¿Cómo definir el aprendizaje reforzado si no es comparándolo con otros tipos de aprendizaje? Pues desde mi experiencia como madre y como especialista en machine learning diría que es lo más parecido a educar a un hijo, le das las herramientas para que pueda tomar sus propias decisiones y esperas que esas decisiones sean las más acertadas, y a grandes rasgos, eso es lo que he hecho en este proyecto.

Mi idea al preparar el proyecto era elegir un entorno de los disponibles en gym que tratara el problema del movimiento de un agente que se encuentra obstáculos inesperados a lo largo de su camino. El nivel de dificultad de los diferentes entornos hizo que eligiera un juego de Atari, Ms Pacman, ya que podríamos definirlos como entornos de dificultad media, aunque justamente este juego no es fácil de resolver precisamente. En este juego, el objetivo es que Ms Pacman se coma todos los puntitos que aparecen en la pantalla antes de que uno de los fantasmas acabe con ella. En este caso el agente tiene que tomar decisiones basadas en el análisis del entorno de forma visual, teniendo que extraer toda la información de los diferentes píxeles que conforman la imagen de cada estado *s* en cada momento *t*.

Para llevar a cabo la extracción de información de una combinación casi infinita de píxeles como es el caso de nuestro entorno, haremos uso del algoritmo de aprendizaje por refuerzo DQN (Deep Q-Network, por sus siglas en inglés) que se basa en la ecuación de Bellman y su sentido de la recursividad.

Con la ayuda de este algoritmo, obtendremos la mejor **predicción de recompensa futura** dada una acción a tomar. Esta es una aproximación que busca obtener siempre el valor *Q* mayor, siendo *Q* el valor total de tomar una acción *a* en un estado *s* (Q(s,a)) y se obtiene sumando las recompensas futuras *r* y ajustando el resultado con un valor de descuento *gamma*. Con este proceso nos aseguramos de que escogemos la acción que nos llevará a la mayor puntuación en cada paso.

Para poder utilizar los valores que obtendremos en el futuro, necesitamos hacer una predicción de ellos. Para llevar a cabo esta labor, duplicamos nuestra red neuronal de forma que podamos obtener un valor predicho y un valor "real" que nos permitan llevar a cabo el descenso por gradiente y así ir actualizando los pesos para que nuestro modelo aprenda.

La primera red (*q-model*) se encarga de predecir la acción más adecuada mientras que la copia (*t-model*) se encarga de generar los valores target. Realmente la única red que entrenamos es la primera, pero transferimos los pesos de esta a la copia de vez en cuando para optimizar sus parámetros. Como ya he explicado, el aprendizaje se lleva a cabo usando descenso por gradiente, entendiendo que se calcula la diferencia entre los valores de Q estimados (los resultados de *q-model*) y los valores de Q reales o target (los resultados de *t-model*).

La principal diferencia en este entrenamiento respecto del entrenamiento de un modelo supervisado es que aquí nuestro dataset lo crea la misma red que entrenamos, siendo este siempre variable y viéndose afectado por las acciones elegidas por la propia red. Sin embargo, el dataset de un modelo supervisado es preciso e invariable, ya que ha sido revisado por los humanos antes de ser utilizado en el entrenamiento y no se vuelve a modificar durante este.

Por otro lado, nuestro agente también aprende gracias a lo que llamamos "buffer", el cual acumula las experiencias que va creando nuestro modelo. Normalmente genera un episodio por cada iteración del bucle principal y lo almacena. Podemos considerar el buffer como un tipo de dataset que contiene las experiencias pasadas de nuestro agente. Cada una de estas experiencias está definida como una quíntupla *(s, a, r, n_s, d)*, en ella *s*, *a* y *r* mantienen sus definiciones anteriores, la *d* es un valor booleano que indica al agente si este es el último estado del episodio o no, y por último, *n_s* representa el estado siguiente después de K una vez el agente ha ejecutado la acción K. De esta forma el buffer se llena de experiencias a medida que Ms Pacman se mueve por la pantalla, a continuación, durante el entrenamiento nuestro modelo usa esas experiencias para aprender y así tomar mejores decisiones en acciones futuras. Este tipo de aprendizaje es una réplica de cómo los humanos aprendemos de experiencias pasadas.

Este buffer tiene un tamaño determinado, de forma que al alcanzar el máximo de su capacidad, nuestro modelo elimina las experiencias más antiguas y deja sitio para almacenar las nuevas manteniendo

siempre el mismo orden de llegada. Sería algo así como “first in, first out”. Una característica más de la variabilidad de nuestro dataset.

En cuanto a la recompensa media esperada teniendo en cuenta los scores reales del juego, estaría alrededor de 1660, ya que en la pantalla en la que se está jugando hay 150 puntitos (incluyendo las 4 pastillas que activan la magia de Ms Pacman la cual permite comer fantasmas). La puntuación mínima (sin comerse ningún fantasma ni ninguna fruta que dan puntuación extra) sería la siguiente:

146 puntos x 10 = 1460
4 pastillas x 50 = 200
TOTAL = **1660** (puntuación mínima para pasar de pantalla)

Por otro lado, definimos el tipo de estado que se observa en nuestro entorno como un estado continuo, ya que los cambios de estado se producen en cualquier instante y hacia cualquier estado dentro de un espacio continuo de estados.

EXPERIMENTOS

Con el objetivo de comparar los resultados del algoritmo DQN utilizando distintas técnicas, he llevado a cabo diversos experimentos usando tres versiones diferentes de este algoritmo: DQN Vanilla, Double DQN y Dueling DQN.

A continuación, se lista una serie de variables entre las cuales situaremos los posibles parámetros a modificar para afinar nuestro modelo, especialmente cuando la convergencia se complica, como ha sido mi caso. En la siguiente tabla se detallan algunos de estos parámetros y los diferentes ajustes que he probado.

Parámetro	**Ajustes Realizados**
Número máximo de episodios (max_episodes) Este entorno necesita millones de episodios para poder converger, cosa que hace realmente imposible alcanzar ese objetivo con una máquina corriente, haciendo necesario el uso de equipamientos o recursos cloud más potentes que incluyan una o varias GPUs.	100 – 500 – 1000 – 2000 – 10k
Recompensa que alcanzar antes de parar el entrenamiento (stopping_reward_criteria) He ajustado este parámetro teniendo en cuenta que la puntuación mínima para pasar de pantalla es 1660 y la máxima son 13760 puntos. Mi razonamiento ha sido que darle la puntuación mínima no era suficiente ya que podría conseguir esa puntuación comiendo fantasmas, por ejemplo, sin haber terminado de comerse todos los puntos. Por lo tanto, he decidido darle margen para que pudiera comer un par de fantasmas por pastilla, cosa que ya es complicada, y que se pudiera comer la cereza además de los puntitos. Esto sería un total de 4160 puntos y lo he redondeado a 5000. * Durante mis experimentos, intenté modificar el cálculo de la recompensa para que se penalizara el inmovilismo del agente, sin embargo, no lo conseguí. Mi idea era comparar el estado actual state_n con el estado siguiente next_state_n y penalizar al agente si ambos estados eran iguales, pero no funcionó porque el estado también lo componen los fantasmas y estos nunca están quietos, por lo tanto, ambos estados prácticamente nunca son iguales.	80 - 100 – 1120 – 1500 – 5k
Número de pasos que conforman nuestra experiencia (n_steps) La idea al aumentar el número de pasos es darle mayor margen de maniobra al agente. Sin embargo, cuando se aumenta mucho lo que se consigue es que aumente también el tiempo que dura cada episodio y no se observan mejoras en las métricas.	2, 5, 1k, 10, 100
Tamaño del batch (batch_size) No es un parámetro que influya demasiado en el comportamiento de este modelo.	32, 16, 64
Valor de gamma (gamma) Se refiere al valor de descuento de las recompensas futuras. El incremento aplicado parece haber tenido un efecto positivo en el entrenamiento.	0.95, 0.99

Valor de épsilon máximo y mínimo (epsilon y epsilon_min) Después de modificarlo un par de veces y ver que no hacía sino darle menos oportunidades a mi agente de explorar, volví a su configuración original max. 1.0 y min. 0.01.	Max. 1, 0.85 Min. 0.01
Número aproximado de iteraciones para afinar el epsilon decay (approx_iterations) Es el parámetro que más ha afectado al rendimiento de mi modelo. En su valor máximo el modelo aprendía muy despacio. Si lo ajustaba muy bajo, el modelo apenas exploraba nada e iba directo a explotar lo ya conocido. El punto de inflexión está entre 2e5 y 2e6, al menos para el número de iteraciones que mi máquina logra alcanzar. Es posible que, con mayor número de iteraciones, un valor mayor fuera más adecuado.	2e7, 2e6, 2e5, 2k, 200, 150k
Porcentaje de decrecimiento de épsilon (epsilon_decay) Intenté ajustar este parámetro sumando un número real (1.0, 0.5) al divisor del segundo paréntesis, pero la épsilon caía a 0.1 nada más empezar el entrenamiento y por eso restauré los valores originales.	
Tamaño del buffer de experiencias (mem_length) Mi intención al reducirlo era aumentar la probabilidad de que el agente se viera obligado a repetir las mismas acciones con mayor frecuencia. Pero realmente se consigue el efecto contrario, ya que le obligo a olvidar experiencias que le enseñarían si las volviera a ver.	20k, 400, 600, 1k
Tasa de aprendizaje (learning_rate) La he aumentado para conseguir un aprendizaje más rápido, aunque he intentado mantenerla baja para no caer en un mínimo local.	0.01, 0.15, 0.2, 0.25
Activación del aprendizaje conjunto de las redes DQN (double_dqn_learning) Lo he dejado activado (True) en la mayoría de los experimentos. Solo lo he desactivado al entrenar las otras variantes (DQN Vanilla y Dueling DQN). En los experimentos llevados a cabo con la técnica Double DQN, el objetivo era evitar una sobreestimación de las Qs provocada por la ecuación de Bellman, la cual suele elegir la Q mayor independientemente de la acción. En contraposición, el double DQN learning propone escoger el valor Q de la acción elegida en lugar del mayor valor Q de todas las acciones posibles.	
Activación variante (dueling_dqn) Solo he activado este parámetro para llevar a cabo el experimento con la variante Dueling DQN, el resto del tiempo ha estado desactivado. Esta variante nos ofrece una combinación del valor de la recompensa en el estado *s* y la ventaja de escoger una acción sobre las demás. Esta técnica es especialmente útil en situaciones en las que no conocemos el valor exacto de cada acción, ya que en esas situaciones conocer la función estado-valor sería suficiente.	
Número de capas ocultas A mayor número de capas, los resultados mejoraban.	64, 32, 128

Además de los parámetros mencionados, también he cambiado varias veces la estructura de mis redes. En las capas convolucionales ajustando el número de filtros de 16, 32, 32 a 32, 64, 64 , además, he añadido una capa sin activación antes de cada capa activada con Relu, basándome en el modelo *pacman_full_DQN* de Prasoon Shukla (https://github.com/prasoon2211/pacman-RL).

En cuanto a la recompensa promedio esperada en cada episodio, he obtenido las siguientes recompensas medias tras 100 episodios.

DQN Vainilla: 210

Double DQN: 60

Dueling DQN: 210

Para poder comparar las tres técnicas (Double DQN, DQN Vanilla y Dueling DQN), he usado los mismos ajustes en el entrenamiento. Al analizar sus métricas y resultados, observamos que:

- El número máximo de pasos alcanzado se sitúa entre 370k y 380k steps. La variante que ha obtenido un mayor número de steps ha sido la DDQN, mientras que la que menos ha obtenido ha sido la Dueling DQN. Esta última ha sido la que más tiempo ha necesitado para alcanzar los

500 episodios ajustados, alrededor de 12 horas, mientras que las otras dos consumían unas 4-5 horas para hacer lo mismo.

- Por otro lado, la variante Vanilla y la DDQN son las que han obtenido un mayor valor en sus recompensas por episodio, obteniendo una recompensa máxima de aproximadamente 550 por episodio.
- En cuanto al loss, la red que ha alcanzado un valor más bajo de loss ha sido la variante Vanilla, aunque todas ellas tienen grandes picos y valles dando señales de un mayor aprendizaje en determinados momentos del entrenamiento.

Teniendo en cuenta los datos obtenidos con los experimentos realizados, lo más eficiente tanto a nivel computacional como de rendimiento del modelo sería utilizar la variante Vanilla, ya que requiere un menor número de cálculos, el tiempo requerido para ejecutar el modelo es menor y el resultado de la recompensa media es uno de los máximos.

Una peculiaridad de este entorno es que requiere tal capacidad de cómputo que cada iteración tarda muchísimo en realizarse, desde una media de unos 15 segundos por iteración hasta unos 2 minutos por iteración, lo cual hace imposible la tarea de llegar a converger si dependes de herramientas como Google Colab. Según los papers consultados, para conseguir resultados óptimos, es necesario entrenar durante alrededor de 4 millones de pasos, sin embargo, en mis experimentos no he pasado de 380k pasos. Por ese motivo, ninguno de mis modelos ha llegado a converger.

Esta práctica ha sido todo un reto, motivadora y frustrante a la vez, especialmente después de consumir una gran cantidad de tiempo y no obtener los resultados esperados. Sin embargo, me ha ayudado a comprender mucho mejor el aprendizaje reforzado, sus herramientas y sobre todo sus posibilidades.

**** En el anexo 1 se pueden consultar los diferentes gráficos obtenidos durante el entrenamiento de las tres variantes de DQN.***