Implementação do Lunar Lander com Aprendizado por Reforço Utilizando Deep Q Learning

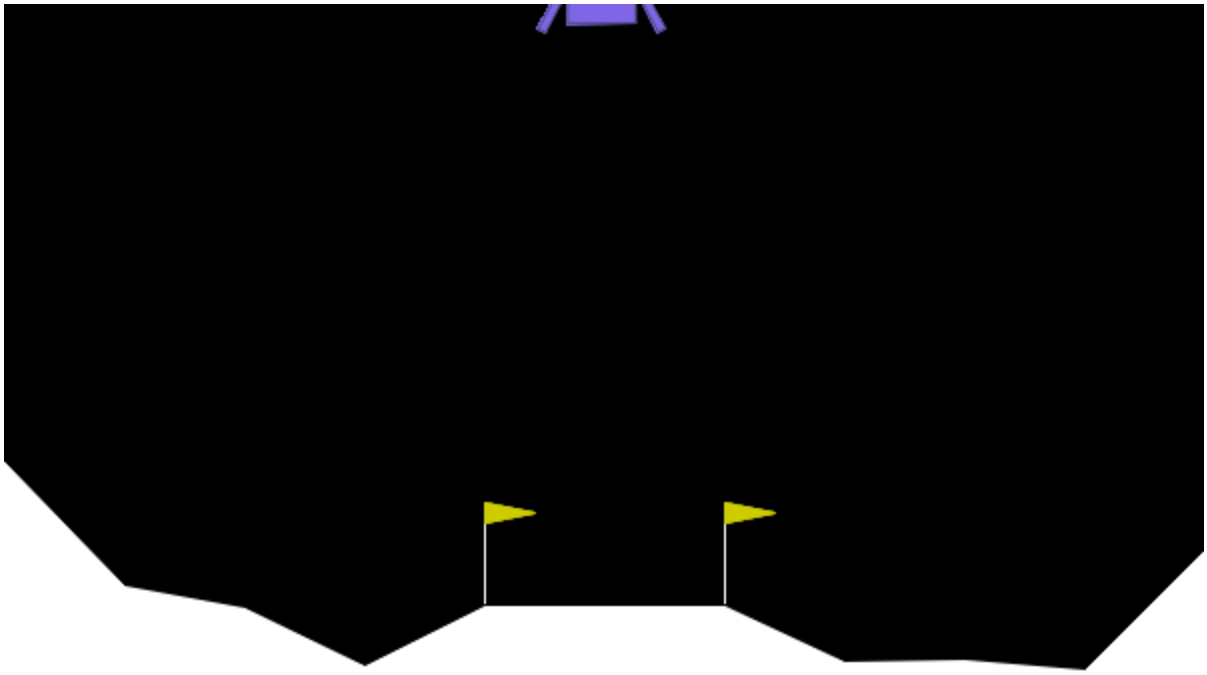
Introdução

O Deep Q-Learning (DQN) é um tipo de aprendizado por reforço que atrela as ideias do Q-Learning clássico com as redes neurais. A principal diferença desse método está na forma de computar uma ação para o presente estado. Ao invés de armazenar o aprendizado em uma Q-Table, o agente utiliza uma rede neural para fazer essa previsão.

O Lunar Lander é um ambiente de simulação da OpenAI, cujo principal objetivo é controlar um módulo lunar para aterrizar em uma superfície irregular gerada aleatoriamente. O módulo é controlado por meio de um conjunto de comandos que determinam a ejeção dos seus propulsores. A cada *step*, o agente recebe as informações do estados, sendo elas:

- Posição (x, y) do módulo
- Velocidade linear (x, y) do módulo
- Velocidade angular
- Booleanos que indicam se cada perna do módulo entrou em contato com o chão.

A cada vez que o módulo aterriza, ou seja, ambas as suas pernas entram em contato com o chão, ou que ocorra uma batida, representada por um contato com o chão de uma parte do módulo que não seja uma de suas perna, o ambiente é reiniciado com um numa nova superfície e uma nova posição inicial.



Implementação

O objetivo desse experimento era de treinar um agente para controlar o pouso do *Lunar Lander* usando as técnicas do DQN. Para isso foi implementado um código usando a linguagem python. Esse código abstraia uma série de componentes usando a programação orientada à objetos para facilitar a criação desse agente. Abaixo, estão listada as classes criadas:

- `State` : Responsável por encapsular o estado do ambiente. Ela armazena a observação (dados como posição, velocidade, etc.), a recompensa, se o episódio foi terminado ou truncado, e outras informações fornecidas pelo ambiente.
- `DeepQNet` : Esta classe define a rede neural usada para aproximar a função Q. O modelo é composto por várias camadas densas (Dense) com uma camada de entrada que recebe o estado atual como entrada e uma camada de saída com o número de ações possíveis.
- `Agent` : Gerencia o agente que interage com o ambiente, toma decisões e atualiza a rede neural com base nas transições de estado.

O processo de treinamento do agente envolve a interação contínua com o ambiente e a atualização da rede neural com base nas transições de estado coletadas ao longo de várias iterações. O agente começa explorando o ambiente de forma aleatória, tomando ações baseadas na política ϵ**-greedy**. As fases desse treinamento foram listadas abaixo:

1. Exploração inicial: Antes de iniciar o aprendizado na rede neural, o agente toma decisões de ação aleatórias e coleta as informações em um buffer de 5-uplas (estado, ação tomada, recompensa recebida, próximo estado, flag de episodio finalizado) com as observações resultantes.
2. Exploração (*Exploration*) / Exploração (*Exploitation*): Usando a política ϵ**-greedy** o agente usa a rede neural para prever a próxima ação, fornecendo à rede as informações do estado.
3. Atualização da rede: A atualização da rede é feita a partir do *backpropagation* usando as equações de estado de Bellman e uma amostra do buffer como parâmetros.
4. Estabilização da rede: Para estabilizar o treinamento, uma rede alvo é utilizada. Periodicamente, os pesos da rede principal (que gera os valores de Q estimados) são copiados para a rede alvo. A rede alvo não é atualizada com a mesma frequência que a rede principal, o que ajuda a suavizar as flutuações nos valores de Q e melhora a estabilidade do treinamento.

A medida que o agente interage repetidamente com o ambiente, ele gradualmente converge para uma política que maximiza a recompensa acumulada ao longo do tempo. Em termos práticos, isso significa que o agente começa a prever as melhores ações para executar em cada estado, com base nas estimativas da função Q fornecida pela rede neural.