mínimo de la función de recompensa. Se ha usado un valor de 0.001 para el crítico y un valor de 0.00025 para el actor.

5. Al configurar la simulación también existen dos valores de implementación propia que se encargan terminar el episodio de manera prematura. Si la media de las últimas N recompensas es menor que un valor dado que normalmente es 0.1, se termina el episodio y se genera una recompensa negativa. Normalmente esto no sería necesario si se tuviese acceso a la memoria RAM, pero esto es una buena manera de detectar si ha perdido o si se ha quedado atascado.

Otro detalle importante sobre el entrenamiento es la decisión de qué botones permitir que presione. En una situación ideal se le permitiría pulsar todos ellos y todas las combinaciones. Sin embargo, demasiadas opciones ralentiza el entrenamiento y en ocasiones puede imposibilitarlo completamente.

En el caso del juego seleccionado, es suficiente con habilitar el botón de salto y de derecha. Todos los demás se incluyen en pos de demostrar su capacidad de aprendizaje. Por ejemplo, el botón de izquierda es completamente irrelevante en este primer nivel, sin embargo sirve para demostrar que el agente es capaz precisamente de aprender eso.

El botón START es un caso especial, ya que en la mayoría de juegos de la época sirve para pausar el juego. En esta época, la pausa no abría normalmente ningún menú, sino que añadía algún cartel a la pantalla o añadía algún efecto. Esto entorpecería mucho el entrenamiento y por ello no está habilitado el botón.

En cuanto a botones, en resumen, están activados los botones A y B, las 4 direcciones de la cruceta, y cada una de las combinaciones de dirección más A o B.

Puesto que el núcleo del trabajo no es tanto el desarrollo de un algoritmo de aprendizaje automático sino el explorar el comportamiento de la recompensa previamente explicada, se partirá de una implementación base del algoritmo PPO, el cual se

ajustará para funcionar con las librerías mencionadas y para usar la recompensa (Schneider (2021)).

Una vez explicados las configuraciones, pasamos a ver los resultados obtenidos.

Capítulo 4

Resultados

En los objetivos se explica que se va a realizar experimentos incrementalmente más complicados para el juego principal "Super Mario Land 2". La idea inicial era probar primero a permitir solo los botones de derecha y saltar, que son suficiente para completar el primer nivel, y poco a poco ir añadiendo botones. La motivación de esto era que no se sabía si realmente la implementación iba a funcionar. Sin embargo, los resultados de cada uno de los test fueron satisfactorios hasta llegar a probar con todos los botones. Como todos estos tests llegaron a la misma conclusión, se han omitido para explicar únicamente el más complejo, ya que ya incluye los tests más pequeños.

Muchos de los resultados son difíciles de evaluar de manera numérica y merece más la pena hacer un análisis más subjetivo de la manera en la que los agentes juegan. Por ello, además de esta explicación, se dejan el apéndice enlaces a vídeos con ejemplos.

A partir de las configuraciones explicadas en el apartado anterior, y teniendo en cuenta que se han realizado varias ejecuciones, se han obtenido los siguientes resultados:

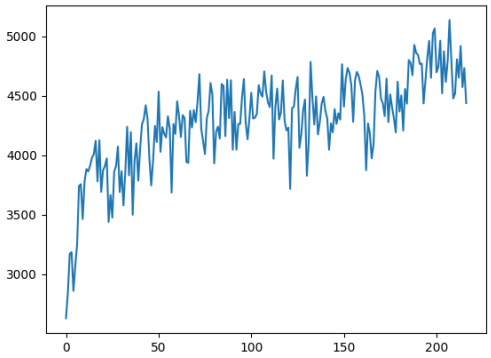

Figura 4.1: Media de recompensa de los últimos 10 episodios. (elaboración propia, 2023)

En la figura 3.5 se observa que el agente cumple su objetivo de maximizar la recompensa obtenida. Cabe destacar que la curva es tan irregular porque recordemos que el algoritmo de PPO es un algoritmo estocástico y por tanto elige las acciones por probabilidad, y esto puede llevarlo a no obtener la mejor recompensa posible en todos los episodios. Además esto puede suponer un problema en situaciones en las que sea necesario un input determinado durante varios fotogramas como por ejemplo durante un salto. Si en medio del salto, por mala suerte, se elige otra acción, el salto acabará prematuramente y puede hacer que pierda.

En cualquier caso, este trabajo no se puede evaluar directamente viendo una gráfica de recompensas puesto que el hecho de que maximice la recompensa obtenida no tiene porqué traducirse en que juegue bien si la propia función de recompensa no es correcta. Es por esto que gran parte de los resultados obtenidos son observaciones e interpretaciones propias y no tanto valores numéricos.

El primer nivel del juego está diseñado para funcionar de tutorial. Este nivel introduce los conceptos necesarios para jugar poco a poco. Al igual que lo haría un niño pequeño empezando a jugar por primera vez, el agente lo primero que aprende es a moverse hacia la derecha. A continuación se encuentra con el primer enemigo, contra el cual morirá y acabará aprendiendo a superarlo a base de prueba y error.

El agente se muestra capaz de completar el primer nivel en tan solo 1 hora de entrenamiento, y empieza a hacerlo de manera consistente en 2 horas. Cabe destacar que aunque el agente mejora muy rápido, nunca ha llegado a ser capaz de completar el nivel el 100 % de las veces. Esto puede tener varias explicaciones que se explorarán más adelante.

El agente es capaz de navegar el nivel esquivando enemigos, pero también toma caminos alternativos como tuberías y recoge monedas por el camino. En etapas tardías del entrenamiento, es capaz incluso de tomar la salida alternativa del final del nivel, donde es necesario una serie de saltos relativamente precisos para alcanzar un objeto elevado del suelo.

Una vez acabado el nivel, el agente es libre de hacer lo que quiera. Habitualmente, a partir de aquí explora el mapa. A veces se intenta ir a uno de los niveles finales, a los cuales todavía no tiene acceso y el juego no le deja continuar. Otras, se va a niveles más alejados donde comienza a jugar.

Aquí se encuentra el problema de que estos niveles son bastante diferentes al nivel inicial y no suele dar buenos resultados. Con suficientes intentos, se observa que el agente empieza a tener una cierta noción de cómo navegar estos niveles pese a que algunos requieren una manera completamente diferente de jugar.

El agente también aprendió a explotar los fallos de la función de recompensa en algunas ejecuciones. Estos ejemplos no devolvían más recompensa que otras ejecuciones mejores, sino que se trataba de un comportamiento cíclico que devolvía una recompensa elevada. Esto podría considerarse un mínimo local.

Figura 4.2: Mario derrotando un enemigo (elaboración propia, 2023)

Uno de estos ejemplos es el agente descubriendo que una vez muere, tiene la opción de salir del mapa y liberar la cámara, pudiendo moverla libremente. Entonces el agente empieza a moverla por encima del escenario y quedándose en las esquinas, donde la animación del mapa otorga suficiente variación como para satisfacer el sistema de recompensa. Algo parecido ocurrió en algunas ocasiones con la combinación del

movimiento cíclico de algunos enemigos y la cámara.

4.1. Interpretación de los resultados

Algunos de los resultados son destacables ya que de primeras uno podría pensar que en el mejor de los casos, el agente aprendería a navegar el nivel, pero no a hacer cosas opcionales al igual que lo haría un jugador normal. Sin embargo, lo que un jugador humano puede asociar a una recompensa explícita como coger monedas, se puede relacionar también con la recompensa intrínseca de ver que el escenario cambia y que tus acciones tienen consecuencias.

El agente no puede ver, ni entiende, que el contador de monedas ha subido en 1 cada vez que elige una, sin embargo al estarse optimizando una función que busca la mayor novedad de imagen posible, de manera indirecta aprende que coger monedas modifica el escenario y por tanto ofrece recompensa.

El mismo fenómeno se puede observar al romper bloques, excepto que en este caso esta acción tampoco da ninguna recompensa a un jugador humano y sin embargo es algo habitual reforzando la idea de que la recompensa intrínseca de la novedad es un factor importante en los humanos.

Las zonas opcionales también son llamativas para el agente puesto que normalmente tienen monedas o, en el caso de la zona extra del final, animaciones que devuelven la recompensa máxima durante varios fotogramas.

Ya hemos comprobado que efectivamente el sistema preparado es capaz de entrenar un agente haciendo uso únicamente de recompensas intrínsecas, pero recordemos que uno de los motivos por el que este enfoque era interesante es también que lo hace completamente independiente del juego en cuestión, mientras este comparta determinadas características de imagen y jugabilidad. A continuación se harán pruebas con otros juegos similares para comprobar si cumple este objetivo.

4.2. Otras pruebas

Para realizar estas pruebas se han seleccionado los videojuegos “Kirby’s Dream Land” y “Pokémon Red”. Ambos son similares a Super Mario Land 2 pero con

suficiente variación como para que merezca la pena hacer las pruebas. El agente debería aprender a navegar el juego independientemente de sus controles y, en caso de no poder avanzar, es interesante buscar la explicación de por qué y su relación con un jugador real.

4.2.1. Kirby's Dream Land

Este juego pese a tener una navegación similar, dispone de más controles como volar, absorber proyectiles enemigos o devolverlos. También existen elementos como puertas que trasportan al jugador a otra pantalla completamente diferente y normalmente opcional, que tienen el objetivo de obtener puntuación extra. El agente es capaz de aprender a navegar el primer nivel de este juego. Aprende con relativa facilidad a ir a la derecha y a esquivar enemigos.

Sin embargo el primer nivel contiene una pelea contra un enemigo fuerte o "jefe" no es capaz de vencerle. Este enemigo no es ignorable ya que la cámara queda fija. El problema surge de que la pelea es gráficamente poco llamativa. El jugador se supone que debe devolver los proyectiles del enemigo contra él 3 veces. Por simple probabilidad el agente llega a hacerlo alguna vez, pero el único cambio gráfico que produce es la disminución de la barra de vida del enemigo de apenas unos píxeles. Como el sistema no va a asignar recompensa a cambios tan pequeños, el agente nunca va a aprender a ganar esta batalla.

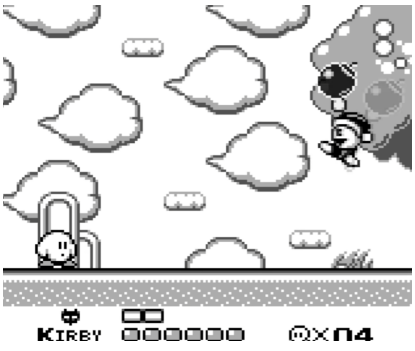

Figura 4.3: Kirby contra el jefe (elaboración propia, 2023)

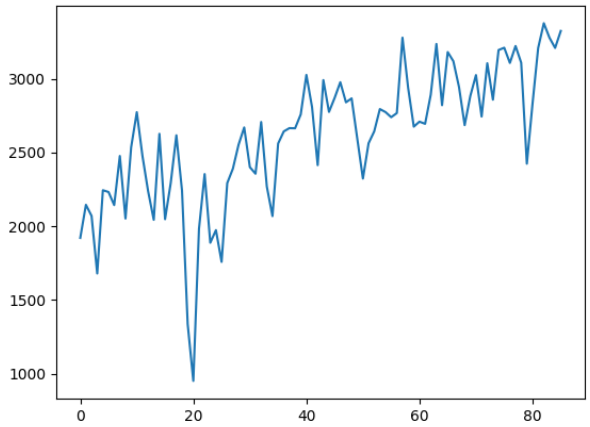

Figura 4.4: Recompensa obtenida a través de los episodios en Kirby (elaboración propia, 2023)

Este comportamiento es fácil de explicar y esperado, puesto que una persona real que no esté familiarizada con los videojuegos, tampoco sabría qué hacer en esta situación. Sin conocimientos como qué es una barra de vida, que hay enemigos que requieren más de un golpe para ser vencidos o que es habitual tener que hacer algo 3 veces.

4.2.2. Pokémon Red

Este juego supone un reto interesante puesto que de primeras, a diferencia del resto de juegos, no hay una manera clara de medir su progreso. Tanto en Super Mario Land como en Kirby's Dream Land, se puede relacionar el desempeño con la distancia recorrida hacia la derecha. Sin embargo este juego dispone de estructuras en las que se debe entrar, personajes con los que se debe hablar e interacciones relativamente complejas, además de todo un sistema separado de combates. ¿Cómo calificar si lo está haciendo bien sin saber de antemano el camino correcto?

Con el método estudiado en este trabajo el agente es, sorprendentemente, capaz de encontrar el camino correcto para seguir la historia principal del juego dentro del pueblo inicial: Salir de su cuarto, de su habitación, intentar salir del pueblo para activar un evento que le llevará a elegir su primer Pokémon y su primer combate.

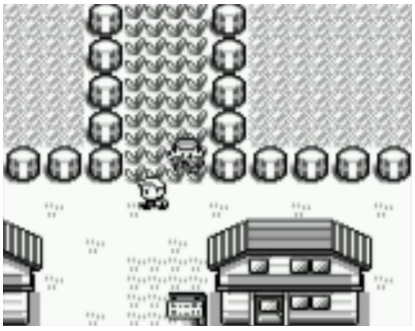

Figura 4.5: Agente explorando en Pokémon (elaboración propia, 2023)

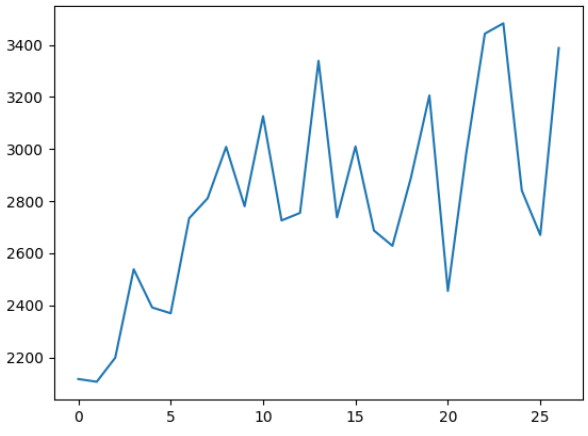

Figura 4.6: Recompensa obtenida a través de los episodios en Pokémon (elaboración propia, 2023)

Es una vez dentro del primer combate donde el agente tiene dificultades para continuar. Gráficamente, los combates no tienen suficiente variedad como para interesar

al agente y esto pone en el foco un problema del método. El sistema de recompensas se fija en la imagen general y no en porciones. Para un jugador normal, los diferentes Pokémon, animaciones y ataques serían algo interesante pese a ser algo que en relación a la pantalla es algo pequeño.

Por último, este juego también revela otro problema: backtracking. Pokémon Red es un juego que requiere backtracking, es decir, volver sobre tus pasos para realizar algo que antes no podías hacer en una sección determinada del mapa. Esto, tal y como está construida la arquitectura, es imposible que lo aprenda ya que requiere que dados 2 inputs exactamente iguales tenga que tomar decisiones diferentes. Sería necesario una arquitectura que mantuviese un estado interno, como una red recursiva, para que aprendiese a recordar acciones pasadas, pero esto dificultaría el entrenamiento enormemente.

Capítulo 5

Trabajo Futuro y Conclusiones

5.1. Trabajo futuro

Como último capitulo repasaré las posibles mejoras que se pueden hacer a este trabajo teniendo ya en cuenta los resultados obtenidos.

Un tema pendiente, y quizás el más evidente, es el aplicar el método aplicado a otros juegos, probando también a combinar las recompensas intrínsecas con otras más convencionales, y medir los resultados. Como ya se ha explicado anteriormente, existen juegos donde no es aplicable y no se trata de si son mejores o peores, sino de su necesidad, pero puede ser interesante estudiar en qué medida mejoran el desempeño en otras situaciones.

Otra mejora importante al método sería aplicar otro tipo de hash o un método similar para medir mejor el nivel de novedad de los estados. Actualmente el entrenamiento no premia cambios de pocos píxeles, pero en determinados juegos el cambio de unos pocos píxeles puede contener mucha información. Por este motivo, auto-encoders o detección de patrones nuevos pueden contribuir a mejorar la función de recompensa.

Relacionado con el punto anterior sobre la importancia de algunos píxeles, y teniendo en cuenta la opacidad de sistemas de redes neuronales, también puede ser interesante aplicar técnicas de explicabilidad. Un ejemplo de esto y seguramente el que más sentido tenga, es generar mapas de atención o de calor a partir de la imagen de entrada, para poder evaluar qué secciones de la imagen están contribuyendo a tomar cada acción. Esta información es interesante por que nos puede indicar en qué se basa

el agente para actuar, pero también qué elementos está ignorando. Un ejemplo de esto sería en el caso de Pokémon Red donde no parece saber a qué prestar atención en los combates.

Por último, y también la mejora más difícil, sería mejorar las capacidades de los agentes dotándolos de recursos para recordar eventos pasados, o al menos mejorar su capacidad de relacionar inputs dentro de la serie temporal. Para esto se pueden añadir bancos de memoria a los agentes a cambio de eficiencia o añadir estructuras recurrentes a las redes neuronales a cambio de un entrenamiento mucho más lento. En muchos videojuegos, como Super Mario Bros, esto no es necesario ni aportará nada, pero en otros como Pokémon, sería un requisito indispensable puesto que hay información que no está siempre en pantalla y que el jugador debe recordar.

5.2. Conclusiones

En este trabajo se ha evaluado la importancia de las recompensas intrínsecas en algunos juegos. Se ha entrenado a agentes usando el Deep Learning y el aprendizaje por refuerzo para que aprendiesen a jugar a videojuegos, y se ha podido observar que este enfoque funciona sorprendentemente bien incluso sin la necesidad de recompensas más convencionales.

Los agentes entrenados han excedido las expectativas. En el juego principal, Super Mario Land 2, ha completado con facilidad el primer nivel y ha dado el sorprendente resultado de incluso tomar caminos opcionales y puntos de guardado en el proceso. En general, se podría decir que los agentes muestran un comportamiento "más humano.ªunque esto sea algo completamente subjetivo.

Por otro lado, incluso cuando los agentes se han encontrado con algún obstáculo que no han conseguido aprender a superar, una observación más detenida del agente arroja explicaciones razonables que no hacen más que reforzar la idea de que este tipo de recompensas son positivas y acotando mejor sus ventajas e inconvenientes.

Chapter 6

Introduction

Reinforcement learning is a branch of machine learning focused on training an agent to make the right decisions in a given scenario. This technique has proven to be effective in solving problems that require skill and adaptation, such as video games.

Video games are the ideal challenge for testing algorithms and new techniques not only in artificial intelligence, but also in reinforcement learning, since they are a controlled environment with a multitude of variables and rules to learn from. In addition, it is easy to interact with them to extract information from the system. The main memory or RAM contains all kinds of useful information such as coordinates, scores, timers, lives and other variables that mark the state of the game at any given time.

However, using the RAM data for training is not the way a human would learn to play, the human would take all the information from the screen. Besides there are games where it is not possible to measure progress numerically, games where exploration is an important factor. Let's compare two concrete examples in two different video games:

The video game "Tetris" is a game of the arcade genre that consists of stacking a series of blocks of different shapes to form rows. It's easy to extract values that indicate the performance of a player, such as the number of rows filled.

An example of the opposite case would be a game like "Pokemon Red", a role-playing video game where, for most of the game, you control a character that explores a predefined map. In this case, the RAM memory does not contain values that, at

least at first glance, provides any relevant information. There are no scores to mark progress, so measuring it is complicated and sometimes even subjective.

This is why using the image as input data is not only more realistic, but also allows training on different games without the need to distinguish between them.

There are several approaches to this problem, but a reasonable one is to exploit an element also found in humans: intrinsic rewards, in this case rewarding novelty. The idea is to apply a hash to each of the states, i.e. images, and count how many times that state is repeated. This technique is called "Counting after hashing".

In most games, the fact of finding new content, new graphics or maps is an indication that a good job is being done, even entire genres of video games have emerged from this concept. On the other hand, not moving or finding multiple times the same image, is associated with something bad by the fact of being boring.

6.0.1. Motivation

It is not uncommon to see young children playing video games from a very early age. Many times they do not yet know how to read or understand well what is happening on screen and yet they are able to complete levels or even the entire game. For example, how can a child who cannot read complete a game like Pokémon, where dialogues are an important part of the gameplay?

This work stems from this question. It focuses on training closer to what a human would have, i.e., using only the screen, and focuses on a type of game that is barely studied.

6.0.2. Objectives

The final objective of this work is to develop a reinforcement learning algorithm that takes as input only the image and uses the "Counting after hashing" technique. Therefore, a series of objectives leading to the final goal will be set and followed as a work plan.

1. Create a development environment with the necessary libraries, emulators and version control to be able to carry out the monitoring.

2. Create the convolutional neural network that will be used as a model for the agent and prepare the training script.

3. Establish the reward system.

4. Train progressively more elaborate agents on the main game.

5. Train agents on a series of secondary games.

6. Analyze the results of each of the agents and compare them with the expected behavior.

6.0.3. Structure

Once the main subject and its objectives have been explained in general terms, Chapter 2 explains the theory framework, starting from the fundamentals and advancing through the different breakthroughs that have led to the current state of the art.

Then, in chapter 3, we will see the development of the work itself, with its corresponding explanations of the methods used, decisions taken and their corresponding results.

Finally, in chapter 4, a brief conclusion will be drawn from the results obtained and the features that one can add to the project to improve it will be listed.

Chapter 7

Future Work and Conclusions

7.1. Future Work

As a last chapter, I will review the possible improvements that can be made to this work, taking into account the results obtained.

A pending issue, and perhaps the most obvious, is to apply the used method to other games, also trying to combine intrinsic rewards with conventional ones, and measure the results. As explained above, there are games where it is not applicable and it is not a question of whether they are better or worse, but of their necessity. It could be interesting to study to what extent they improve performance in other situations.

Another important improvement to the method would be to apply another type of hash or a similar method to better measure the novelty level of the states. Currently training does not reward changes of a few pixels, but in certain games the change of a few pixels can contain a lot of information. For this reason, auto-encoders or detection of novel patterns can contribute to improve the reward function.

Related to the previous point about the importance of some pixels, and taking into account the opacity of neural network systems, it can also be interesting to apply explainability techniques. An example of this, and probably the one that makes the most sense, is to generate attention or heat maps from the input image, in order to evaluate which sections of the image are contributing to take each action. This information is interesting because it can tell us what the agent is acting on, but also

what elements it is ignoring. An example of this would be in the case of Pokémon Red where it doesn't seem to know what to pay attention to in battles.

Finally, and also the most difficult improvement, would be to improve the capabilities of the agents by providing them with resources to remember past events, or at least improve their ability to relate inputs within the time series. This can be done by adding memory banks to the agents in exchange for efficiency or by adding recurrent structures to the neural networks in exchange for much slower training. In many video games, such as Super Mario Bros, this is not necessary and will not contribute anything, but in others, such as Pokémon, it would be an indispensable requirement since there is information that is not always on screen and that the player must remember.

7.2. Conclusions

This paper has evaluated the importance of intrinsic rewards in some games. Agents have been trained using Deep Learning and reinforcement learning to learn to play video games, and it has been observed that this approach works surprisingly well even without the need for more conventional rewards.

The trained agents have exceeded expectations. In the main game, Super Mario Land 2, he has completed the first level with ease and has had the surprising result of even taking optional paths and save points in the process. Overall, it could be said that the agents exhibit "more human" behavior although this is entirely subjective.

On the other hand, even when agents have encountered some obstacle that they have not been able to learn to overcome, closer observation of the agent yields reasonable explanations that only reinforce the idea that these types of rewards are positive and better delineate their advantages and disadvantages.

Bibliografía

CONTENT BLOCKCHAIN. Testing different image hash functions. 2019. URL: https://content-blockchain.org/research/testing-different-image-hash-functions/.

BROWNLEE, J. A gentle introduction to the rectified linear unit (relu). 2020. URL: https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks.

CLOUD, A. Google image search explained. 2017. URL: https://alibaba-cloud.medium.com/google-image-search-explained-30af8ba9cbea.

DAABOUL, K. Reinforcement learning: Dealing with sparse reward environments. 2020. URL: https://medium.com/@m.k.daaboul/dealing-with-sparse-reward-environments-38c0489c844d.

JANIESCH, C., ZSCHECH, P. y HEINRICH, K. Machine learning and deep learning. *Electronic Markets*, vol. 31(3), páginas 685–695, 2021. ISSN 1422-8890.

KRAWETZ, D. N. Looks like it. 2011. URL: https://www.hackerfactor.com/blog/index.php?/archives/432-Looks-Like-It.html/.

KUMAR, A. Real world applications of convolutional neural networks. 2021. URL: https://vitalflux.com/real-world-applications-of-convolutional-neural-networks/.

LASKIN, M. Reinforcement learningn with deep q networks. 2022. URL: https://www.anyscale.com/blog/reinforcement-learning-with-deep-q-networks.

Lawrence, S., Giles, C., Tsoi, A. C. y Back, A. Face recognition: a convolutional neural-network approach. *IEEE Transactions on Neural Networks*, vol. 8(1), páginas 98–113, 1997.

Mcculloch, W. y Pitts, W. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, vol. 5, páginas 127–147, 1943.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. y Riedmiller, M. A. Playing atari with deep reinforcement learning. *CoRR*, vol. abs/1312.5602, 2013.

Ocio Barriales, S. Inteligencia artificial en la industria del videojuego aaa. ¿puede el mundo académico contribuir a su éxito? 2014.

O'Shea, K. y Nash, R. An introduction to convolutional neural networks. *CoRR*, vol. abs/1511.08458, 2015.

van Otterlo, M. y Wiering, M. *Reinforcement Learning and Markov Decision Processes*, páginas 3–42. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-27645-3.

Oudeyer, P.-Y. y Kaplan, F. What is intrinsic motivation? a typology of computational approaches. *Frontiers in Neurorobotics*, vol. 1, 2007.

Oudeyer, P.-Y., Kaplan, F. y Hafner, V. V. Intrinsic motivation systems for autonomous mental development. *IEEE Transactions on Evolutionary Computation*, vol. 11(2), páginas 265–286, 2007.

Pathak, A. R., Pandey, M. y Rautaray, S. Application of deep learning for object detection. *Procedia Computer Science*, vol. 132, páginas 1706–1717, 2018. ISSN 1877-0509. International Conference on Computational Intelligence and Data Science.

Popescu, M.-C., Balas, V., Perescu-Popescu, L. y Mastorakis, N. Multilayer perceptron and neural networks. *WSEAS Transactions on Circuits and Systems*, vol. 8, 2009.

Rosenblatt, F. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, vol. 65(6), páginas 386–408, 1958. ISSN 0033-295X.

Schneider, N. A simple guide to reinforcement learning with the super mario bros. environment. 2021. URL: https://medium.com/geekculture/a-simple-guide-to-reinforcement-learning-with-the-super-mario-bros-environment-495a13974a54.

Schulman, J., Levine, S., Moritz, P., Jordan, M. I. y Abbeel, P. Trust region policy optimization. *CoRR*, vol. abs/1502.05477, 2015. URL: http://arxiv.org/abs/1502.05477.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A. y Klimov, O. Proximal policy optimization algorithms. 2017.

smithsonianmag. The original 'space invaders' is a meditation on 1970s america's deepest fears. 2018. URL: https://www.smithsonianmag.com/science-nature/original-space-invaders-icon-1970s-America-180969393/.

Sutton, R. S., Barto, A. G. et al. Reinforcement learning. *Journal of Cognitive Neuroscience*, vol. 11(1), páginas 126–134, 1999.

Tang, H., Houthooft, R., Foote, D., Stooke, A., Chen, X., Duan, Y., Schulman, J., Turck, F. D. y Abbeel, P. #exploration: A study of count-based exploration for deep reinforcement learning. *CoRR*, vol. abs/1611.04717, 2016.

towardsdatascience. First neural network for beginners explained (with code). 2019. URL: https://towardsdatascience.com/first-neural-network-for-beginners-explained-with-code-4cfd37e06eaf.

Watkins, C. J. C. H. y Dayan, P. Q-learning. *Machine Learning*, vol. 8(3), páginas 279–292, 1992. ISSN 1573-0565.

Weng, L. Exploration strategies in deep reinforcement learning. 2020. URL: https://lilianweng.github.io/posts/2020-06-07-exploration-drl/.

Wikipedia. Aprendizaje por refuerzo. 2023a. URL: https://es.wikipedia.org/wiki/Aprendizaje_por_refuerzo.

Wikipedia. Q-learning. 2023b. URL: https://es.wikipedia.org/wiki/Q-learning.

Wong, Q. Rl-deep qlearning. 2023. URL: https://gzwq.github.io/2019/03/10/RL-Deep-QLearning/.

Apéndice A

Apéndice

A.1. Enlaces

Enlaces a muestras del entrenamiento en Youtube:

1. Super Mario Land 2: `https://www.youtube.com/watch?v=_pNxVyJS4DY`
2. Pokémon Red: `https://www.youtube.com/watch?v=XFjtxeT_iLU`
3. Kirby's Dream Land: `https://www.youtube.com/watch?v=qnggguywQA8`
4. Super Mario Land 2 ejemplo de entrenamiento: `https://www.youtube.com/watch?v=7PF6XCtiJDY`