Relatório do Lab 13 de CT-213

Trabalho 13 – Aprendizado por Reforço Profundo (*Deep Reinforcement Learning*)

Problema do *Mountain Car* com Deep Q-Learning / Deep Q-Networks (DQN)

Aluno:

Bruno Costa Alves Freire

Turma:

T 22.4

Professor:

Marcos Ricardo Omena de Albuquerque Máximo

Data:

15/06/2019

Instituto Tecnológico de Aeronáutica – ITA
Departamento de Computação

1. Implementação da rede neural DQN

Para resolver o problema do treinamento do *Mountain Car* por meio de Deep Q-Learning, a estratégia é modelar o problema como um MDP livre de modelo, para o qual iremos tentar estimar a função ação-valor, q(s, a), através de técnicas de aprendizado supervisionado profundo, *deep learning*. Daí o nome Q-Learning.

O MDP do *Mountain Car* consiste de um espaço de estados composto por posição e velocidade de um carrinho, ambos contínuos, e de um espaço de ações composto por 3 ações, mover pra direita, mover pra esquerda e ficar parado. O objetivo do carrinho é subir a montanha à sua direita e tocar a bandeira. Cada episódio tem duração de 200 *time steps*, e a recompensa, a priori, é de -1 por cada instante em que o carrinho não estiver no seu objetivo. Por se tratar de um MDP livre de modelo, a estratégia que será utilizada será a de estimar a função ação-valor por meio de aprendizado supervisionado, ou seja, utilizando uma rede neural.

Para isso, implementamos no *script* `dqn_agent.py` o agente DQN, o qual conta com um modelo de rede neural de 3 camadas densas, sendo as duas primeiras de 24 neurônios com ativação ReLU, e a camada de saída com uma quantidade de neurônios igual ao tamanho do espaço de ações do MDP, no caso, 3, e ativação linear. A entrada da rede é um *array* com o estado do agente, que no caso consiste de uma posição e uma velocidade. A função de custo utilizada é de erro quadrático médio, e o *backpropagation* utiliza o algoritmo de otimização Adam. Um resumo da implementação da rede foi salvo no arquivo `NN_summary.txt`, e é também reproduzido abaixo.

```
_________________________________________________________________
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
_________________________________________________________________
```

2. Implementação da política de ação

Assim como nos MDPs anteriores, o agente vai obter a política de ação a partir da função ação-valor estimada pela rede neural. Nesse caso, vamos implementar, no método `act` do agente DQN, uma estratégia ε-greedy sobre a estimativa $\hat{q}(s,a)$, fornecida pela rede neural dado o estado atual como entrada. Além disso, para acelerar o treinamento do agente, é feito um escalonamento exponencial de ε com um limitante inferior, segundo a equação:

$$\varepsilon_{new} = \max\{\varepsilon_{min}, \varepsilon_0 d^{e-1}\}$$

onde *d* é a taxa de decaimento, *e* é o episódio sendo rodado, e ε_{min} é um limite inferior para ε que visa garantir que ainda haja um comportamento exploratório conforme o treinamento avança.

3. Facilitando o treinamento do *Mountain Car*: *Reward Engineering*

O MDP do *Mountain Car* apresenta uma grande dificuldade para o aprendizado do agente, que reside no fato da única recompensa da tarefa estar muito distante de praticamente qualquer ação do agente. A recompensa só chega no final do episódio em que o agente obtém sucesso, e em todo outro caso, a recompensa é a mesma: -1 por cada instante em que o objetivo não tiver sido alcançado. Nesse modelo, todas as políticas executadas pelo agente que não atinjam o objetivo são igualmente penalizadas, de forma que o agente não recebe qualquer feedback positivo, ainda que esteja se aproximando do objetivo.

Por essa razão, um método de viabilizar o aprendizado do agente é criar recompensas intermediárias por objetivos artificiais, que, na concepção do engenheiro, ajudarão o agente do MDP a entender o caminho que deve seguir para aprender a tarefa. Esse método é chamado de *reward engineering*.

No caso, a implementação do reward engineering foi feita no arquivo `utils.py`, no método `reward_engineering_mountain_car`, e segue a equação:

$$r_{eng} = r_{orig} + (start - position)^2 + velocity^2 + 50 \cdot \mathbf{1}\{next\_position \geq goal\}$$

em que r_{eng} é a recompensa engendrada, r_{orig} é a recompensa original do MDP, *start* e *goal* são respectivamente as posições inicial e o objetivo do MDP, *position*, *velocity* e *next_position* são as componentes do estado do MDP, no instante presente e no seguinte.

Outra possibilidade seria recompensar o agente por aproveitar a força gravitacional para se impulsionar (quando estivesse caindo de um morro, por exemplo), com bônus quando fosse na direção do objetivo. Contudo, o esquema utilizado já foi suficiente para um aprendizado eficiente.

4. Treinando a DQN

Uma vez implementada toda a estrutura do agente, procedemos a realizar o treinamento da DQN, através do *script* `train_dqn.py`. O agente treinou durante 300 episódios, e os pesos da rede neural foram salvos no disco ao final do treinamento. Nas figuras de 1 a 3, temos gráficos do retorno (recompensa acumulada no MDP) obtido pelo agente ao longo de treinamento.

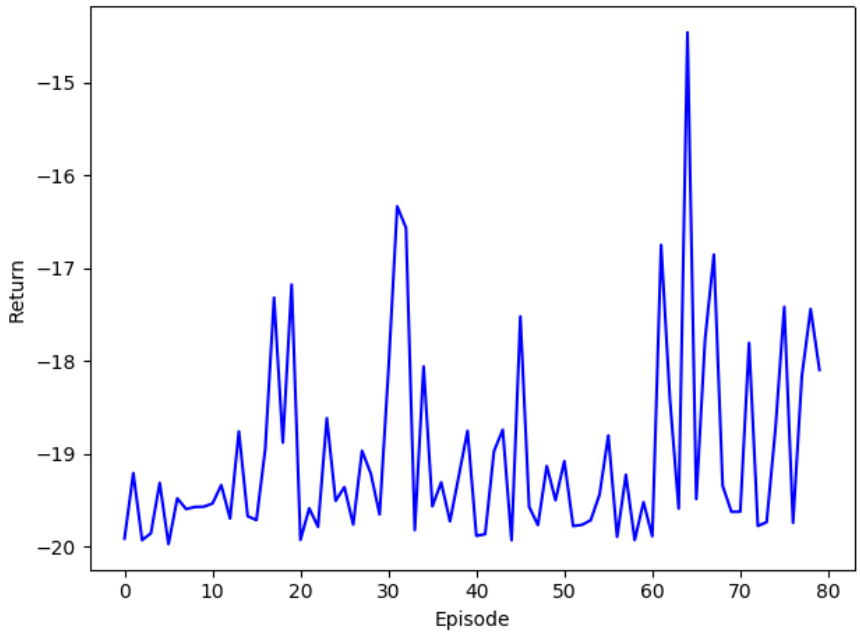

Figura 1: Gráfico do retorno do MDP ao longo dos 80 primeiros episódios de treinamento.

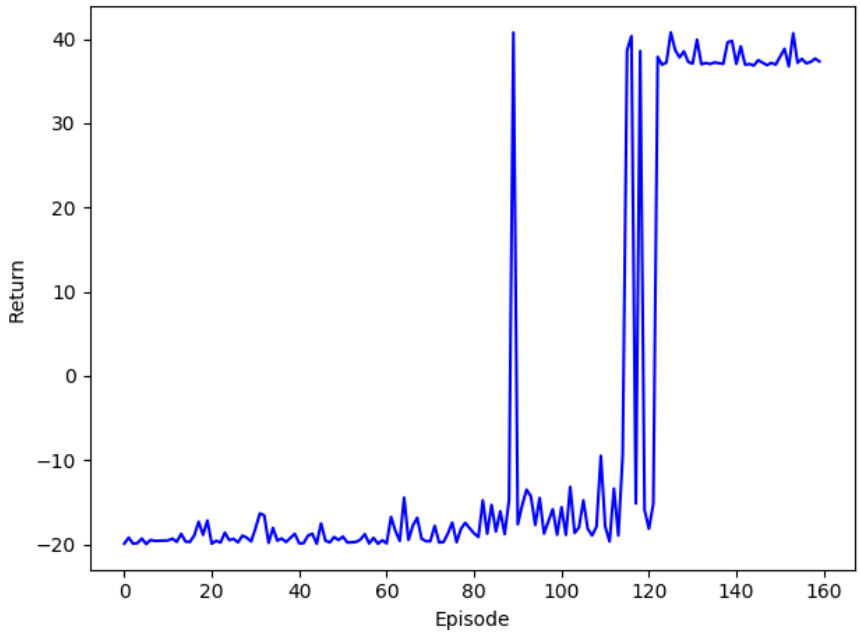

Figura 2: Gráfico do retorno do MDP ao longo dos 160 primeiros episódios de treinamento.

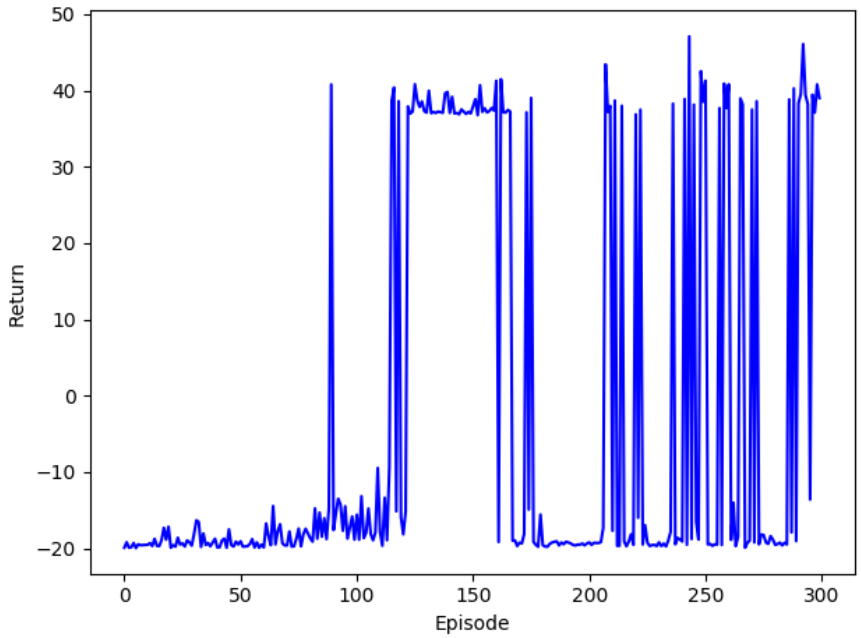

Figura 3: Gráfico do retorno do MDP durante todo o treinamento.

Podemos observar pelas figuras 1, 2 e 3 que o agente só conseguiu completar a tarefa pela primeira vez por volta do 90º episódio, o que a princípio foi um caso isolado. A partir de cerca de 116 episódios, o agente começou a completar a tarefa com sucesso várias vezes, e viveu uma fase de cerca de 40 episódios sem falhar no cumprimento do objetivo. Entre 160 e 210 episódios, uma fase ruim, e dali em diante, vários altos e baixos, com uma média relativamente boa de retornos.

Vale ressaltar que, apesar de estarmos utilizando escalonamento do ε, colocamos um limite inferior neste parâmetro para permitir ainda uma certa exploração por parte do agente, o que explicar por que o desempenho médio começou a cair mesmo após uma longa boa fase. No próprio gráfico da figura 3 podemos ver porque isso foi importante, dado que os melhores desempenhos ao longo de todo o treinamento só foram obtidos posteriormente à boa fase dos episódios 160-210.

Contudo, temos que levar em conta que a estratégia de *reward engineering* deturpa o propósito original do MDP, havendo a possibilidade de que políticas igualmente boas para a tarefa original sejam diferenciadas por uma métrica artificial. Eventualmente, pode ser que uma política se sobressaia às outras simplesmente por passar mais tempo na beira de atingir o objetivo, enquanto políticas que atingem o objetivo mais rapidamente pontuem menos, devido ao peso artificial dado para o distanciamento da posição inicial. Por essa razão, é preciso ter muito cuidado ao realizar a engenharia de recompensa, pois caso seja feita de maneira descuidada, o agente pode aprender vícios que o distanciam do verdadeiro aprendizado.

5. Avaliando a política aprendida

Uma vez treinada a rede neural que estima a função ação-valor, nosso agente pode ser avaliado pela escolha de política gulosa que ele faz. No *script* `evaluate_dqn.py`, os pesos da rede neural são carregados e o agente é configurado para agir de modo totalmente guloso com relação à saída da rede neural. Na figura 4 temos um gráfico da política aprendida pelo agente, mostrando para cada estado (par posição-velocidade) a ação gulosa tomada pelo agente.

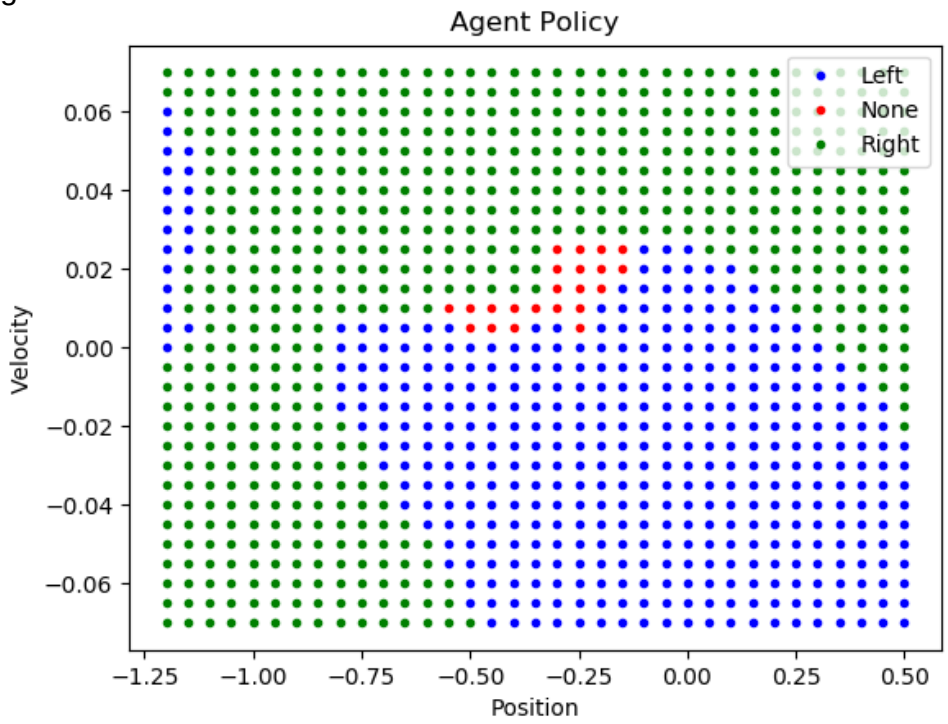

Figura 4: Política gulosa aprendida pelo agente DQN. Para cada par posição-velocidade, temos a ação correspondente escolhida.

Analisando a política mostrada na figura 4, podemos notar que, quando o agente está próximo da posição inicial, -0.50, ele aprendeu basicamente que deve se mover na direção de sua velocidade (positiva pra direita), de modo aproveitar o embalo. Nas posições mais à esquerda, ele aprendeu que quase sempre deve tentar se mover para a direita, seja para ganhar velocidade com a descida do morro, seja para se aproximar do objetivo. Curiosamente, em posições mais à direita, ele aprendeu que deveria se mover para a esquerda (descer o morro) caso sua velocidade não fosse positiva o bastante, de certa forma antecipando que não seria capaz de subir o morro naquela condição. Por fim, podemos observar que conforme ele esteja mais próximo do objetivo, o limiar de velocidade para que ele insista em ir para a direita é mais baixo, ou seja, quanto mais perto do objetivo, mais ele irá se esforçar para vencer a gravidade e alcançar a bandeira.

O processo de avaliação da política do agente contou também com um teste, para medir o retorno obtido pelo agente em 30 episódios com um início ligeiramente aleatório. Note que, apesar da política do agente nesse caso ser determinística (fizemos $\varepsilon = 0$ de modo a obter uma política totalmente gulosa), o ambiente de simulação configura o estado inicial do episódio com uma pequena aleatoriedade, de modo tenhamos, efetivamente, testes distintos que permitam explorar a potencialidade da política aprendida. Na figura 5 temos o gráfico do retorno nos 30 episódios de avaliação.

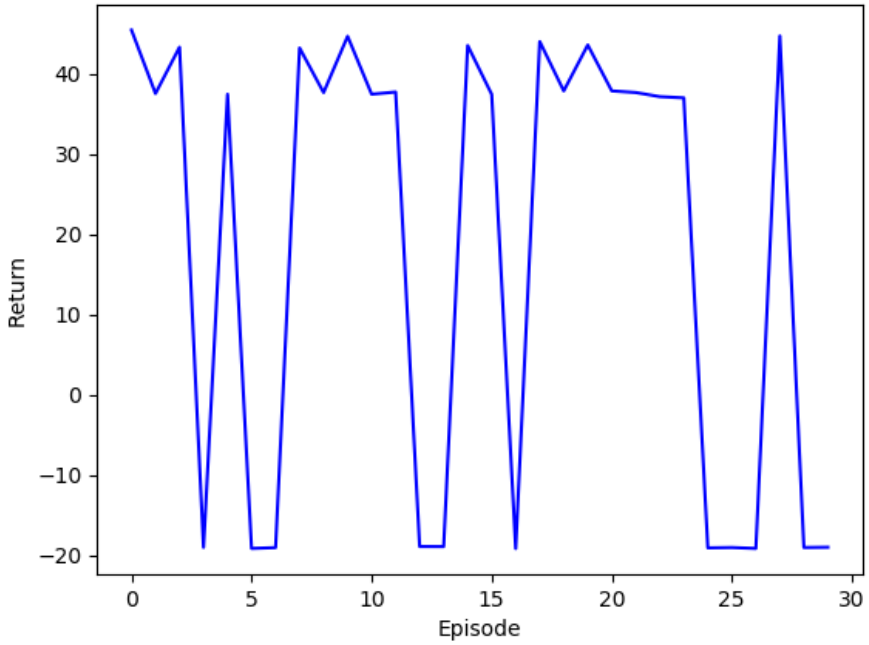

Figura 5: Gráfico do retorno obtido pelo agente nos episódios de avaliação.

Ao longo da avaliação, o retorno médio obtido pelo agente foi de 18.524. Podemos notar que a política aprendida para esse MDP não é tão robusta, visto que houve uma grande variância no retorno obtido para pequenas variações no estado inicial.

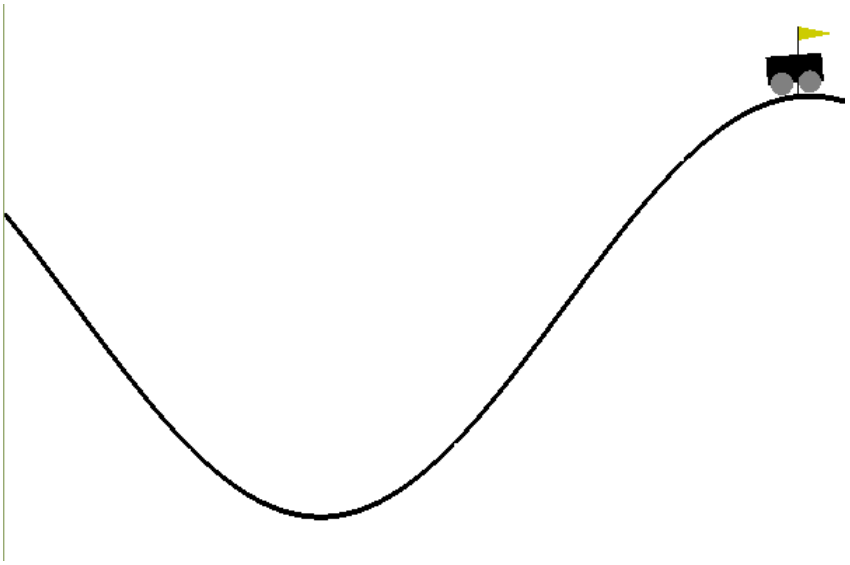

Figura 6: *Screenshot* do carrinho atingindo seu objetivo.