Reinforcement Learning
Treinamento de veículos autônomos com HighWayEnv

Beatriz Rianho Bernardino

20 de maio de 2023

Insper

1- Introdução

Com o advento da tecnologia na sociedade, cada vez mais tem-se discutido sobre a viabilidade de carros autônomos. Para que isso seja possível, existem diversos métodos de treinamento de modelos para que seu refinamento fique ideal. Dentre eles, Reinforcement Learning tem sido uma solução amplamente discutida, por como o agente interage com o ambiente através de diferentes políticas e recompensas.

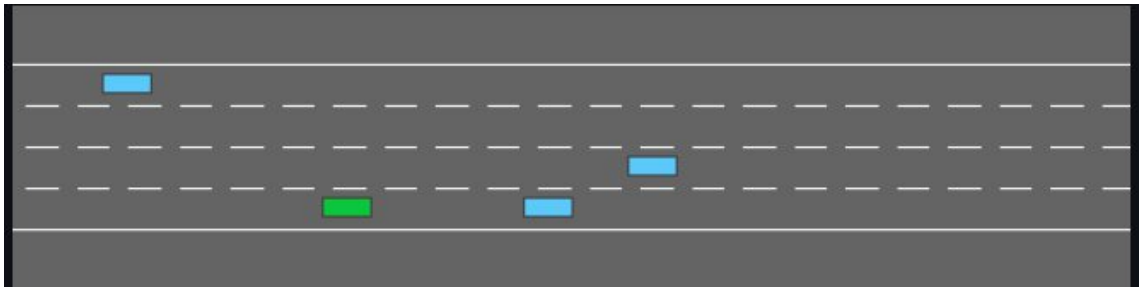
2- Objetivos

O objetivo do presente projeto é implementar e observar o comportamento dos algoritmos de reinforcement learning DQN e PPO no contexto de carros autônomos através do ambiente HighWayEnv da biblioteca gym.

3- Métodos

3.1 Actions

Para uma análise comparativa entre os algoritmos, foi utilizado um tutorial disponibilizado pela Farama-Foundation, o qual ensina a utilizar o ambiente e aplicar um algoritmo de treinamento. Desse modo, o ambiente é composto por uma rodovia com tráfego de outros veículos, no qual o agente precisa obter uma alta velocidade enquanto evita colisões. Além disso, ficar no lado direito também é recompensado.



Assim, o ambiente é do tipo single-agent e cooperativo, além de discreto e com a possibilidade de realizar ao todo 5 ações:

```
ACTIONS_ALL = {
0: 'LANE_LEFT',
1: 'IDLE',
```

```
2: 'LANE_RIGHT',
3: 'FASTER',
4: 'SLOWER'
}
```

3.2 Observation

O ambiente está configurado com uma observation padrão do tipo “kinematics”, fornecendo uma matriz com com quatro informações: [x, y, vx, vy], sendo o primeiro array sempre o do agente, e o resto referente aos outros veículos. Esses valores são normalizados, resultando em um range de [100, 100, 20, 20].

Vehicle	x	y	v_x	v_y
ego-vehicle	0.05	0.04	0.75	0
vehicle 1	-0.1	0.04	0.6	0
vehicle 2	0.13	0.08	0.675	0
...	...	...	...	...
vehicle V	0.222	0.105	0.9	0.025

3.3 Rewards

A função de rewards do ambiente é definida da seguinte forma:

$$R(s,a) = \alpha \cdot \frac{v - v_{min}}{v_{max} - v_{min}} - b, collision$$

Onde v, v_{min}, v_{max} são a velocidade atual, máxima e mínima do agente, e α e b são coeficientes.

O ambiente não permite uma reward final negativa pois pode influenciar o modelo a terminar um episódio antes, causando uma colisão, do que continuar e arriscar não ter uma reward boa caso uma trajetória satisfatória não for encontrada.

3.4 Algoritmos

Para o presente projeto, foi decidido que dois algoritmos seriam utilizados: o DQN e o PPO. O DQN é um algoritmo que combina o Q-learning com redes neurais, para realizar a aproximação da função Q. A rede neural recebe um estado como entrada e produz uma estimativa dos valores Q para cada ação possível nesse estado. O DQN é um algoritmo do tipo off-policy, o qual aprende a função Q de acordo com uma política de epsilon-greedy durante a exploração, ou seja, seleciona a ação com base em um trade-off entre ação ótima e exploração aleatória . Já o PPO é um algoritmo on-policy, que possui como objetivo maximizar a função de recompensa acumulada através da otimização de uma função de perda que mede a diferença entre a política atualizada e a política anterior. A otimização é realizada por meio de um método de otimização de gradiente, como o método de gradiente estocástico (SGD).

Os algoritmos foram escolhidos para treinar o modelo para ser possível observar como suas diferenças afetam a eficácia do aprendizado em um ambiente como o HighEnv. Dentre essas divergências, é interessante observar a diferença na estabilidade de treinamento, e como a função de aprendizado influencia o modelo.

3.5 Hiperparâmetros

Como o estado fornecido é uma matriz, foi utilizada a política ‘CnnPolicy’ para o DQN, por sua melhor eficiência com as convoluções neurais. Desse modo, os hiperparâmetros utilizados foram:

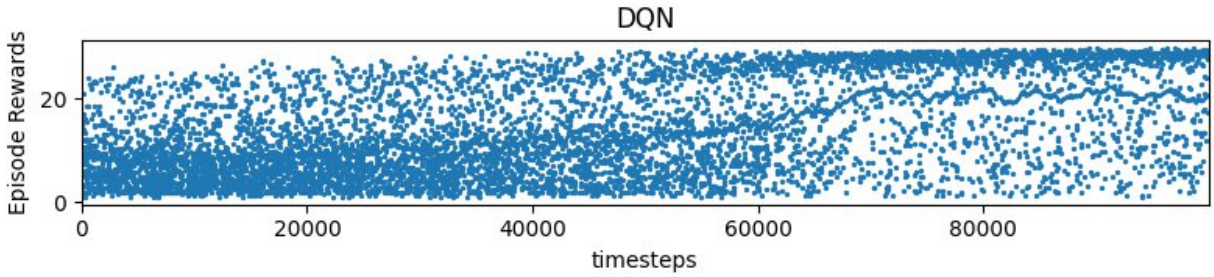
```
                learning_rate=5e-4,
                buffer_size=15000,
                learning_starts=200,
                batch_size=32,
                gamma=0.8,
                train_freq=1,
                gradient_steps=1,
                target_update_interval=50,
                exploration_fraction=0.7,
```

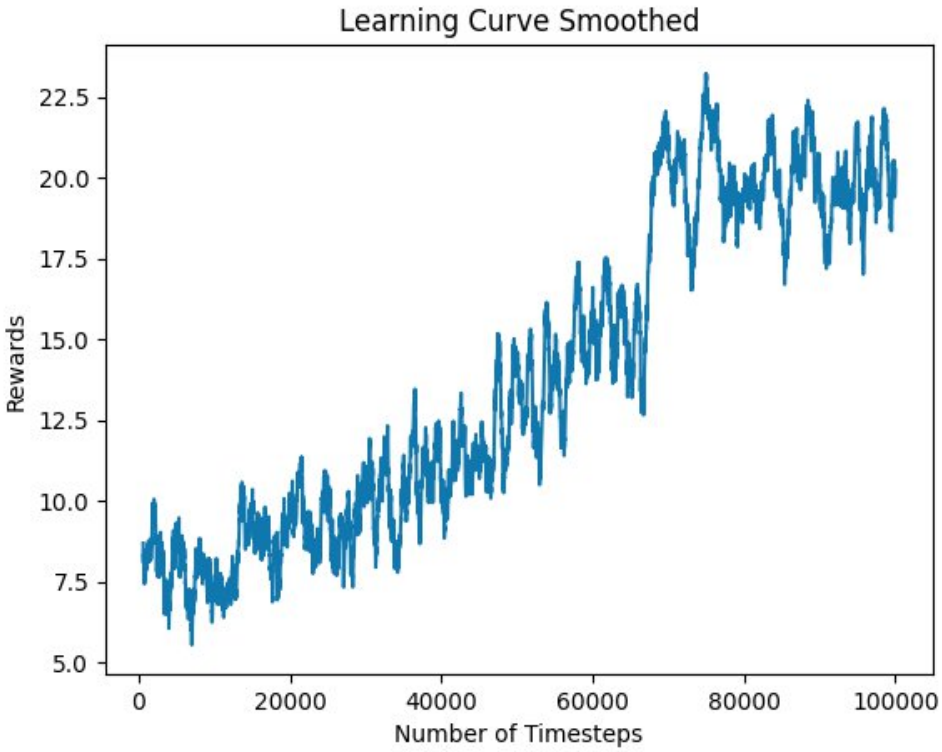
Já com o PPO, por mais que a ‘CnnPolicy’ pudesse também resultar em um melhor resultado, a ‘MlpPolicy’ já se mostrou suficiente para um resultado efetivo. Assim, os seguintes hiperparâmetros foram utilizados:

```
            policy_kwargs=dict(net_arch=[dict(pi=[256,
256],vf=[256, 256])]),
            n_steps=batch_size * 12,
            batch_size=batch_size,
            n_epochs=10,
            learning_rate=0.0001,
            gamma=0.95,
```

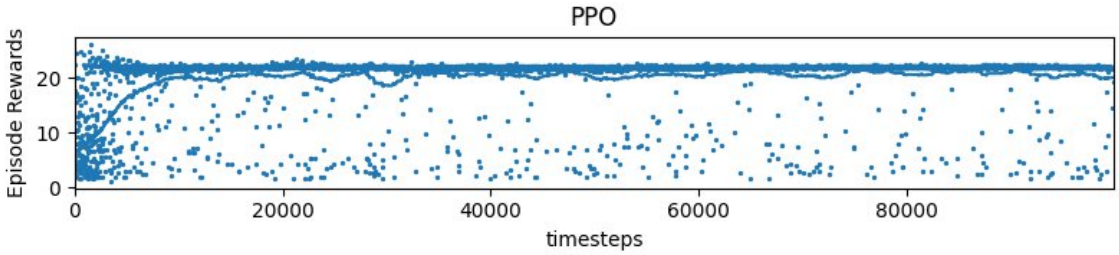
4- Resultados

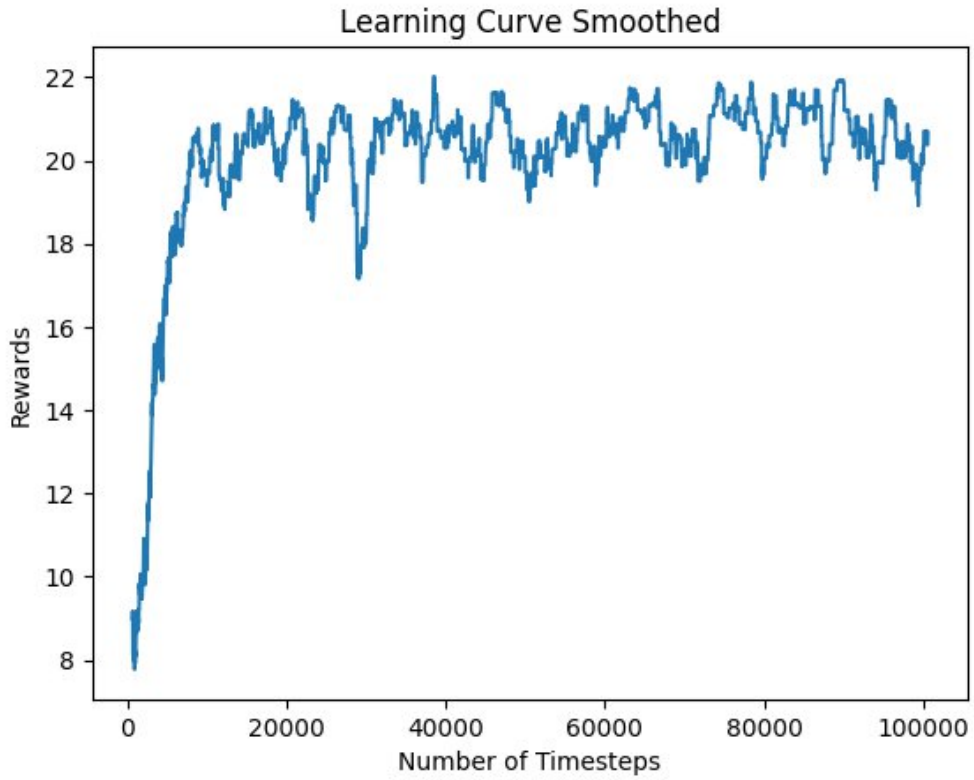
Após o treinamento foram obtidos os seguintes resultados para o DQN:





E para o PPO:





Em ambos é possível observar uma alta instabilidade nos resultados, mas que existe uma convergência no resultado, indicando um aprendizado efetivo do modelo. É possível observar pela simulação feita no código que o agente se comporta como esperado na maioria dos casos, com uma reward média de 20.76 para o PPO e 20.46 para DQN**.**

5- Conclusão

Após analisar a eficiência dos modelos pode-se concluir que o PPO, algoritmo on-policy, é melhor para lidar com o ambiente escolhido, dado que se estabiliza consideravelmente antes que o DQN. Porém, ambos os algoritmos apresentaram uma alta instabilidade de treinamento, podendo não resultar em um modelo final bom. Para futuros projetos, seria ideal considerar outras políticas de treinamento, realizar um ajuste fino dos hiperparâmetros utilizados e alterar as configurações de rewards do ambiente, para que o modelo final ficasse ideal para uso.

6- Bibliografia

https://github.com/Farama-Foundation/HighwayEnv

https://stable-baselines.readthedocs.io/en/master/guide/examples.html

https://colab.research.google.com/github/Stable-Baselines-Team/rl-colab-notebooks/blob/master/monitor_training.ipynb#scrollTo=mPXYbV39DiCj