UNIVERSIDADE FEDERAL DE MINAS GERAIS

INTRODUÇÃO À INTELIGÊNCIA ARTIFICIAL

TP 2

Integrantes:

Arthur Siríaco Martins de Lacerda - 2020054218

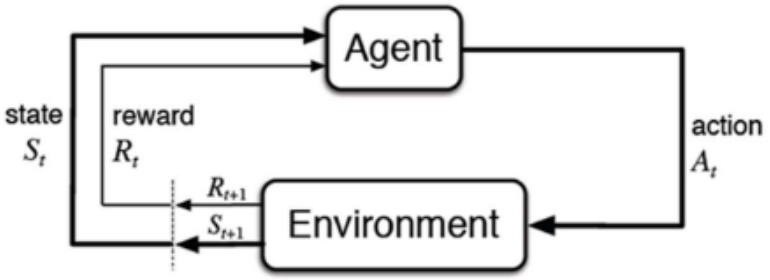
Trabalho Prático II - Reinforcement Learning

Documentação

Belo Horizonte
2022

INTRODUÇÃO

Conforme visto em aula, o Aprendizado por Reforço tem como característica o fato de os agentes aprenderem ao passo em que eles exploram e interagem com o ambiente. O aprendizado se dá por meio de uma função que busca maximizar um resultado, que é fruto de uma sequência de recompensas recolhidas pelo agente ao longo do percurso.



Nesse contexto, é bastante usual e eficiente a utilização do algoritmo de Q-Learning, que consiste na exploração e interação do ambiente por um agente que seleciona uma ação *a* em um determinado estado *s* e atualiza o valor de Q(s,a) à partir de uma recompensa *r* e um estado *s'*, conforme relação abaixo:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a' \in A} Q(s',a') - Q(s,a) \right]$$

Posto isso, este trabalho consiste na exploração, via Q-Learning, de um mapa discretizado de um cenário, em forma de matriz, no qual cada posição possui um símbolo que faz referência a um tipo de terreno. Dessa forma, cada símbolo tem atrelado a si uma recompensa, conforme tabela abaixo:

Terreno	Símbolo	Recompensa
Grama	.	-0.1
Grama Alta	;	-0.3
Água	+	-1.0
Fogo	x	-10.0
Objetivo	O	10.0
Parede	@	-∞

CLASSES E ESTRUTURAS DE DADOS

Para a correta modularização do código e melhor abstração de entidades do mundo real, o trabalho foi dividido em três arquivos, sendo um deles o “main.py” (onde está contido o corpo do programa e sua função principal). Os outros dois arquivos - “rl_models.py” e “map_handler.py” - tem a função de criar o modelo de Q-Learning e treiná-lo partindo dos parâmetros escolhidos e lidar com os mapas discretizados, respectivamente.

Main.py:
O arquivo em questão é responsável por absorver os parâmetros da execução de um arquivo “appconfig.json” que fica no diretório “source” do programa e efetuar um tratamento desses parâmetros, para posteriormente passá-los para o bloco principal do programa. Uma vez definidos tais parâmetros, instanciamos um modelo e aplicamos sobre ele o algoritmo de Q-Learning, para posteriormente treiná-lo e imprimir os resultados na tela.

Map:
A classe Map fica dentro do módulo “map_handler.py”, recebe um caminho para o arquivo de entrada, e é responsável por transferir o mapa fornecido na entrada para um grid montado a partir de uma lista de listas. Além disso, a classe possui métodos para retornar o valor de uma determinada posição e a altura ou a largura do grid. Finalmente, a classe Map sabe criar uma Q-Matrix, para utilização em um eventual algoritmo de Q-Learning, que possa a vir a ser aplicado sobre uma instância desse tipo.

QLearning:
A classe em questão fica dentro do arquivo “rl_models.py” e possui alguns blocos para seu correto funcionamento. Inicialmente, temos variáveis globais e dicionários que representam as ações, os estados, o efeito dos movimentos no grid, as posições perpendiculares à cada uma e os pesos representando a probabilidade de uma ação ser executada (para o caso de utilização em ambiente estocástico). Vale ressaltar que o construtor da classe recebe os estados possíveis, as ações possíveis e as recompensas disponíveis, sendo este último parâmetro passível de ser alterado, no caso de ambiente com recompensas positivas (informação a ser passada via arquivo de config). Além disso, temos outros parâmetros como a taxa de aprendizado (alpha), o fator de desconto para as recompensas (gamma), o caminho para o arquivo de entrada (map_path), o número de iterações, o valor de epsilon, as coordenadas iniciais e o modo de execução (que especifica se o ambiente será padrão, com recompensas positivas ou estocástico).

O funcionamento do algoritmo se dá por meio de um loop que itera um número de vezes pré-definido e itera novamente até achar um estado terminal selecionando ações e checando os estados correntes e futuros. Dessa forma, o algoritmo seleciona uma ação entre as disponíveis, de maneira randômica, e faz um teste que retorna um possível estado futuro. Por fim, o maior valor da Q_Matrix para um estado futuro é usado para atualizar o valor corrente de Q.

Algoritmo 1 Algoritmo para agente *Q-Learning*

```
1:  procedure Q-LEARNING(Mapa, Iterações)
2:      Q ← INICIALIZAMATRIZQ(Mapa)
3:      i ← 0
4:      while True do      ▷ Executa até que o número máximo de iterações seja atingido
5:          Estado ← SELECIONAESTADOINICIAL(Mapa)
6:          while Estado ≠ Terminal do
7:              Acao, NovoEstado ← SELECIONAAÇÃO(Q, Estado)
8:              MaxQ ← OBTEMMAIORQ(Q, NovoEstado)
9:              Q ← ATUALIZAQ(Q, Acao, Estado, MaxQ)
10:             Estado ← NovoEstado
11:             i ← i + 1
12:             if i = Iteracoes then
13:                 break
14:             end if
15:         end while
16:     end while
17: end procedure
```

ANÁLISE DAS POLÍTICAS

Para analisar corretamente as políticas, serão mostrados para cada mapa as políticas nas diferentes versões do problema. Dessa forma, temos três mapas de teste:

MAPA 1: “mapa_teste.map”

```
mapa_teste.map
+++.O
.@@.x
.@@..
.;;;.

standard
>>>>O
^@@^x
v@@^<
>>>^^

stochastic
>>>>O
^@@^x
^@@^<
^>>^^

positive_rewards
v<<<O
<@@<x
<@@<<
<<<<<
```

```
Parâmetros:
alfa: 0.1
gamma: 0.9
epsilon: 0.1
iterations: 100000
initial_x : 0
initial_y : 3
```

MAPA 2: “choices.map”

```
choices.map
@.........@
@+@.x@x.@;@
@+@.@@x.@;@
@+@.x@x.@;@
@+..@@x..;@
@+@.x@x.@;@
@+@.@@x.@;@
@+...O...;@

standard
@>>v<>>v<<@
@^@vx@xv@v@
@v@v@@xv@v@
@v@vx@xv@v@
@>>v@@xv<<@
@v@vx@xv@v@
@v@v@@xv@v@
@>>>>O<<<<@

stochastic
@>>v<<>v<<@
@^@vx@xv@v@
@v@v@@xv@v@
@v@vx@xv@v@
@>>v@@xv<<@
@v@vx@xv@v@
@v@v@@xv@v@
@>>>>O<<<<@

positive_rewards
@<<<<<<<<<@
@^@<x@x<@<@
@^@<@@x<@<@
@^@<x@x<@<@
@<<<@@x<<<@
@<@<x@x<@<@
@<@<@@x<@<@
@<<<<O<<<<@
```

```
Parâmetros:
alfa: 0.1
gamma: 0.9
epsilon: 0.1
iterations: 300000
initial_x : 5
initial_y : 0
```

MAPA 3: “maze.map”:

```
maze.map
x@x@x@x@x@.@
..........@
.x@@@@@@@@.@
.x@........@
.x@.@@@@@@@@
.x@........x
.x@@@@@@@@.@
O..........@

standard
x@x@x@x@x@v@
v<<<<<<<<<<@
vx@@@@@@@@^@
vx@v<>>>>>^@
vx@v@@@@@@@@
vx@>>>>>>>vx
vx@@@@@@@@v@
O<<<<<<<<<<@
```

```
stochastic
x@x@x@x@x@v@
v<<<<<<<<<<@
vx@@@@@@@@^@
vx@v<<>>>>^@
vx@v@@@@@@@@
vx@>>>>>>>vx
vx@@@@@@@@v@
O<<<<<<<<<<@

positive_rewards
x@x@x@x@x@<@
<<<<<v<<<<<@
<x@@@@@@@@<@
<x@<<<<<<<<@
<x@<@@@@@@@@
<x@<<<v<<<<x
<x@@@@@@@@<@
O<<<<<<<<<<@
```

```
Parâmetros:
alfa: 0.1
gamma: 0.9
epsilon: 0.1
iterations: 300000
initial_x : 10
initial_y : 0
```

De maneira geral, os ambientes padrão e estocástico apresentaram comportamentos parecidos para os mapas passados com seus respectivos parâmetros. Isso se dá pois, mesmo a ação não sendo escolhida com 100% de exatidão, com o passar das várias iterações, o agente vai conseguindo resultados “ótimos” próximos do padrão. Entretanto, com as recompensas positivas, observa-se uma absoluta descompensação no que diz respeito à trajetória do agente, pelo fato de agora termos uma discrepância absurda entre os estados terminais e os não terminais, e também não haver como chegar em um valor de ação negativo (que não seja infinitamente negativo).

Bibliotecas Utilizadas e Instruções para Execução

Este trabalho foi desenvolvido utilizando a linguagem Python na sua versão 3.9.12, portanto, recomenda-se fortemente a execução deste em ambiente com interpretadores que suportem esta versão ou superior.

As bibliotecas utilizadas foram:

- Math
- Json
- Random
- Numpy
- Operator

Para a execução correta deste programa via terminal, é **NECESSÁRIO** o preenchimento do arquivo “appsettings.json” presente no diretório raiz do código fonte. O arquivo se encontra nesses moldes:

```
{
 "HyperParameters": {
   "alpha": 0.1,
   "gamma": 0.9,
   "epsilon": 0.1
 },
 "RunMode": {
   "iterations": 300000,
   "run_mode": "positive_rewards",
   "initial_x": 10,
   "initial_y": 0
 },
 "FileNames": {
   "map_name": "maze.map"
 }
}
```

RUN MODE: Apenas os valores “standard”, “stochastic” e “positive_rewards” serão aceitos, do contrário o programa irá forçar o modo de execução para standard;
MAP NAME: Os mapas estão colocados no diretório “maps” contido dentro do diretório raiz (“source”) do programa. No arquivo de config, basta colocar o nome do arquivo que se deseja analisar.

Uma vez preenchido corretamente o arquivo de config, basta digitar o seguinte comando no terminal e a política será impressa no mesmo.

```
$ python main.py
```