Universitat Oberta de Catalunya

Aprendizaje por Refuerzo

Práctica

Implementación de un agente para la robótica espacial

Límite :
18.01.2023

Máster en Ciencia de Datos

Universitat Oberta de Catalunya

Leroy Deniz

leroy@uoc.edu

(+34) 669 987 109

https://leroydeniz.com

Índice

Contexto

Presentación

A lo largo de las tres partes de la asignatura hemos entrado en contacto con diferentes clases de algoritmos de aprendizaje por refuerzo que permiten solucionar problemas de control en una gran variedad de entornos.

Esta práctica, que se va a extender a lo largo de un mes aproximadamente, da la posibilidad de enfrentarse al diseño de un agente para solucionar un caso específico de robótica.

Atacaremos el problema a partir de la exploración del entorno y sus observaciones. Luego, pasaremos a la selección del algoritmo más oportuno para solucionar el entorno en cuestión. Finalmente, pasaremos por el entrenamiento y la prueba del agente, hasta llegar al análisis de su rendimiento.

Para ello, se presentará antes el entorno de referencia, y luego, se pasará a la implementación de un agente Deep Q-Network (DQN) que lo solucione. Después de estas dos primeras fases de toma de contacto con el problema, se buscará otro agente que pueda mejorar el rendimiento del agente DQN anteriormente implementado.

Objetivos

Los objetivos concretos de esta actividad son:

- Conocer y profundizar en el desarrollo de un entorno real que se pueda resolver mediante técnicas de aprendizaje por refuerzo.
- Aprender a aplicar y comparar diferentes métodos de aprendizaje por refuerzo para poder seleccionar el más adecuado a un entorno y problemática concretos.
- Saber implementar los diferentes métodos, basados en soluciones tabulares y soluciones aproximadas, para resolver un problema concreto.
- Extraer conclusiones a partir de los resultados obtenidos.

Entorno

Estamos trabajando sobre un problema de robótica espacial y en particular queremos solucionar el problema de aterrizaje propio, por ejemplo, de drones autónomos.

Para ello, se elige **lunar-lander** como entorno simplificado. El entorno se puede encontrar en el siguiente enlace: https://github.com/openai/gym/blob/master/gym/envs/box2d/lunar_lander.py

Lunar Lander consiste en una nave espacial que debe aterrizar en un lugar determinado del campo de observación. El agente conduce la nave y su objetivo es conseguir aterrizar en la pista de aterrizaje, coordenadas (0,0), y llegar con velocidad 0.

La nave consta de tres motores (izquierda, derecha y el principal que tiene debajo) que le permiten ir corrigiendo su rumbo hasta llegar a destino.

Las acciones que puede realizar la nave (espacio de acciones) son discretas.

Las recompensas obtenidas a lo largo del proceso de aterrizaje dependen de las acciones que se toman y del resultado que se deriva de ellas.

- Desplazarse de arriba a abajo, hasta la zona de aterrizaje, puede resultar en [+100,+140] puntos
- Si se estrella al suelo, pierde -100 puntos
- Si consigue aterrizar en la zona de aterrizaje (velocidad 0), gana +100 puntos
- Si aterriza, pero no en la zona de aterrizaje (fuera de las banderas amarillas) se pierden puntos
- El contacto de una pata con el suelo recibe +10 puntos (si se pierde contacto después de aterrizar, se pierden puntos)
- Cada vez que enciende el motor principal pierde -0.3 puntos
- Cada vez que enciende uno de los motores de izquierda o derecha, pierde -0.03 puntos

La solución óptima es aquella en la que el agente, con un desplazamiento eficiente, consigue aterrizar en la zona de aterrizaje (0,0), tocando con las dos patas en el suelo y con velocidad nula. Se considera que el agente ha aprendido a realizar la tarea (i.e. el "juego" termina) cuando obtiene una media de al menos 200 puntos durante 100 episodios consecutivos.

Ejercicio 1

Ejercicio 1.1

Se pide explorar el entorno y representar una ejecución aleatoria.

Para la exploración se utiliza la versión 2 del entorno propuesto, denominado ***LunarLander-v2***, a través del cual se obtiene utilizando el mismo con el comando *make* en la variable *env*. En el momento de hacer el reset del entorno, se obtiene un array de Numpy con los datos asociados al mismo. El orden en el que se presenta y su contenido se pueden aprecier a continuación en un ejemplo que está, además, publicado en el código adjunto:

```
array([
        0.00570612  ,       Posición de la nave en X
        1.3990337   ,       Posición de la nave en Y
        0.5779653   ,       Velocidad lineal en X
       -0.5282997   ,       Velocidad lineal en Y
       -0.0066053   ,       Ángulo de movimiento
       -0.13091765  ,       Velocidad angular de la nave
        0.          ,       Booleano activado si la pata izquierda toca el suelo lunar
        0.                  Booleano activado si la pata derecha toca el suelo lunar
], dtype=float32)
```

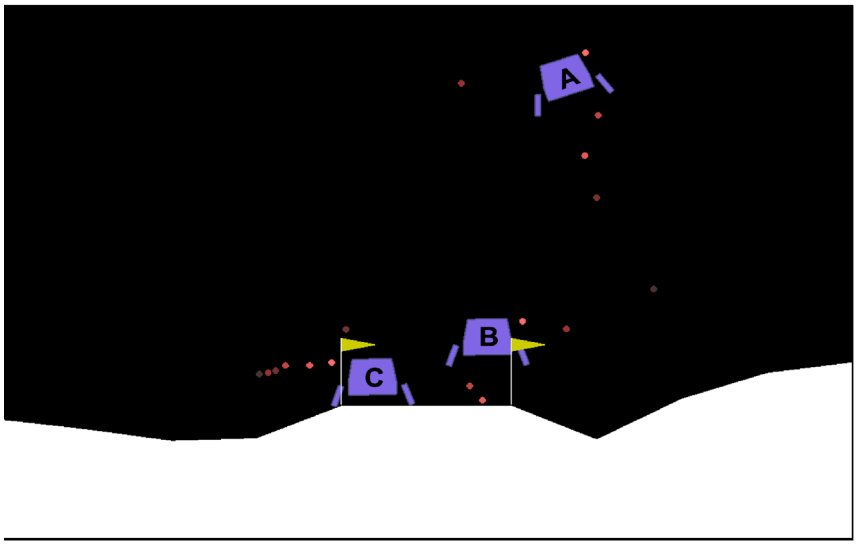

FIGURA 1
Movimiento objetivo del agente.
Fuente: https://www.mdpi.com/2076-3417/12/21/10947

Se tienen disponibles también en el entorno el umbral de recompensas (*env.reward_range*) que oscila entre (-inf, inf), el número de pasos máximo por episodios (*env.spec.max_episode_steps*) que ascienden a 1000.

Ejercicio 1.2

Explicar los posibles espacios de observaciones y de acciones.

La dimensión del espacio de acciones (*env.action_space.n*) que son en total 4, cuyos movimientos disponibles son estos:

- no hacer nada
- activar el motor izquierdo
- activar el motor principal debajo
- activar el motor derecho

El vector observation_low ofrece las cotas inferiores que pueden tomar cada uno de los parémetros, mostrando lo opuesto el vector observation_high.

```
Observation shape: (8,)
Observation high:  [ 1.5, 1.5,    5.,   5.,   3.1415927,   5., 1.,   1. ]
Observation low:   [-1.5, -1.5,   -5., -5., -3.1415927, -5., -0., -0. ]
```

De esta información sabemos que, los valores de X e Y oscilarán ambos entre -1.5 y 1.5, así como la velocidad lineal en X e Y lo hará entre -5 y 5, el ángulo de movimiento entre -π y π, la velocidad angular entre -5 y 5 y, claramente por ser booleanos, los identificadores si alguna de las patas de la nave ha tocado el suelo tomarán valores 0 o 1.

En la práctica anterior donde se trabajó con un entorno cuya información era una cantidad considerable de píxeles, se utilizaba una red convolucional para interpretar esta información y obtener los datos necesarios para el aprendizaje. Por el contrario, en este caso el entorno ya provee un array con la información relevante necesaria para el proceso de entrenamiento y aprendizaje, por lo que no es necesario procesarla; de ahí que se descarta la red convolucional.

Ejercicio 2

Ejercicio 2.1

Implementar un agente DQN para el entorno lunar-lander.

Antes de implementar un agente, es necesario definir la red neuronal que estará por detrás realizando el trabajo operativo. Para este caso inicial, se elige utiliza una red Deep Q-learning como solicita el enunciad, tomando como base la misma que se utilizó en la práctica 2, con la salvedad de que, en esta oportunidad, se elimina la red convolucional inicial puesto que no se está trabajando con imágenes que la requieran.

Además, a la red lineal que resta, se le añade una capa oculta más, por lo que ahora se cuenta con una capa de entrada, dos ocultas y una de salida. La definición queda entonces establecida de la siguiente forma:

```
self.red_lineal = nn.Sequential(
      nn.Linear(self.input_shape, 512, bias=True),
      nn.ReLU(),
      nn.Linear(512, 256, bias=True),
      nn.ReLU(),
      nn.Linear(256, 512, bias=True),
      nn.ReLU(),
      nn.Linear(512, self.n_outputs, bias=True)
 )
```

Ahora bien, siguiendo la línea establecida en la práctica anterior, el optimizador es Adam ya que favorece la convergencia rápida en este tipo de redes y la política de toma de acciones se hace a través de *epsilon-greedy*.

Una vez la red está definida, se procede a la definición del agente. De la misma forma que el paso anterior, se utiliza el mismo agente que en la práctica anterior, con los mismos parámetros y configuraciones, adaptándolo para eliminar, por ejemplo, *stack_frame* que para este dominio no corresponde.

El código del agente es similar al anterior, puede verse en el notebook adjunto en el apartado 2.1.

Ejercicio 2.2

Entrenar el agente DQN y buscar los valores de los hiperparámetros que obtengan un alto rendimiento del agente. Para ello, es necesario listar los hiperparámetros bajo estudio y presentar las gráficas de las métricas que describen el aprendizaje.

Para este apartado, se entrenarán cuatro agentes con cambios en los hiperparámetros, en pro de una búsqueda de optimización, tal y como se aprecia en la tabla de la Figura 2.

DNN_UPD	100	100	100	100
DNN_SYNC	5000	5000	5000	5000
BATCH_SIZE	32	64	32	10
GAMMA	0.99	0.70	0.001	0.99
MAX_EPISODES	1500	1500	1500	1500
MEAN TRAINING REWARD	**-190.12**	**-253.73**	**-201.36**	**-345.37**
MAX TRAINING REWARD	**-95.24**	**-56.40**	**-123.94**	**-96.83**

FIGURA 2
Configuración de los agntes testeados.
Fuente: producción propia

Con los distintos hiperparámetros probados, la mejor configuración ha sido la primera de todas (Configuración 0), puesto que, aunque el valor máximo obtenido no alcanza a ser el más grande de los cuatro, el promedio de todas las partidas arroja el mejor de ellos.

De todas formas, cabe destacar que con cualquiera de los cuatro agentes entrenados, los resultados son bastante malos, aunque el único agente que consigue cierta estabilización a partir del episodio 900 es el que tiene la Configuración 2 (línea verde en la Figura 3).

FIGURA 3
Comparación de los cuatro agentes
Fuente: producción propia

Ejercicio 2.3

Probar el agente entrenado en el entorno de prueba. Visualizar su comportamiento (a través de gráficas de las métricas más oportunas).

Como puede verse en la gráfica de la Figura 4, el único de los agentes que logra al menos puntuaciones positivas, aunque esporádicas, es el agente con la Configuración 1; por lo que, a pesar de no mostrar un comportamiento estable, sería la mejor opción de cara a una comparación de modelos, ya que con los demás, en ningún momento se consigue siquiera un resultado decente.

Se elige entonces, priorizando la mayor recompensa obtenida por un agente, al mismo con la configuración 1, por lo expuesto anteriormente como el agente que mejor resultados arroja. **

FIGURA 4
Reward en 200 jugadas
Fuente: producción propia

** Por espacio y longitud del documento, el resto de las pruebas están en el notebook adjunto.

Ejercicio 3

En esta parte se pide implementar otro agente, entre aquellos que hemos visto a lo largo de la asignatura, que pueda solucionar el problema de robótica espacial de forma más eficiente con respecto al agente DQN.

Ejercicio 3.1

Implementar el agente identificado en el entorno lunar-lander. Justificar las razones que han llevado a probar este tipo de agente. Detallar qué tipos de problemas se espera se puedan solucionar con respecto a la implementación DQN anterior.

Para este ejercicio, se ha optado por imlpementar un agente REINFORCE que, en la práctica anterior, alcanzó buenos resultados y la estabilidad a los pocos episodios, mejorando el tiempo de entrenamiento puesto que de ese punto en adelante, ni mejoraba ni empeoraba.

Quedan descartados los agentes de soluciones tabulares puesto que el espacio de observación en este caso es demasiado grande y sería imposible de manejar una tabla de las dimensiones necesarias.

De la misma forma que antes, se adapta el código que se tenía de la PEC2 eliminándole la red convolucional puesto que no se está trabajando con imágenes, ni modificando la configuración de los hiperparámetros a este punto.

Se utiliza Adam como optimizador y se adapta la función *get_action_prob* para obtener las probabilidades de posibles acciones. El código del agente ***REINFORCEAgent*** es igual, no tiene cambios significativos además del *stack_frame* que se suprime.

Ejercicio 3.2

Entrenar el agente identificado y buscar los valores de los hiperpárametros que obtengan el rendimiento "óptimo" del agente.

Se entrena el agente con 1500 casos, el mismo número de casos que los demás agentes, permitiendo así una comparativa no solamente en el mismo número de episodios, sino también la visualización de la estabilidad del agente.

Se han obtenido unos valores significativamente mejores que para los agentes DQN, alcanzando un *mean_reward* de 200.8 puntos en los últimos 100 episodios, lo que supera por mucho los valores negativos que los otros presentaban.

La configuración utilizada y cuyos parámetros se entienden óptimos, es la siguiente:

- Learning rate: 0.005
- Batch size: 8
- Memory size: 8000
- Gamma: 0.99
- Max episodes: 1500
- Max score: 200

Ejercicio 3.3

Analizar el comportamiento del agente identificado entrenado en el entorno de prueba y compararlo con el agente implementado en el punto 2 (a través de gráficas de las métricas más oportunas).

A pesar de esta definición inicial de entrenamiento con 1500 casos, el agente utilizando REINFORCE alcanzó un promedio de recompensas de 200 puntos en los últimos 100 casos con 1473 episodios insumiendo 55 minutos en el entrenamiento. En el GIF que muestra una partida aleatoria utilizando ese agente, se puede ver cómo la suavemente nave aluniza y apaga sus motores.

Por su parte, la gráfica que muestra la evolución de las recompensas y la media de estas en los últimos 100 episodios, se aprecia cómo esta mejora constantemente hasta situarse completamente sobre el umbral de recompensas positivas, alcanzando, como se ha mencionado anteriormente, superar el umbral de éxito casi al final de su entrenamiento.

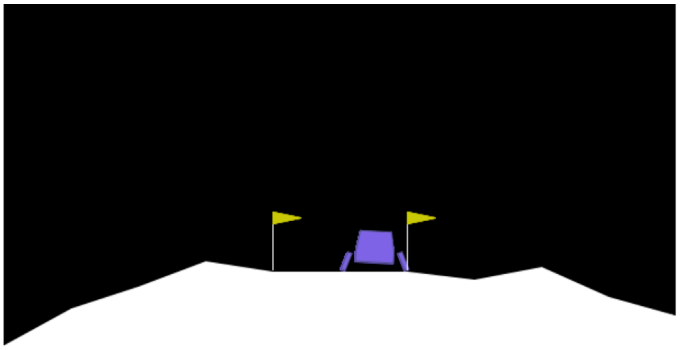

FIGURA 5
Final del episodio utilizando REINFORCE
Fuente: producción propia

El modelo ha sido exportado al directorio *models* bajo el título de ***reinforce_agent.pth***, junto a los demás modelos exportados de los otros entrenamientos de DQN.

Comparando este modelo con el anterior elegido con un agente DQN, las gráficas muestran claramente cómo el agente Reinforce supera ampliamente en recompensas y, por descontado, en aprendizaje al agente DQN, reflejando además una estabilidad en la zona de recompensas positivas frente a la fluctuación de recompensas negativas del agente DQN. Tomando el cuenta la media de los últimos 100 episodios, destaca totalmente la estabilidad del Reinforce frente al primero.

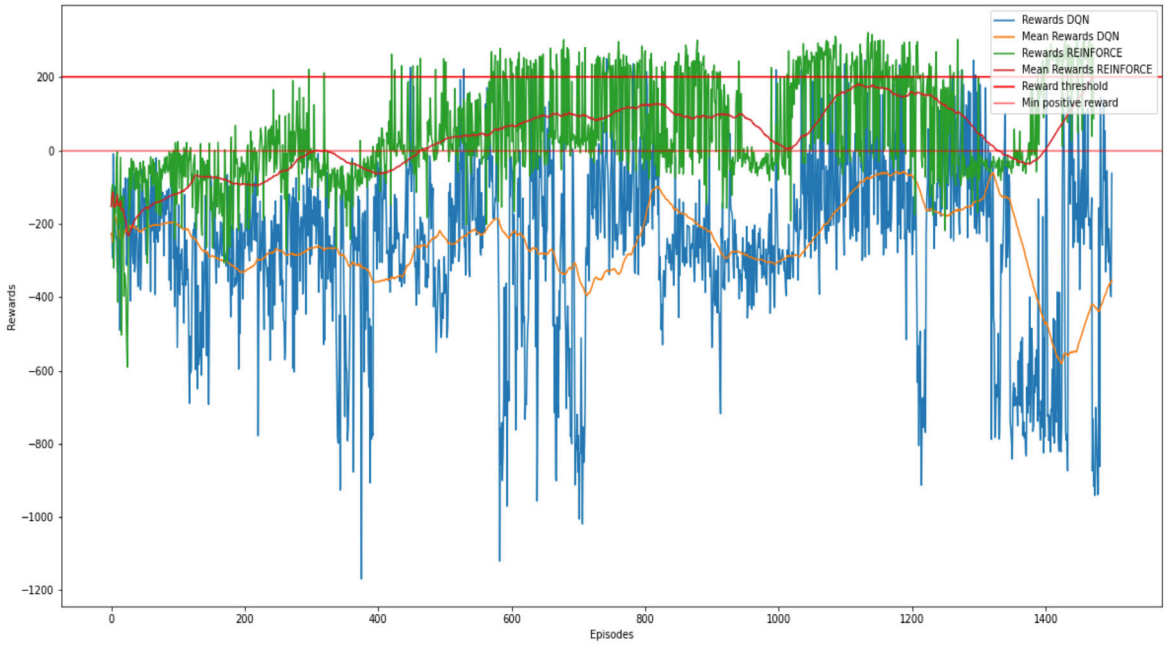

FIGURA 6
Recompensas obtenidas a lo largo del entrenamiento
Fuente: producción propia

Conclusiones

El algoritmo ***Deep Q-Network*** permite utilizar el algoritmo Q-learning con redes neuronales profundas, capaces de aproximar muy bien la función Q siendo esta una función no lineal.

Por el contrario, el algoritmo ***REINFORCE*** es un algoritmo ***policy-based***, es decir, que en lugar de realizar el entrenamiento de una red neuronal para que genere los valores de estados o acciones, entrena una red neuronal para indicar cuál puede ser la sigueinte acción a realizar.

A partir de los datos obtenidos en los apartados anteriores y tomando en cuenta las definiciones de cada uno de los algoritmos utilizados, se podría afirmar que el algoritmo ***REINFORCE*** arroja mejores resultados basándonos en las métricas obtenidas en los entrenamientos anteriores y, además, en la simplicidad del modelo establecido.

Cuando el entorno es tan grande como el caso que nos atañe, el ***DQN*** tiene una cantidad enorme de estados posibles, su solución tabular requiere de una tabla demasiado grande y necesita una red neuronal que aproxime la función de valor Q, por lo que el algoritmo al ser ***value-based*** puede no ser la mejor opción.

Sin embargo, la mejora que ofrece ***REINFORCE***, puede darse gracias a que es, como ya se comentó anteriormente, es un algoritmo ***policy-based***, que no busca aprender del entorno los valores óptimos para aproximar la función de valor, sino que busca aproximar la política óptima de cara a obtener las mejores recompensas.

La gráfica de la figura 6 refleja cómo ***REINFORCE***, apenas consigue buenos resultados, se mantiene en ese umbral de recompensas positivas alcanzando buenos puntajes, mientras que ***DQN*** no termina de converger, y menos aún en recompensas positivas.

El algoritmo ***REINFORCE*** utiliza una distribución de probabilidades que le permiten tomar la decisión de qué acción realizar, lo que le da una ventaja sobre ***DQN*** que necesita de epsilon-greedy para poder explorar.

Bibliografía

1. Berzal Galiano, F. (s. f.). ***Redes Neuronales & Deep Learning***. Universidad de Granada. Recuperado 2 de enero de 2023, de https://elvex.ugr.es/decsai/deep-learning/

2. Casas Roma, J. (s. f.). ***GitHub - jcasasr/Aprendizaje-por-refuerzo: Ejercicios resueltos de la asignatura «Aprendizaje por refuerzo» del Máster Universitario en Ciencia de Datos***. GitHub. Recuperado 2 de enero de 2023, de https://github.com/jcasasr/Aprendizaje-por-refuerzo

3. Christodoulou, P. (s. f.). ***GitHub - p-christ/Deep-Reinforcement-Learning-Algorithms-with-PyTorch: PyTorch implementations of deep reinforcement learning algorithms and environments***. GitHub. Recuperado 2 de enero de 2023, de https://github.com/p-christ/Deep-Reinforcement-Learning-Algorithms-with-PyTorch

4. Foy, P. (2022, 12 diciembre). ***Implementing Deep Reinforcement Learning with PyTorch: Deep Q-Learning***. MLQ.ai. Recuperado 2 de enero de 2023, de https://www.mlq.ai/deep-reinforcement-learning-pytorch-implementation/

5. Hubbs, C. (2021, 12 diciembre). ***Learning Reinforcement Learning: REINFORCE with PyTorch!*** Medium. https://towardsdatascience.com/learning-reinforcement-learning-reinforce-with-pytorch-5e8ad7fc7da0

6. Kansal, S. & Martin, B. (s. f.). ***Reinforcement Q-Learning from Scratch in Python with OpenAI Gym***. Learn Data Science. Recuperado 2 de enero de 2023, de https://www.learndatasci.com/tutorials/reinforcement-q-learning-scratch-python-openai-gym/

7. NeuralNet.ai. (2018, 24 noviembre). ***Coding a Deep Q Network in PyTorch – NeuralNet.ai***. Recuperado 2 de enero de 2023, de https://www.neuralnet.ai/coding-a-deep-q-network-in-pytorch/

8. PL team. (s. f.). ***How to train a Deep Q Network — PyTorch Lightning 1.8.6 documentation***. Recuperado 2 de enero de 2023, de https://pytorch-lightning.readthedocs.io/en/stable/notebooks/lightning_examples/reinforce-learning-DQN.html

9. Sanghi, N. (2021). ***Deep Reinforcement Learning with Python: With Pytorch, Tensorflow and Openai Gym***. Apress.

10. Singh, U. (2021, 9 diciembre). ***Deep Q-Network with Pytorch*** - Unnat Singh. Medium. https://unnatsingh.medium.com/deep-q-network-with-pytorch-d1ca6f40bfda

11. Terra, A., Inam, R. & Fersman, E. (2022). ***BEERL: Both Ends Explanations for Reinforcement Learning***. Applied Sciences, 12(21), 10947. https://doi.org/10.3390/app122110947

Universitat
Oberta
de Catalunya

Leroy Deniz

Aprendizaje por Refuerzo

Máster en Ciencia de Datos

Universitat Oberta de Catalunya