

Relatório do Laboratório 12: Aprendizado por Reforço Livre de Modelo

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 implementação de algoritmos de Aprendizado por Reforço (RL) Livre de Modelo, a saber Sarsa e Q-Learning, e utilizá-los para resolver o problema do robô seguidor de linha.

Index Terms—Aprendizado por reforço, Sarsa, Q-Learning

I. IMPLEMENTAÇÃO

A. Implementação dos algoritmos de RL

- 1) Função *epsilon_greedy_action*:
- 2) Função *greedy_action*:
- 3) Função *get_greedy_action* para algoritmo Sarsa:
- 4) Função *learn* para algoritmo Sarsa:
- 5) Função *get_greedy_action* para algoritmo Q-Learning:
- 6) Função *learn* para algoritmo Q-learning:

Essa parte do laboratório se tratava da implementação da função *policy_evaluation()*, presente no arquivo *dynamic_programming.py*, fornecido pelo código base do professor.

De maneira simples, essa função consiste em codificar a equação: $v_{k+1}(s) = \sum_{a \in A} \pi(a|s) r(s, a) + \gamma \sum_{a \in A} \sum_{s' \in S} \pi(a|s) p(s'|s, a) v_k(s')$, apresentada de forma bem semelhante no roteiro do laboratório [1].

Para implementá-la, os estados s se tornaram tuplas (i, j) , que foram iteradas por todo o grid world. Foram feitos loops também para iterar pelas ações a e pelos possíveis próximos estados s' , e os resultados da equação acima foram somados ao valor associado ao estado s em que se estava.

- 1) $\pi(a|s)$ era encontrado em *policy*;
- 2) $r(s, a)$ era encontrado em *grid_world.reward()*;
- 3) $p(s'|s, a)$ era encontrado em *grid_world.transition_probability()*;
- 4) $v_k(s')$ era a *policy* para um próximo estado s' , encontrado iterando-se sobre *grid_world.get_valid_sucessors()*.

Vale ressaltar que, após um número definido previamente de iterações, ou após a convergência dos valores de $v_{k+1}(s)$, o loop era interrompido.

B. Aprendizado da política do robô seguidor de linha

Já essa parte tratava-se da implementação da função *value_iteration()*, também presente no arquivo *dynamic_programming.py*, fornecido pelo código base do professor.

Análogo a função anterior, essa função consiste na codificação da equação: $v_{k+1}(s) = \max_{a \in A} (r(s, a) + \gamma \sum_{s' \in S} p(s'|s, a) v_k(s'))$, também presente no roteiro do laboratório [1].

A implementação também se tornou bastante semelhante à função de Avaliação de Política, com os estados s sendo tuplas (i, j) , que foram iteradas por todo o grid world, e loops para iterar pelas ações a e pelos possíveis próximos estados s' , dessa vez buscando os valores máximos dos resultados da equação, para serem considerados como valor associado ao estado s em que se estava nessa situação.

- 1) $r(s, a)$ estava em *grid_world.reward()*;
- 2) $p(s'|s, a)$ estava em *grid_world.transition_probability()*;
- 3) $v_k(s')$ era a *policy* para um próximo estado s' , encontrado iterando-se sobre *grid_world.get_valid_sucessors()*.

Vale ressaltar novamente que, após um número definido previamente de iterações, ou após a convergência dos valores de $v_{k+1}(s)$, o loop também era interrompido.

II. RESULTADOS E CONCLUSÕES

A. Primeiro Grid World

Foram gerados os resultados para os parâmetros de Grid World abaixo:

- 1) CORRECT_ACTION_PROB = 1.0
- 2) GAMMA = 1.0

Primeiro comparando-se os resultados apresentados nas Figuras 2 e 3, é possível notar que eles são idênticos, o que é esperado, uma vez que ambas as técnicas levam a convergência dos valores corretos de *policy* e *value*.

Sobre a Figura 1, a tendência observada é o *value* calculado ser maior em módulo para estados mais distantes do estado objetivo.

Nos três resultados é possível notar o *value* 0.0 para o estado objetivo, o que também condiz com o esperado.

B. Segundo Grid World

Foram gerados os resultados para os parâmetros de Grid World abaixo:

- 1) CORRECT_ACTION_PROB = 0.8
- 2) GAMMA = 0.98

Primeiro comparando-se os resultados apresentados nas Figuras 5 e 6, também é possível notar que eles são idênticos,

```

Value function:
[ -384.09, -382.73, -381.19, *, -339.93, -339.93]
[ -380.45, -377.91, -374.65, *, -334.92, -334.93]
[ -374.34, -368.82, -359.85, -344.88, -324.92, -324.93]
[ -368.76, -358.18, -346.03, *, -289.95, -309.94]
[ *, -344.12, -315.05, -250.02, -229.99, * ]
[ -359.12, -354.12, *, -200.01, -145.00, 0.00]
Policy:
[ SURDL, SURDL, SURDL, *, SURDL, SURDL ]
[ SURDL, SURDL, SURDL, *, SURDL, SURDL ]
[ SURDL, SURDL, SURDL, SURDL, SURDL, SURDL ]
[ SURDL, SURDL, SURDL, *, SURDL, SURDL ]
[ *, SURDL, SURDL, SURDL, SURDL, * ]
[ SURDL, SURDL, *, SURDL, SURDL, S ]

```

Figura 1. Resultado observado para o teste da *policy_evaluation()*, para a primeira opção de Grid World.

```

Value iteration:
Value function:
[ -10.00, -9.00, -8.00, *, -6.00, -7.00]
[ -9.00, -8.00, -7.00, *, -5.00, -6.00]
[ -8.00, -7.00, -6.00, -5.00, -4.00, -5.00]
[ -7.00, -6.00, -5.00, *, -3.00, -4.00]
[ *, -5.00, -4.00, -3.00, -2.00, * ]
[ -7.00, -6.00, *, -2.00, -1.00, 0.00]
Policy:
[ RD, RD, D, *, D, DL ]
[ RD, RD, D, *, D, DL ]
[ RD, RD, RD, R, D, DL ]
[ R, RD, D, *, D, L ]
[ *, R, R, RD, D, * ]
[ R, U, *, R, R, SURD ]

```

Figura 2. Resultado observado para o teste da *value_iteration()*, para a primeira opção de Grid World.

o que é novamente esperado, uma vez que ambas as técnicas levam a convergência dos valores corretos de *policy* e *value*.

Sobre a Figura 4, a mesma tendência que no primeiro Grid World é observada, ou seja, o *value* calculado é maior em módulo para estados mais distantes do estado objetivo.

Nos três resultados também é possível notar o *value* 0.0 para o estado objetivo, o que também condiz com o esperado.

Por fim, comparando-se as duas situações de Grid World, é possível notar que, com a adição do desconto *GAMMA*, e agora com a probabilidade de o agente executar uma ação diferente da escolhida para cada estado, tem-se que os *value* referentes a cada estado são maiores em módulo do que os calculados na primeira situação.

```

Policy iteration:
Value function:
[ -10.00, -9.00, -8.00, *, -6.00, -7.00]
[ -9.00, -8.00, -7.00, *, -5.00, -6.00]
[ -8.00, -7.00, -6.00, -5.00, -4.00, -5.00]
[ -7.00, -6.00, -5.00, *, -3.00, -4.00]
[ *, -5.00, -4.00, -3.00, -2.00, * ]
[ -7.00, -6.00, *, -2.00, -1.00, 0.00]
Policy:
[ RD, RD, D, *, D, DL ]
[ RD, RD, D, *, D, DL ]
[ RD, RD, RD, R, D, DL ]
[ R, RD, D, *, D, L ]
[ *, R, R, RD, D, * ]
[ R, U, *, R, R, SURD ]

```

Figura 3. Resultado observado para o teste da *policy_iteration()*, para a primeira opção de Grid World.

```

Value function:
[ -47.19, -47.11, -47.01, *, -45.13, -45.15]
[ -46.97, -46.81, -46.60, *, -44.58, -44.65]
[ -46.58, -46.21, -45.62, -44.79, -43.40, -43.63]
[ -46.20, -45.41, -44.42, *, -39.87, -42.17]
[ *, -44.31, -41.64, -35.28, -32.96, * ]
[ -45.73, -45.28, *, -29.68, -21.88, 0.00]
Policy:
[ SURDL, SURDL, SURDL, *, SURDL, SURDL ]
[ SURDL, SURDL, SURDL, *, SURDL, SURDL ]
[ SURDL, SURDL, SURDL, SURDL, SURDL, SURDL ]
[ SURDL, SURDL, SURDL, *, SURDL, SURDL ]
[ *, SURDL, SURDL, SURDL, SURDL, * ]
[ SURDL, SURDL, *, SURDL, SURDL, S ]

```

Figura 4. Resultado observado para o teste da *policy_evaluation()*, para a segunda opção de Grid World.

```

Value iteration:
Value function:
[ -11.65, -10.78, -9.86, *, -7.79, -8.53]
[ -10.72, -9.78, -8.78, *, -6.67, -7.52]
[ -9.72, -8.70, -7.59, -6.61, -5.44, -6.42]
[ -8.70, -7.58, -6.43, *, -4.09, -5.30]
[ *, -6.43, -5.17, -3.87, -2.76, * ]
[ -8.63, -7.58, *, -2.69, -1.40, 0.00]
Policy:
[ D, D, D, *, D, D ]
[ D, D, D, *, D, D ]
[ RD, D, D, R, D, D ]
[ R, RD, D, *, D, L ]
[ *, R, R, D, D, * ]
[ R, U, *, R, R, S ]

```

Figura 5. Resultado observado para o teste da *value_iteration()*, para a segunda opção de Grid World.

Isso se justifica e condiz com o esperado, uma vez que não se sabendo deterministicamente a ação tomada em cada estado, a função valor entende esse estado como "pior" quando comparado a situação na qual *CORRECT_ACTION_PROB* = 1. Além disso, o fator *GAMMA* adiciona mais imediatismo à recompensa das ações do agente, o que também diminui a medida de quão "bom" é determinado estado em comparação a situação no qual todas as recompensas até o objetivo são igualmente contabilizadas.

REFERÊNCIAS

- [1] M. Maximo, "Roteiro: Laboratório 11 - Programação Dinâmica". Instituto Tecnológico de Aeronáutica, Departamento de Computação. CT-213, 2019.

```

Policy iteration:
Value function:
[ -11.65, -10.78, -9.86, *, -7.79, -8.53]
[ -10.72, -9.78, -8.78, *, -6.67, -7.52]
[ -9.72, -8.70, -7.59, -6.61, -5.44, -6.42]
[ -8.70, -7.58, -6.43, *, -4.09, -5.30]
[ *, -6.43, -5.17, -3.87, -2.76, * ]
[ -8.63, -7.58, *, -2.69, -1.40, 0.00]
Policy:
[ D, D, D, *, D, D ]
[ D, D, D, *, D, D ]
[ RD, D, D, R, D, D ]
[ R, D, D, *, D, L ]
[ *, R, R, D, D, * ]
[ R, U, *, R, R, S ]

```

Figura 6. Resultado observado para o teste da *policy_iteration()*, para a segunda opção de Grid World.