Relatório do Laboratório 13: Deep Q-Learning

Isabelle Ferreira de Oliveira
CT-213 - Engenharia da Computação 2020
Instituto Tecnológico de Aeronáutica (ITA)
São José dos Campos, Brasil
isabelle.ferreira3000@gmail.com

***Resumo*—Esse relatório documenta a resolução do problema de Mountain Car no ambiente OpenAI Gym usando o algoritmo seminal de Deep Reinforcement Learning: o Deep Q-Learning/Deep Q-Networks (DQN).**

***Index Terms*—Mountain Car, OpenAI Gym, Deep Reinforcement Learning, Deep Q-Learning, Deep Q-Networks**

I. Implementação

A. Implementação da Definição da Rede Neural

Essa primeira etapa se tratou da implementação do método build_model() da classe DQNAgent de dqn_agent.py, script fornecido no código base do laboratório. Nesse método, era preciso construir uma rede em Keras de acordo com as especificações apresentadas na Tabela 3 do roteiro do laboratório [1].

Essa implementação foi feita de forma bastante análoga à maneira do laboratório 8 [2], ou seja, seguindo o apresentado no pseudo-código em Python a seguir.

```
# Adds the first layer
model.add(layers.Dense(num_neurons,
   activation=activations.some_function,
   input_dim=state_size))

# Adds another layer (not first)
model.add(layers.Dense(num_neurons,
   activation=activations.some_function))
```

Vale ressaltar que, para atender os critérios requisitados, some_function do pseudo-código acima se tratou de *relu* para as duas primeiras camadas, e de *linear* para terceira camada. Além disso, num_neurons foram 24, 24 e action_size para as primeira, segunda e terceira camada, respectivamente.

1) Função epsilon_greedy_action: A política epsilon-greedy foi implementada da seguinte maneira: gerou-se um número aleatório entre 0 e 1 e, caso esse valor aleatório seja menor que epsilon, então uma ação aleatória é escolhida; caso contrário, é escolhida a ação gulosa, através da chamada de greedy_action.

2) Função greedy_action: Conforme sugerido na seção Dicas do roteiro [1], foi pegue o índice do máximo elemento do array q[state], que é a tabela action-value para o estado naquele momento, e foi retornado esse valor.

3) Função get_greedy_action para algoritmo Sarsa: Retorna a função epsilon_greedy_action, aplicada na tabela action-value q, no estado em questão e com o episolon especificado.

4) Função learn para algoritmo Sarsa: O aprendizado é feito atualizando o valor da tabela de action-value, acrescentando ao valor anterior o resultado de $\alpha * (recompensa + \gamma * q[nextstate][nextaction] - q[state][action])$.

5) Função get_greedy_action para algoritmo Q-Learning: Retorna a função greedy_action, aplicada na tabela action-value q e no estado em questão.

6) Função learn para algoritmo Q-learning: Foi feito de forma análoga ao descrito em Função learn para algortimo Sarsa, I-A4.

B. Escolha de Ação usando Rede Neural

Para realizar o aprendizado do robô seguidor de linha, utilizando as duas técnicas implementadas anteriormente (Sarsa e Q-Learning), foi executado o script main.py, alterando-se o valor da variável rl_algorithm, entre os construtores: Sarsa e QLearning, com seus respectivos parâmetros.

C. Reward Engineering

D. Treinamento usando DQN

E. Avaliação da Política

II. Resultados e Conclusões

A. Implementação dos algoritmos de RL

Conforme apresentado nas Figuras **??** e **??**, pode-se observar resultados equivalentes para ambas as técnicas. Assim, tanto a tabela de action-value possuiu valores similares, como as sequências de ações foram idênticas nas duas situações. Nota-se também que quanto mais perto do objetivo, maior o valor da action-value.

B. Escolha de Ação usando Rede Neural

Os resultados do aprendizado da política do robô seguidor de linha, utilizando os algoritmos de Sarsa e Q-Learning estão representados nas Figuras de **??** a **??** e **??** a **??**, respectivamente.

Comparando as tabelas de action-value para os dois algoritmos, em **??** e **??**, pode-se notar a quase equivalência entre os resultados. Nota-se também que quanto mais perto do objetivo,

```
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 24)                72
_________________________________________________________________
dense_2 (Dense)              (None, 24)                600
_________________________________________________________________
dense_3 (Dense)              (None, 3)                 75
=================================================================
Total params: 747
Trainable params: 747
Non-trainable params: 0
```

Figura 1. Sumário do modelo implementado em Keras.

.

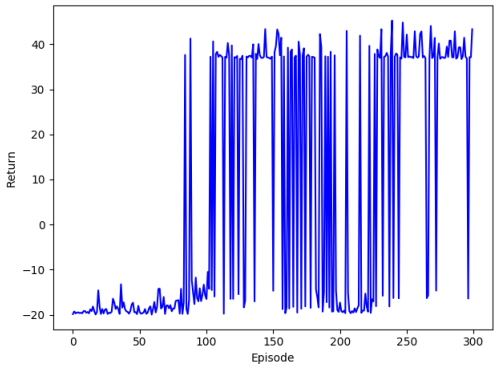

Figura 2. Gráfico gerado a partir do treinamento do modelo para 300 episódios.

.

maior o valor da action-value. Nas Figuras **??** e **??**, a tendência das duas também são semelhantes, embora já se consiga ver mais claramente algumas diferenças, nada que prejudique o processo de aprendizado.

Por fim, as Figuras **??** e **??** comprovam a convergência (até consideravelmente rápida) dos métodos, chegando aos resultados de caminho apresentados nas Figuras **??** e **??** para Sarsa e Q-Learning, respectivamente.

Esses resultados foram obtidos após 500 iterações no algoritmo Sarsa, e 556 no algoritmo Q-Learning, e demonstraram a correta implementação do código e funcionalidade para problemas de aprendizado por reforço.

C. Reward Engineering

D. Treinamento usando DQN

E. Avaliação da Política

REFERÊNCIAS

[1] M. Maximo, "Roteiro: Laboratório 12 - Deep Q-Learning". Instituto Tecnológico de Aeronáutica, Departamento de Computação. CT-213, 2019.

[2] M. Maximo, "Roteiro: Laboratório 8 - Imitation Learning com Keras". Instituto Tecnológico de Aeronáutica, Departamento de Computação. CT-213, 2019.

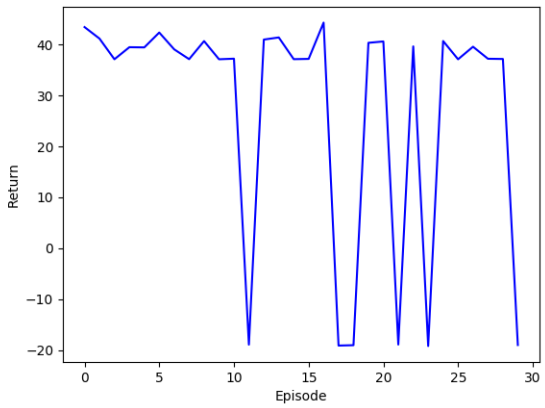

Figura 3. Representação em cores da tabela de action-value calculada, para algoritmo de Sarsa.

.

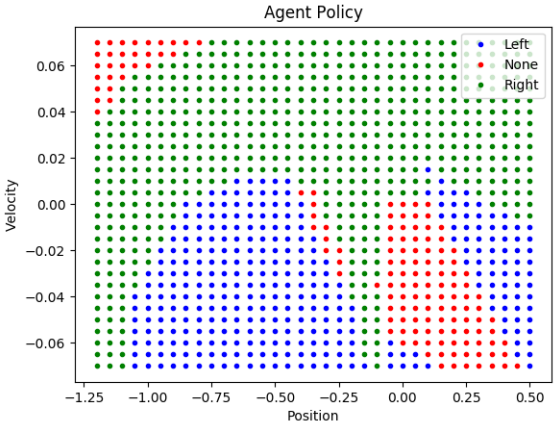

Figura 4. Representação em cores da tabela de greedy-policy calculada, para algoritmo de Sarsa.

.

```
episode: 1/30, time: 107, score: 43.4619, epsilon: 0.0
episode: 2/30, time: 94, score: 41.2016, epsilon: 0.0
episode: 3/30, time: 159, score: 37.1377, epsilon: 0.0
episode: 4/30, time: 85, score: 39.5076, epsilon: 0.0
episode: 5/30, time: 84, score: 39.4702, epsilon: 0.0
episode: 6/30, time: 101, score: 42.3907, epsilon: 0.0
episode: 7/30, time: 83, score: 39.0879, epsilon: 0.0
episode: 8/30, time: 160, score: 37.1565, epsilon: 0.0
episode: 9/30, time: 91, score: 40.7103, epsilon: 0.0
episode: 10/30, time: 159, score: 37.138, epsilon: 0.0
episode: 11/30, time: 167, score: 37.2522, epsilon: 0.0
episode: 12/30, time: 200, score: -18.9488, epsilon: 0.0
episode: 13/30, time: 92, score: 41.0069, epsilon: 0.0
episode: 14/30, time: 95, score: 41.4258, epsilon: 0.0
episode: 15/30, time: 160, score: 37.1562, epsilon: 0.0
episode: 16/30, time: 163, score: 37.2112, epsilon: 0.0
episode: 17/30, time: 113, score: 44.3414, epsilon: 0.0
episode: 18/30, time: 200, score: -19.131, epsilon: 0.0
episode: 19/30, time: 200, score: -19.0591, epsilon: 0.0
episode: 20/30, time: 89, score: 40.3713, epsilon: 0.0
episode: 21/30, time: 90, score: 40.6353, epsilon: 0.0
episode: 22/30, time: 200, score: -18.9305, epsilon: 0.0
episode: 23/30, time: 86, score: 39.6682, epsilon: 0.0
episode: 24/30, time: 200, score: -19.2182, epsilon: 0.0
episode: 25/30, time: 91, score: 40.7233, epsilon: 0.0
episode: 26/30, time: 160, score: 37.1333, epsilon: 0.0
episode: 27/30, time: 85, score: 39.6013, epsilon: 0.0
episode: 28/30, time: 165, score: 37.2369, epsilon: 0.0
episode: 29/30, time: 161, score: 37.1958, epsilon: 0.0
episode: 30/30, time: 200, score: -19.0234, epsilon: 0.0
Mean return:  27.79701181215042
```

Figura 5. Recompensa acumulada em função das iterações, para algoritmo de Sarsa.

.