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.

Fora isso, bastou-se descomentar as linhas de criação de uma pilha linear de camadas, as linhas compilação do modelo e impressão do summary do modelo, apresentado futuramente na seção II (Resultados e Conclusões), e a linha de retorno da função.

B. Escolha de Ação usando Rede Neural

Já essa etapa se tratou da implementação do método act() também da classe DQNAgent de dqn_agent.py. Nesse método, era escolhido e retornado uma ação de acordo com a política ϵ-greedy.

Essa implementação foi feita de forma bastante análoga à maneira do laboratório 12 [3]. Assim, 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 do retorno do índice do máximo elemento do array *model.predict(state)[0]*.

C. Reward Engineering

Nesse momento, foi implementado o método reward_engineering_mountain_car() de utils.py, script também fornecido no código base do laboratório. Nesse método, eram calculadas e retornadas as recompensas intermediárias "artificias", chamadas reward engineering, a fim de tornar o treino mais rápido no ambiente do Mountain Car.

Essa implementação foi feita conforme as equações apresentadas na seção 4.3 do roteiro do laboratório [1], ou seja, assim como apresentado no pseudo-código em Python a seguir.

```
reward = reward + (position - start) *
    (position - start) + velocity * velocity

aux = 0
if next_position >= 0.5:
   aux = 1

reward += 50 * aux
```

Os valores de position, start, velocity e next_position também eram fornecidos no roteiro [1], e bastava substituí-los no pseudo-código acima.

D. Treinamento usando DQN

Bastava treinar o modelo implementado, executando o script train_dqn.py, também do código base, e observar os resultados e os gráficos obtidos.

E. Avaliação da Política

Bastava aplicar o modelo implementado no ambiente do Mountain Car, executando o script evaluate_dqn.py, também do código base, e observar a simulação, os resultados e os gráficos obtidos.

```
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.

II. Resultados e Conclusões

O summary do modelo implementado em make_model() foi apresentado na Figura 1, e condiz com os requisitos pedidos na Tabela 3 do roteiro do laboratório [1].

Já a Figura 2 representa as recompensas acumulativas advindas do treinamento do modelo em 300 episódios. Esse resultado dependem diretamente da correta implementação e funcionamento dos métodos make_model() e act().

Pode-se dizer que esse gráfico condiz com o esperado, uma vez que é possível notar inicialmente recompensas pequenas para os primeiros episódios e, mais ou menos a partir do episódio 80, tornou-se frequente recompensas com valores elevados, chegando a valores próximos de 40, indicando um aprendizado significantemente correto.

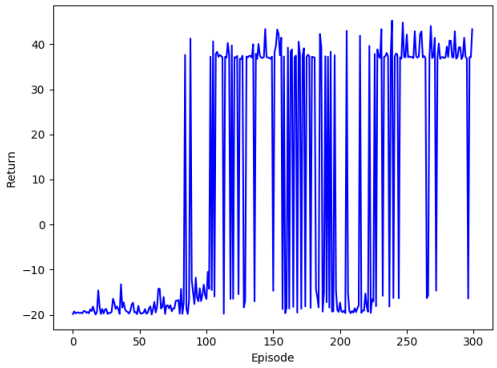

Figura 2. Recompensa acumulativa com o passar dos episódios, no treinamento do modelo para 300 episódios.

A. 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, 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.

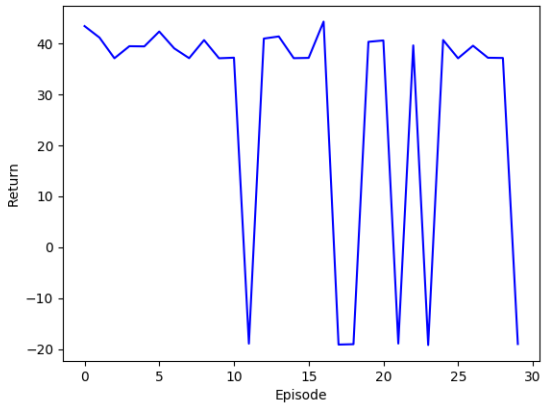

Figura 3. Representação em cores da tabela de action-value calculada, para algoritmo de Sarsa.

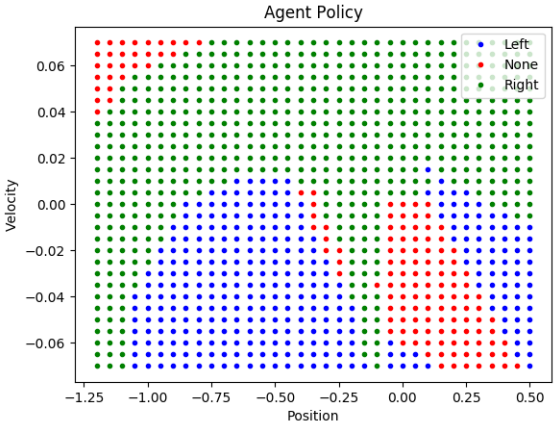

Figura 4. Representação em cores da tabela de greedy-policy calculada, para algoritmo de Sarsa.

B. Reward Engineering

C. Treinamento usando DQN

D. 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.

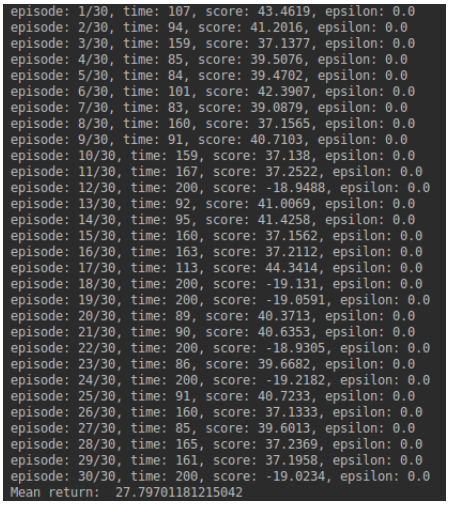

Figura 5. Recompensa acumulada em função das iterações, para algoritmo de Sarsa.

.

[3] M. Maximo, “Roteiro: Laboratório 12 - Aprendizado por Reforço Livre de Modelo”. Instituto Tecnológico de Aeronáutica, Departamento de Computação. CT-213, 2019.