Implementación de un Agente para

la Robótica Espacial

Resolución el entorno Luna-Lander de la librería Gym de OpenAi, a través de técnicas de aprendizaje por refuerzo profundo

Jorge Álvarez Gracia

Universitat Oberta de Catalunya. Máster Oficial de Ciencia de Datos. Asignatura de Aprendizaje por Refuerzo.

Introducción

Motivación

Este trabajo forma parte de la práctica de la asignatura ***Aprendizaje por Refuerzo*** del Máster Oficial de Ciencia de Datos de la Universidad Oberta de Cataluña, del que he formado parte durante el año 2022 y principios del año 2023 como estudiante y en concreto de esta asignatura optativa.

Agradecimientos

Agradecer a los profesores Pablo Dini y Jordi Casas Roma, que han planteado esta práctica siendo necesaria y fundamental para consolidar todos los conceptos teóricos aprendidos durante el curso y que han apoyado a mí y a mis compañeros durante este periodo académico.

Objetivos

El objetivo de este proyecto es afianzar todos los conocimientos adquiridos durante el curso tomando como referencia el entorno Luna-Lander de la librería *Gym* de *OpenAI*, para ello, deberemos utilizar técnicas de aprendizaje por refuerzo profundo y alcanzar el umbral establecido de recompensas, 200 puntos de media durante al menos 100 partidas.

Para ello, presentaremos el entorno de referencia, implantaremos un agente Deep Q-Network (DQN) que lo solucione y posteriormente crearemos otro agente que pueda mejorar el rendimiento del agente DQN anteriormente implementado.

Estructura del proyecto

Vamos a dividir este trabajo en cuatro partes diferenciadas, en la primera describiremos el entorno elegido y su funcionamiento. Veremos cinco puntos importantes de estudio en cualquier problema de aprendizaje por refuerzo como son: el estado inicial, la descripción del entorno, el espacio de acción, el espacio de observación, los criterios de las recompensas y los hechos que llevarán a la finalización de cada episodio. Además, en este apartado crearemos un agente 'torpe' que realizará acciones aleatorias sirviéndonos como utilidad para la comprensión del entorno y como referencia para los distintos agentes.

En el segundo apartado, describiremos y utilizaremos el algoritmo DQN para resolver el entorno, realizando una optimización de los hiper-parámetros que nos permitan resolver el problema de la manera más eficiente posible.

En la tercera parte de este trabajo, intentaremos mejorar los resultados obtenidos en el apartado anterior con distintas variaciones y mejoras del algoritmo DQN así como la búsqueda de sus mejores hiper-parámetros, definiremos y explicaremos estos nuevos algoritmos y expondremos los criterios que hemos seguido para elegir entre uno u otro.

Para finalizar, explicaremos las diferencias de cada uno de los agentes obtenidos y expondremos las conclusiones obtenidas del proyecto.

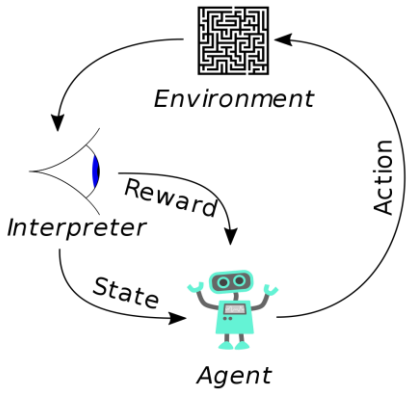
Aprendizaje por Refuerzo y Aprendizaje por refuerzo Profundo

Dentro del aprendizaje automático, machine learning, encontramos tres ramas diferenciadas.

Aprendizaje supervisado, podemos crear modelos capaces de aprender a través de unos datos de entrada y sus correspondientes etiquetas, las cuales nos dan información en la obtención de los outputs, aprendizaje no supervisado, donde no disponemos estos datos etiquetados y el aprendizaje por refuerzo. Hay que destacar las técnicas de aprendizaje profundo, redes neuronales, fuertemente utilizadas en el aprendizaje por refuerzo.

¿En qué consiste pues el aprendizaje por refuerzo? ¿Y el aprendizaje por refuerzo profundo?

Podemos definir al aprendizaje por refuerzo como la ciencia de tomar decisiones a partir de la iteración de un entorno y mediante la obtención de ciertas recompensas que dependerán de las distintas decisiones llevadas a cabo realizadas para cumplir ciertos objetivos.



El aprendizaje por refuerzo profundo pues, combinas técnicas de Deep Learning como son las redes neuronales, por simplificar y en rasgos generales, las observaciones del entorno serán los inputs de la red neuronal y las acciones a llevar a cabo los outputs.

Tecnologías Utilizadas

Hemos utilizado el lenguaje de programación **Python**, además de distintas librerías comunes en la ciencia de datos de este lenguaje, como son Numpy, Pandas, etc.

Por otro lado, como hemos señalado anteriormente, nos ayudamos de la **librería Gym** de la empresa openIA para el uso del entorno de esta actividad. Para la creación de las redes neuronales y de los agentes usaremos el framework **PyTorch**.

El Entorno Luna Lander

Correspondiente al ejercicio 1

Descripción del Entorno:

Este entorno es un problema clásico de optimización de la trayectoria de un cohete. Encontramos dos versiones del entorno: discreto o continuo. En nuestro caso, trabajaremos con el entorno de manera discreta.

Algunas de sus principales características son:

- La plataforma de aterrizaje siempre está en las coordenadas (0,0).
- Las coordenadas son los dos primeros números en el vector de estado.
- Es posible aterrizar fuera de la plataforma de aterrizaje.
- El combustible es infinito, por lo que un agente puede aprender a volar y luego aterrizar en su primer intento.

Estado inicial

El módulo de aterrizaje comienza en el centro superior de la ventana gráfica con una fuerza inicial aleatoria aplicada a su centro de masa.

Espacio de acción

Hay cuatro acciones discretas disponibles: no hacer nada, disparar el motor de orientación izquierda, disparar el motor principal, disparar el motor de orientación derecha.

- No hacer nada.
- Orientar el fuego del motor a la izquierda.
- Disparar el motor principal.
- Orientar el fuego del motor a la derecha.

Espacio de observación

El estado o espacio de observación es un vector de ocho dimensiones, donde cada valor contiene información diferente sobre el módulo de aterrizaje:

- Coordenada de plataforma horizontal (x).
- Coordenada de plataforma vertical (y)

- Velocidad horizontal (x).
- Velocidad vertical (y).
- Ángulo.
- Velocidad angular.
- Si la pierna izquierda tiene un punto de contacto tocó la tierra.
- Si la pierna derecha tiene un punto de contacto tocó la tierra.

Recompensas

- Moverse desde la parte superior de la pantalla hasta la plataforma de aterrizaje y detenerse a velocidad de cero es de aproximadamente 100 a 140 puntos.
- Cada disparo del motor principal por paso dentro del episodio es -0.3 puntos.
- Cada contacto con el suelo de los soportes de la nave es +10 puntos.
- Si al terminar el episodio el módulo de aterrizaje se estrella (- 100 puntos) y si se detiene (+100 puntos)

Finalización del episodio

El episodio termina si:

1. El módulo de aterrizaje se estrella (el cuerpo del módulo de aterrizaje entra en contacto con la luna).
2. El módulo de aterrizaje sale de la ventana gráfica (la coordenada x es mayor que 1.
3. El módulo de aterrizaje no está en funcionamiento. No estar en funcionamiento significa que el cuerpo no se mueve y no choca con ningún otro cuerpo.

Comportamiento del Agente Torpe, elección de la acción de forma aleatoria

Observamos que en la mayoría de los episodios los pasos realizados en cada uno de ellos son inferior a 200, esto ocurre porque la nave al aterrizar se estrella ya que al realizar acciones aleatorias no aprenderá en el transcurso del tiempo. De la misma manera, las recompensas del agente aleatorio serán negativas, la mayor parte entre -50 y -250 puntos.

Gráfico 1: Recompensas totales y distribución de pasos por episodio de un agente aleatorio

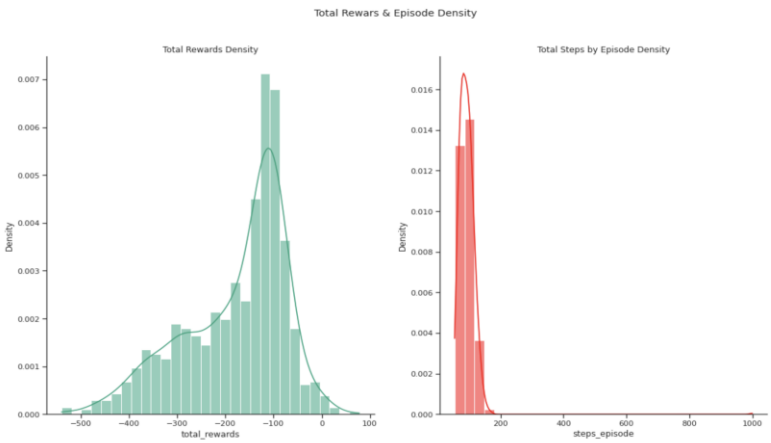
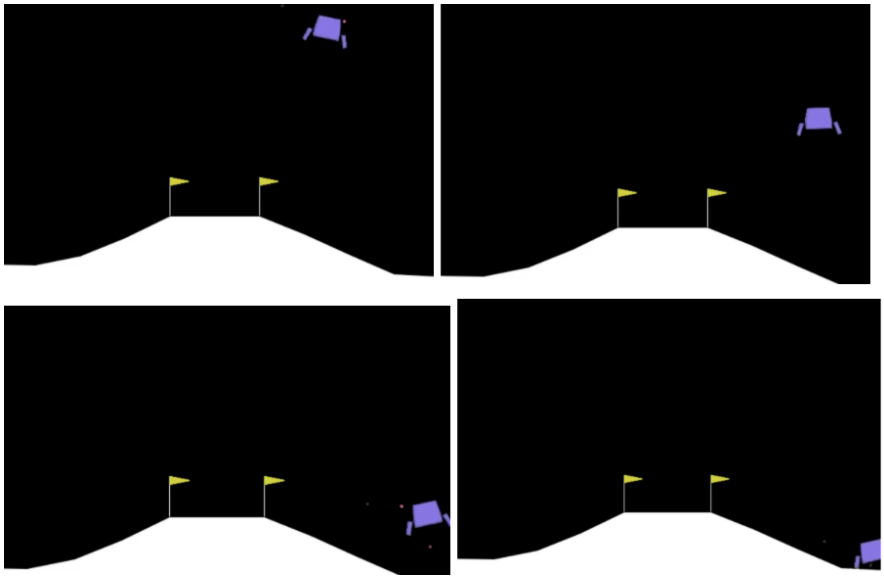


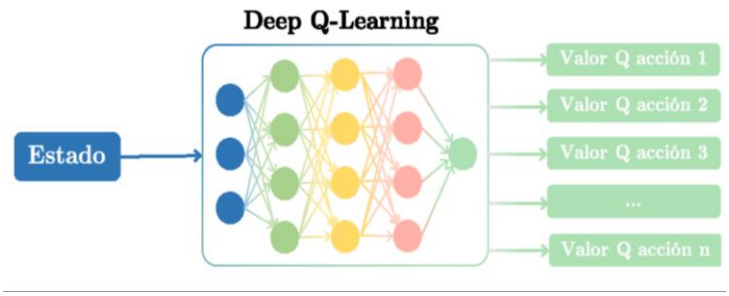
Figura 1: Fotogramas del comportamiento del agente aleatorio:



Si observamos los fotogramas de la figura dos, vemos que el agente torpe lleva a que la nave choque con el suelo. * *Los videos se puede encontrar en el repositorio Github de la actividad.*

Algoritmo y Agente DQN.

(Correspondiente al ejercicio 2)



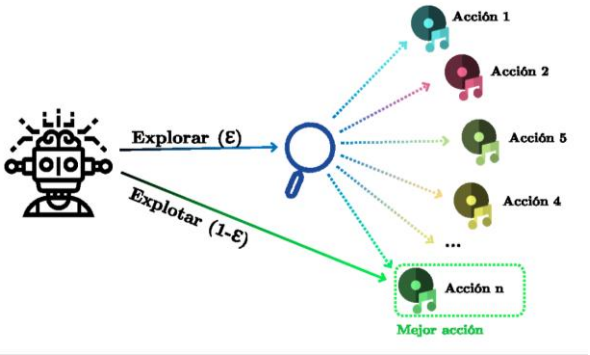
El algoritmo DQN, Deep Q-Network, fue propuesto por el equipo de la empresa Deepmind en 2013, siendo el primero que combinaría redes neuronales convolucionales, deep learning, con el aprendizaje por refuerzo.

Esto llevaría a un futuro prometedor de esta área siendo capaz de tratar problemas más complejos.
A pesar de ello, esa nueva técnica poseía ciertas limitaciones que se debían solucionar y que expondremos brevemente ya que hemos debido integrarlas en este proyecto:

Métodos o técnicas de mejora del algoritmo DQN:

1) Método ε-greedy

Nuestro agente deber ser capaz de explorar el entorno para poder conocer los distintos estados vistos y el resultado de las distintas acciones llevadas a cabo. En un principio, el agente tomará acciones prácticamente aleatorias, pero conforme los episodios transcurran y vaya aprendiendo irá adquiriendo cierta certeza de que las acciones que lleva a cabo tienen sentido, conociendo su resultado al haberlas realizado anteriormente.



El método e-greedy nos permitirá establecer dicho comportamiento del agente, incentivando a una mayor exploración al principio y una reducción de está a favor de una explotación más elevada según avanzan los episodios realizados. Introducimos un parámetro de probabilidad ε, el que indica cuándo pasar de una política aleatoria a una política Q. Cuando el valor de épsilon es uno, todas las acciones que se toman son aleatorias. Conforme se vaya reduciendo esta probabilidad, el agente pasará a tomar más acciones acordes con la política Q, llegando en nuestro caso a un mínimo de 0.01.

2) Experience replay Buffer

A la hora de usar redes neuronales y por tanto el descenso del gradiente, necesitaremos que los datos sean independientes y estén idénticamente distribuidos.

Nuestros datos no son independientes ya que al estar introduciendo los de forma secuencian estará fuertemente relacionados entre ellos. Por otro lado, tampoco tendrán una distribución idéntica.

Para solucionar estos inconvenientes llevaremos a cabo la técnica conocida como *experience replay buffer*, almacenaremos cierta cantidad de experiencias mientras este sigue experimentado y de esta manera, el agente consiga aprender de las experiencias recientes.

Además, seleccionaremos de forma aleatoria un subconjunto de estos, utilizándolos por la red neuronal con el objetivo de reducir la correlación entre ellos.

3) Red objetivo

La correlación entre estados nos llevará a un problema añadido, los vectores (s, a, r, s′) de un estado y el siguiente serán muy similares, casi indistinguibles para la red neuronal. Esto podría llevar a problemas de sobreajuste, tendiendo a generalizar las situaciones más comunes u observadas. Para solucionar este problema podemos introducir una segunda red neuronal para mejorar el aprendizaje. En lugar de obtener el valor objetivo Q(s′,a′) y el valor predicho Q(s,a) con la misma red neuronal, calcularemos el valor objetivo con esta segunda red neuronal a la que llamaremos red objetivo. Esta segunda red será una copia de la principal, pero con los pesos fijos. Con este mecanismo conseguiremos estabilizar el entrenamiento y generalizar mejor los patrones en los datos.

Algoritmo DQN final

Esta última mejora fue introducida por *Deepmind* durante el año 2015, observando una mejoría respecto al algoritmo anterior. Las DQN usan, por lo general, redes neuronales convolucionales, especialmente cuando los estados son imágenes.

En nuestro caso, usaremos una red neuronal simple, ya que nuestros estados están representados por vectores numéricos y no por imágenes.

Agente DQN Inicial. Resultados iniciales

AL tratarse de un entorno que podemos considerar sencillo, hemos implementado el algoritmo DQN descrito anteriormente de la manera más sencilla posible, usando una red neuronal simple con tres capas lineales.

```
self.layers = nn.Sequential(
    nn.Linear(in_dim, units),
    nn.ReLU(),
    nn.Linear(units, units),
    nn.ReLU(),
    nn.Linear(units, out_dim)
```

Usando los siguientes parámetros del agente como punto de partida:

MEMORY_SIZE = 100000	LR = 5e-4
BATCH_SIZE = 64	MAX_EPSILON = 1.0
TARGET_UPDATE = 4	MIN_EPSILON = 0.01
EPSILON_DECAY = 0.995	THRESHOLD = 200
NN_UNITS = 64	SEED = 123
GAMMA = 0.99	
TAU = 1e-3	

En un primer lugar, con los parámetros anteriores, con 64 neuronas o unidades, hemos entrenado el agente, consiguiendo resolver el desafío en 1823 episodios. Tras ello primer estudio realizado ha conseguido incrementar estas unidades a 128, comprobando que el modelo llevaría a una mejoría de casi 1000 episodios. Resuelto en 950 episodios tomaremos este agente como referencia para el siguiente apartado.

Gráfico 2: Diferencias en el entramiento del Agente para diferentes unidades de la red neuronal

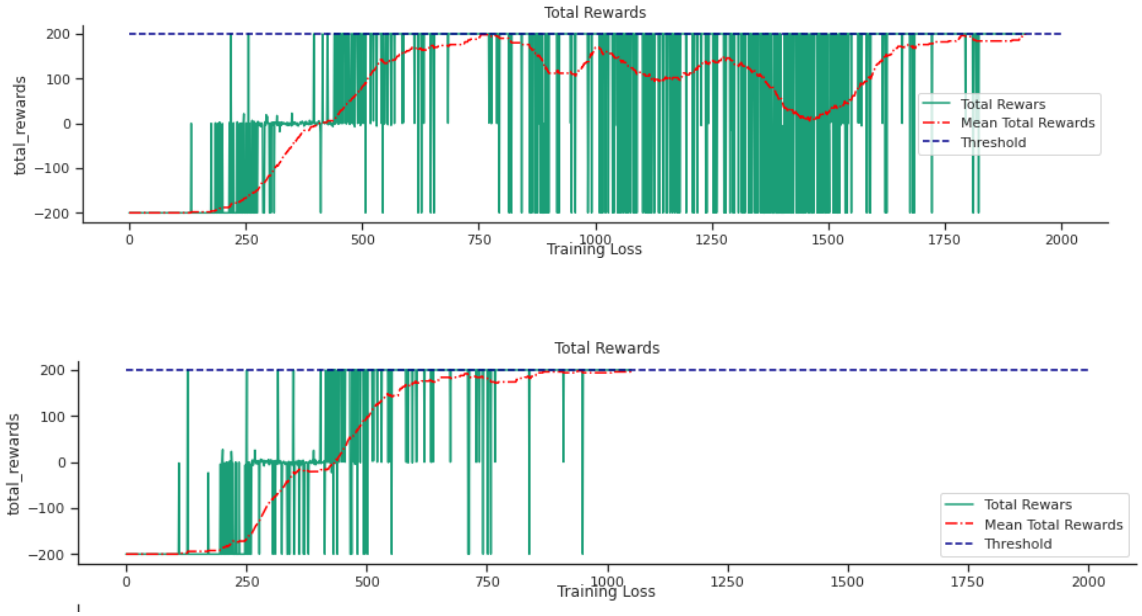


Gráfico de ambos resultados, en primer lugar, con 64 unidades y el segundo de 128.

Agente DQN Inicial. Optimización de los hiper-parámetros.

Tras este experimento, hemos llevado a cabo gran cantidad de variaciones en ciertos hiper-parámetros, entre ellos: la reducción de épsilon, el épsilon mínima, el factor de descuento, etc. Una vez conocidos aquellos para los que más influencia observábamos en el resultado final, hemos llevado a cabo un estudio más específico de estas variables, en concreto para el *batch size y learning rate.*

BATCH_SIZE = [32, 64, 128]
GAMMA = [0.99_LR = [5e-4,3e-4,2e-4,1e-4]

Los resultados obtenidos los hemos almacenado en un archivo *.csv* y los mejores modelos los hemos guardado en la carpeta modelos junto al código del proyecto.

Mostramos los distintos resultados en ordenados por de menor a mayor según el episodio en el que hemos resuelto el aterrizaje. Cabe destacar que hemos utilizado el primer episodio teniendo en cuenta 100 episodios siguientes con una media de 200 puntos.

Tabla 1. Búsqueda de hiper-parámetos agente DQN

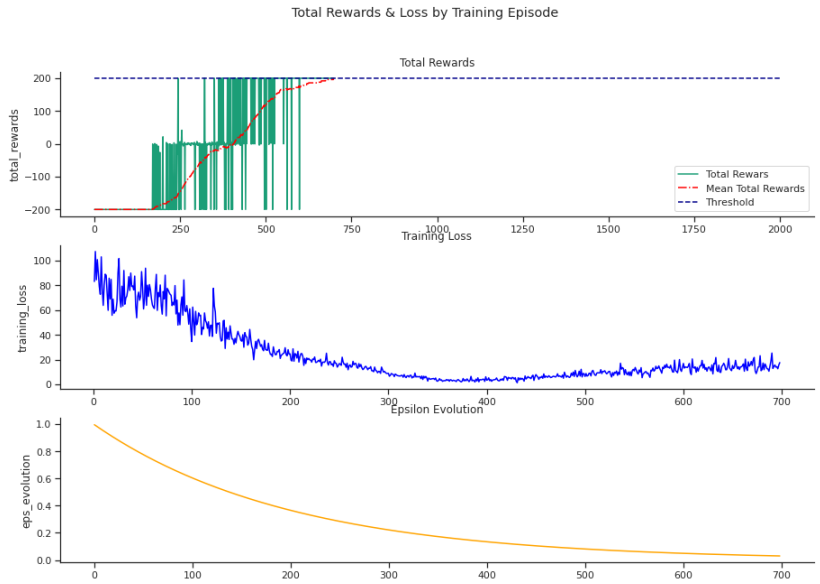
	agent_name	batch_size	lr	gamma	reward_mean	reward_std	training_time	solved_episode
8	9_DQN_Agent	128	0.0005	0.99	4.533190	172.469400	17.20	599
0	1_DQN_Agent	32	0.0005	0.99	53.335016	173.351734	16.63	834
10	11_DQN_Agent	128	0.0002	0.99	64.920167	170.826992	26.76	1135
4	5_DQN_Agent	64	0.0005	0.99	80.215897	162.246338	25.60	1302
5	6_DQN_Agent	64	0.0003	0.99	78.654492	164.925037	28.53	1478
11	12_DQN_Agent	128	0.0001	0.99	81.150435	163.085680	40.25	1501
2	3_DQN_Agent	32	0.0002	0.99	56.797561	169.410304	33.69	1534
9	10_DQN_Agent	128	0.0003	0.99	107.982696	153.269677	31.67	1813
1	2_DQN_Agent	32	0.0003	0.99	104.088434	156.038595	29.64	99999
3	4_DQN_Agent	32	0.0001	0.99	-22.406172	117.928658	69.08	99999
6	7_DQN_Agent	64	0.0002	0.99	123.463664	147.794207	28.36	99999
7	8_DQN_Agent	64	0.0001	0.99	93.157390	162.800422	34.82	99999

Agente DQN. Mejor Modelo

El agente número nueve es el más eficiente, con un **batch size** de **128**, **learning rate** de **0.0005** y **factor de descuento de 0.99**. El tiempo de entrenamiento ha sido de 17 minutos y 20 segundos. Podríamos estar satisfechos con estos resultados, pero vamos a intentar mejorarlos durante este proyecto por ello tomaremos como agente de referencia y de objetivo de mejora para el siguiente.

Adjuntamos los gráficos de esta actuación óptima:

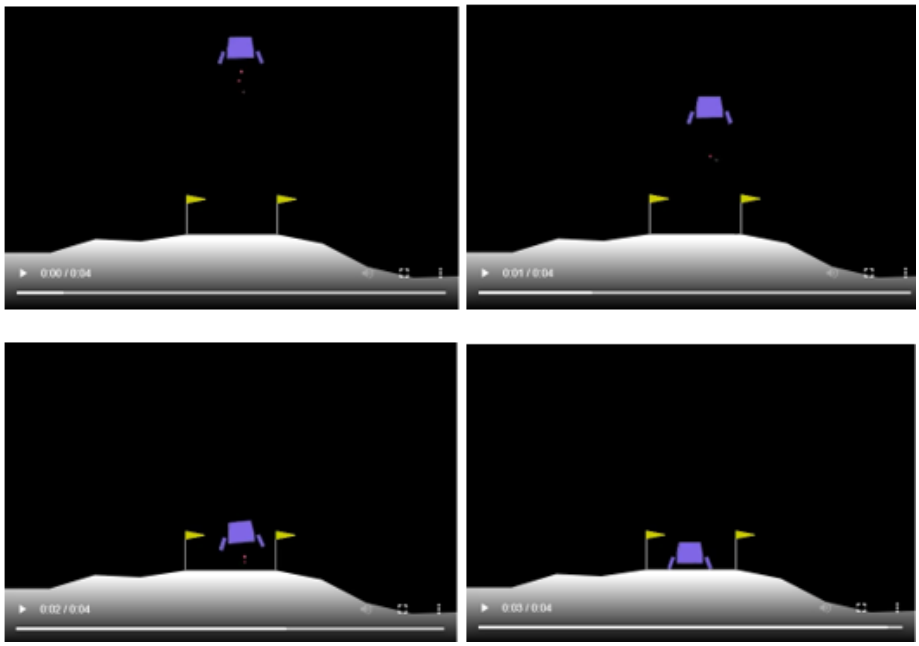
Gráfico 3: Evolución del entrenamiento del mejor agente DQN, recompensas por episodio, perdida y evolución de épsilon.



Agente DQN. Inferencia y Resultados

Con este agente hemos realizado la inferencia únicamente para un episodio y hemos obtenido una recompensa total de 269 puntos, y el siguiente comportamiento del agente obtenido de los fotogramas del video.

Figura 2: Fotograma del mejor agente DQN



El agente parece haber aprendido de forma adecuada el objetivo, desciende verticalmente de manera rápida y se queda fijado en las coordenadas (0,0) en el centro de la pantalla entre las banderas. Aunque entre el segundo uno y dos se mueve ligeramente a la derecha utilizando recursos extra de combustible.

Mejoras del agente DQN

(Correspondiente al ejercicio 3)

Consideraciones iniciales

Tras realizar diversas pruebas que no añadiremos en este proyecto observamos que los modelos de actor crítico no funcionan de manera tan eficiente como el modelo DQN inicial, observando un proceso de aprendizaje más largo y costoso.

Dentro de los modelos DQN, hemos utilizado el sistema de prueba y error utilizando los parámetros a cabo las distintas variaciones y mejoras propuestas por Deepmind. Entre ellas, PER, n-step, y Rainbow DQN.

Por alguna razón, estás técnicas más avanzadas no han dado los resultados esperados ralentizando considerablemente el proceso de entrenamiento sin completar el aterrizaje en los episodios necesarios. Por ello, y con el afán de no extender sin sentido este proyecto he optado por no añadir los resultados ni el código utilizado.

Debemos tener en cuenta que en este entorno sencillo las soluciones más complejas no son necesarias llegando a resultar ineficientes.

Los mejores resultados obtenidos los hemos encontrado combinando las técnicas de Dueling DQN y Noisy Network que explicaremos a continuación.

Variaciones utilizadas del algoritmo DQN

Dueling DQN

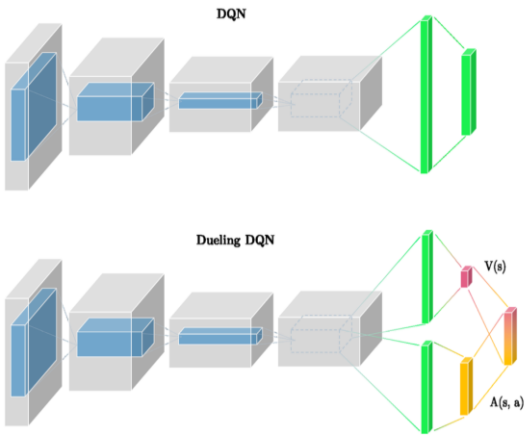
Esta mejora nos permite ganar eficiencia obviando aquellos estados donde las acciones que hayamos tomado no sean relevantes en el resultado final.

Para ello, debemos descomponer la función Q en dos componentes:

- V (s): el valor de estar en el estado s, equivalente a la recompensa esperada con descuento que es posible conseguir en el estado s
- A(s, a): ventaja de tomar la acción a en el estado s, cuánto mejor es esa acción a (qué cantidad extra de recompensa nos da) respecto de las otras posibles acciones.

Este tipo de arquitectura que permite obtener separadamente la estimación de la función de valor de estado y la función de ventaja es la que se conoce como Dueling Q-Network y fue propuesta de nuevo por DeepMind en el año 2016.

Figura 3. Arquitectura de la red Dueling DQN

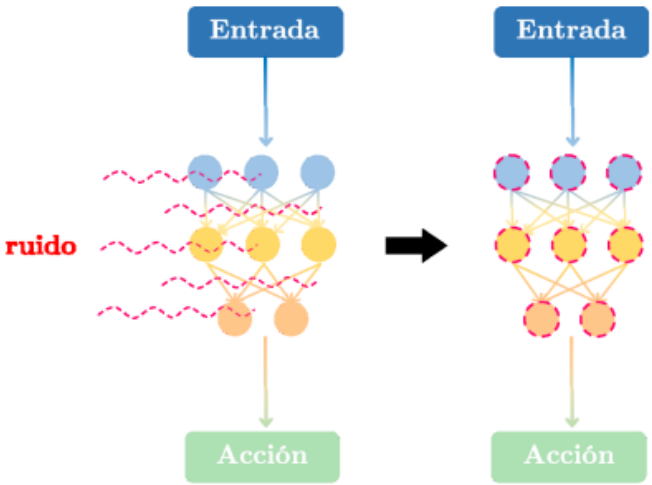


Noisy Red

Esta variación fue propuesta por el equipo de *DeepMind* en el año 2017 y trata de añadir ruido a los pesos y sesgos de las capas completamente conectadas de la red neuronal. Se demostró que un cierto componente estocástico en la política del agente llevará consigo un proceso de exploración más eficiente.

Al encontrarnos en un entorno sencillo, nuestro método ϵ-greedy ha llevado a buenos resultados por lo que en nuestro caso hemos combinado ambas técnicas mejorando los resultados anteriores.

Figura 4. Arquitectura de la red Noisy



Agente Dueling + Noisy DQN. Resultados iniciales

Mostramos la arquitectura neuronal de este nuevo agente, construyendo una arquitectura dueling dqn con una capa Noisy

```
self.fc1 = nn.Linear(in_dim, units)
self.fc2 = nn.Linear(units, units)
# calculate V(s)
self.fc3_1 = NoisyLinear(units, units)
self.fc4_1 = NoisyLinear(units, 1)
# calculate A(s,a)
self.fc3_2 = NoisyLinear(units, units)
self.fc4_2 = NoisyLinear(units, out_dim)
```

En este caso, directamente hemos entrenado el modelo con 128 neuronas por capa y los parámetros iniciales usados en el modelo DQN. Parámetros:

MEMORY_SIZE = 100000	LR = 5e-4
BATCH_SIZE = 64	MAX_EPSILON = 1.0
TARGET_UPDATE = 4	MIN_EPSILON = 0.01
EPSILON_DECAY = 0.995	THRESHOLD = 200
NN_UNITS = 64	SEED = 123
GAMMA = 0.99	
TAU = 1e-3	

Resultados:

Media de las recompensas por episodio de 192.

Gráfico 4: Entrenamiento primer agente Dueling + Noisy DQN

No hemos conseguido en 2000 episodios llegar a la recompensa media estipulada de 200 y por tanto, tampoco superar al primer modelo. Probaremos con la búsqueda de hiper-parámetros.

Agente Dueling + Noisy DQN. Optimización de los hiper-parámetros.

En lugar de probar con los mismos hiper-parámetros del modelo DQN, hemos decido cambiar las ratios de aprendizaje estudiados ya que hemos comprobado que los muy bajos no era eficientes. Además, solo entrenaremos los agentes durante 1000 episodios ya que si no se cumplen los objetivos en estos periodos el agente generado no nos interesará.

BATCH_SIZE = [32, 64, 128]
GAMMA = [0.99]
LR = [5e-4,4e-4,3e-4]

Resultados:

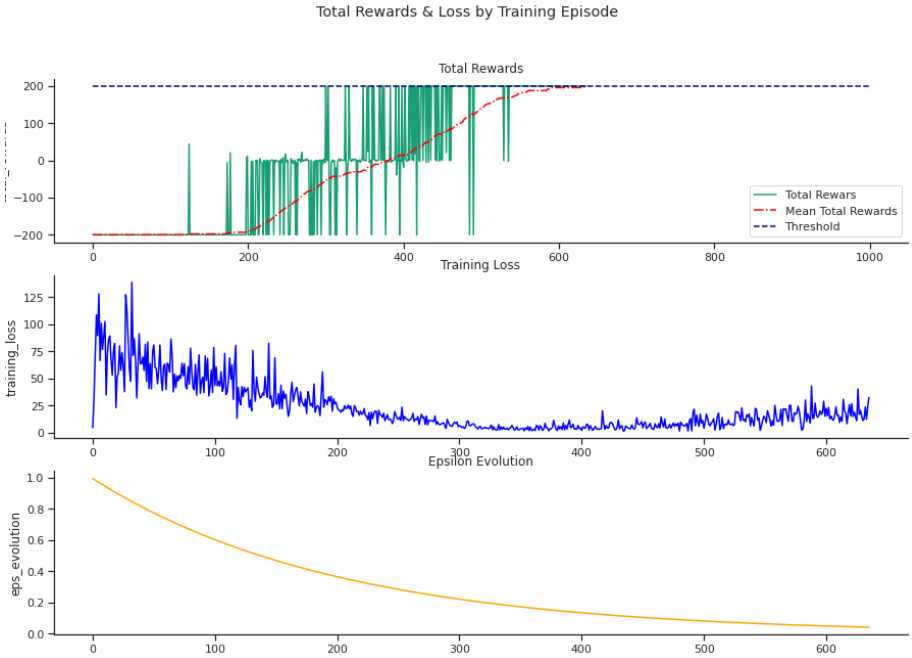
Tabla 2. Búsqueda de hiper-parámetos agente Dueling + Noisy DQN

	agent_name	batch_size	lr	gamma	reward_mean	reward_std	training_time	solved_episode
2	3_DDQN_Agent	32	0.0003	0.99	-4.080469	169.902372	20.78	536
0	1_DDQN_Agent	32	0.0005	0.99	66.803120	170.725828	26.16	99999
1	2_DDQN_Agent	32	0.0004	0.99	79.023623	168.175701	25.74	99999
3	4_DDQN_Agent	64	0.0005	0.99	57.167097	174.175008	26.96	99999
4	5_DDQN_Agent	64	0.0004	0.99	64.195556	170.554555	26.72	99999
5	6_DDQN_Agent	64	0.0003	0.99	59.933260	172.550969	28.35	99999
6	7_DDQN_Agent	128	0.0005	0.99	65.992992	176.330286	26.94	99999
7	8_DDQN_Agent	128	0.0004	0.99	67.477590	170.694480	28.97	99999
8	9_DDQN_Agent	128	0.0003	0.99	74.441499	165.610200	31.09	99999

Agente Dueling + Noisy DQN. Mejor Modelo

El mejor agente es el número 3, se ha resuelto en **536 episodios** con **32 de tamaño de batch** y **learning rate de 0.0003**. Con un tiempo de entrenamiento de aproximadamente **21 minutos**

Gráfico 5: Evolución del entrenamiento del mejor agente Dueling + Noisy DQN, recompensas por episodio, perdida y evolución de épsilon.

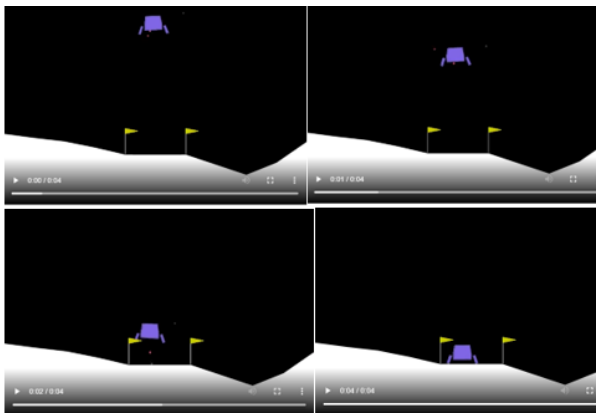


Hemos conseguido superar el Agente DQN simple en 63 episodios.

Agente Dueling + Noisy DQN. Inferencia y Resultados

El resultado de la inferencia para un solo episodio ha sido de 288,109 puntos y los frames del video resultante han sido:

Figura 5: Fotograma del mejor agente Dueling + Noisy DQN



El aspecto es muy similar al anterior, en los videos se puede apreciar como en este último no exite una ralentización que si se aprecia en el primero. De toda maneras, la diferencia entre ambas recompensas totales la encontramos en el hecho en el que este segundo es más eficiente y usa menos energia al aterrizar.

Comparaciones entre los Agentes

(Correspondiente al ejercicio 3)

Hemos pensado que para comparar los dos agentes lo mejor sería hacerlo durante su inferencia y no durante el entrenamiendo, ya que este segundo punto ya lo hemos visto anteriormente.

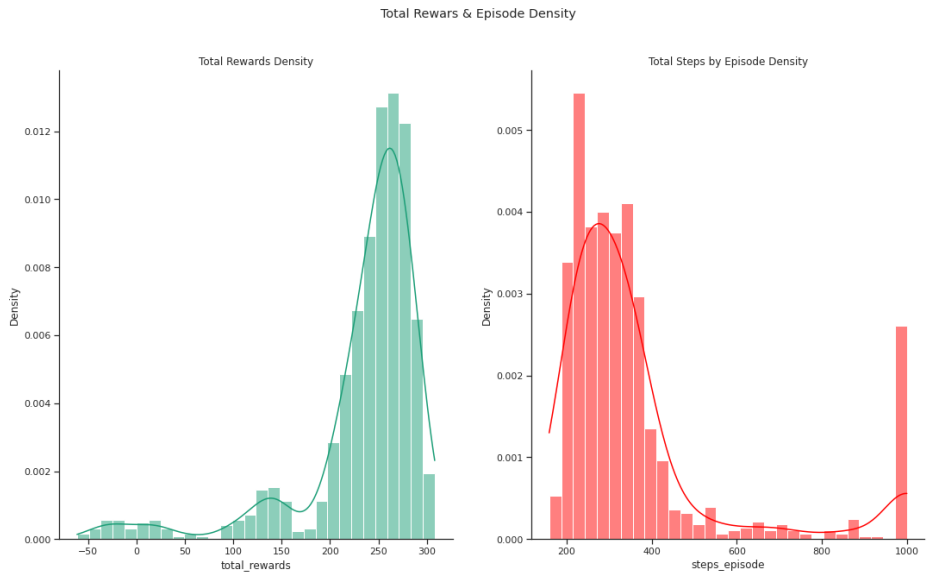
Hemos realizado 1000 episodios de prueba para cada agente y medido ciertas métricas como las recompensas totales, los pasos intermedios por episodio, y la media movil de recompensas de ambas pruebas. Nos hemos ayudado de varios graficos para realizar este análisis más intuitivo. Algunas de estos ersultados:

[INFO]: Best Simple DQN Inference Reward Mean: [233.7370009796463].
[INFO]: Best Dueling + Noisy DQN Inference Reward Mean: [237.5079157191881].
[INFO]: Best Simple DQN Inference Reward standard deviation : [65.60475366732524].
[INFO]: Best Dueling + Noisy DQN Reward standard deviation [61.953035934243346].

La media de recompesas obtenida para el primer agente será menor que la del segundo y lo puesto ocurrira con la varianza.

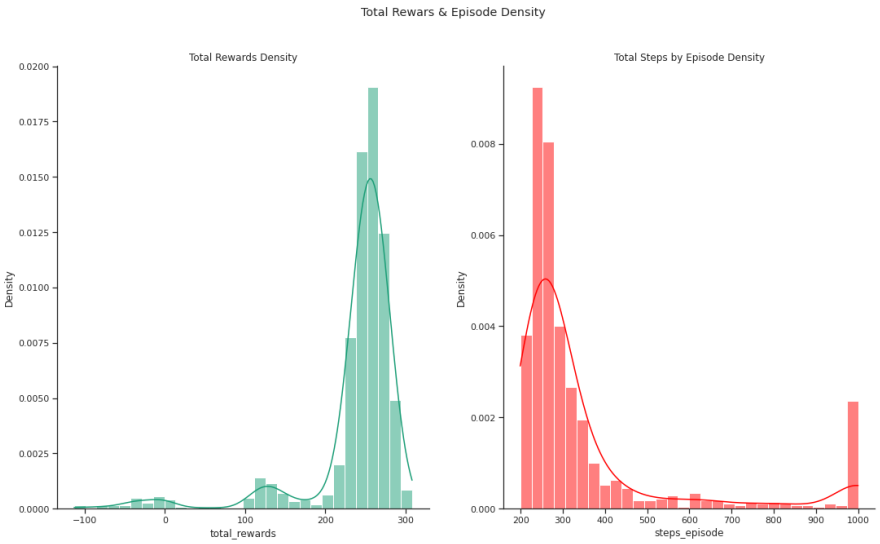
Para el primer agente:

Gráfico 6: Recompensas totales y distribución de pasos por episodio del agente DQN



Para el último agente:

Gráfico 7: Recompensas totales y distribución de pasos por episodio del agente Dueling + Noisy DQN



En ambos gráficos, observamos las recompensas y pasos obtenidos por episodios, en el segundo vemos cómo la distribución para las recomensas mayores es mayor, al igual, que los episodios se resolverám utilizando menores pasos que en el agente DQN Simple.

Gráfico 7: Evolución de la media móvil de recompensas de ambos agentes.

En este gráfico, que presenta las medias móviles del proceso de inferencia durante 1000 episodios observamos claramente como el agente Dueling + Noisy DQN es ligeramente superior al agente de DQN Inicial y es más compacto, con menor varianza.

Ambos superan con creces el umbral de 200 puntos, de este modo podemos concluir que hemos obtenido un agente eficiente.

Conclusiones

Durante esta actividad hemos repasado los distintos algoritmos y hemos practicado realizando los diversos tipos de agentes existentes en aprendizaje por refuerzo profundo afianzando todos los conceptos teoricos.

Al tratarse de un entorno simple, el modelo DQN alcanzaba nuestro umbral de manera eficiente pero intentando buscar un modelo que lo superará hemos podido trabajar con todos los existentes encontrando un mejor agente combinando la arquitectura Dueling DQN con las capas Noisy.

Referencias

Apuntes de la asignatura: Deep Q – Networks, Laura Ruiz Dern