Relatório do Laboratório 11: Programação Dinâmica

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 programação dinâmica no contexto de solução de um Processo Decisório de Markov (Markov Decision Process - MDP). Os algoritmos implementados serão avaliação de política (policy evaluation), iteração de política (policy iteration) e iteração de valor (value iteration), com o objetivo de avaliar políticas e determinar políticas ótimas para um grid world.**

***Index Terms*—MDP, policy evaluation, policy iteration, value iteration**

I. Implementação

A. Implementação de Avaliação de Política

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. Implementação de Iteração de Valor

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: $v_{k+1}(s) = max_{a \in A} \left(r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a)\, v_k(s') \right)$, 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.

C. Implementação de Iteração de Política

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

Assim com as anteriores, essa função também se tratava de um loop, que era interrompido em duas condições: ou ao chegar em um número máximo de iterações, ou após a convergência tanto da *policy*, quanto do *value*.

Dessa vez, o interior do loop se tratava da atualização dos valores de policy e value da seguinte maneira:

1) *value* era atualizado a partir da chamada da função *policy_evaluation()*, usando como parâmetros os valores até então de *policy* e *value*;
2) *policy* era atualizado a partir da chamada da função *greedy_policy()*, usando como parâmetros o valor até então de *value*, ou seja, calculado acima.

D. Comparação entre Grid Worlds diferentes

Para gerar os resultados a serem comparados para Grid Worlds diferentes, bastou alterar-se os valores *CORRECT_ACTION_PROB* e *GAMMA* entre os valores propostos no roteiro do laboratório [1] e executar o script *test_dynamic_programming.py* novamente.

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 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.

```
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.

```
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.

B. Algoritmo YOLO

Uma amostra das imagens geradas pelo algorito de detecção de objetos YOLO foram reproduzidos nas Figuras de **??** a **??**. Por essas imagens consegue-se observar o correto funcionamento do algoritmo, indicando que a implementação foi feita corretamente. Assim como esperado, então, conseguiu-se detectar a bola e os postes das traves nas imagens fornecidas pelo professor.

```
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. Detecção de uma bola e de um dos postes da trave através do algoritmo de YOLO.

```
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. Detecção de uma bola (mesmo que não completamente na imagem) através do algoritmo de YOLO.

```
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    ]
[    R    ,    D    ,    D    ,    R    ,    D    ,    D    ]
[    R    ,    D    ,    D    ,    *    ,    D    ,    L    ]
[    *    ,    R    ,    R    ,    D    ,    D    ,    *    ]
[    R    ,    U    ,    *    ,    R    ,    R    ,    S    ]
----------------------------------------------------------------
```

Figura 6. Detecção de um poste de trave através do algoritmo de YOLO.

Tendo em vista o que foi apresentado, pode-se notar, por fim, que a rede e o algoritmo YOLO realmente se demonstrou eficaz em realizar essa detecção de objetos em imagens.

REFERÊNCIAS

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