

## Trabalho Prático 2: Q-Learning

Data de entrega: 03/07/2023

O objetivo deste trabalho é praticar um dos algoritmos de aprendizado de máquina vistos em sala. Mais especificamente, você deverá implementar o algoritmo de aprendizado por reforço chamado Q-Learning, como será detalhado a seguir.

Basicamente, você deverá fazer com que um agente consiga encontrar a saída em um ambiente com obstáculos, de forma similar ao exemplo visto em sala. O ambiente é um grid (matriz) quadrado de tamanho  $N$  onde, os quadrados podem ser livres ou conter obstáculos. Dois quadrados (terminais) possuem recompensas  $+1$  e  $-1$ , e os outros uma recompensa passada como parâmetro. O agente terá 4 possíveis movimentos (cima, baixo, esquerda, direita), podendo caminhar somente sobre os quadrados livres. Cada ação tem uma chance de 80% de funcionar corretamente, e 10% de chance de “escorregar” para um dos lados, de forma similar à estudada em sala de aula. Cada episódio termina quando o agente atinge um dos quadrados terminais.

Você deverá implementar o Q-Learning e testar o algoritmo em diferentes cenários e variando os diferentes parâmetros. Em especial, você deverá testar o seu algoritmo com e sem o uso do fator de exploração definido pelo  $\epsilon$ -greedy.

O seu código deverá receber como entrada um arquivo de texto, no qual estarão os parâmetros de entrada, no seguinte formato:

```
i a g r e
N
0 0 0 0 0
...
0 0 0 0 0
```

Em que, na primeira linha, separados por um espaço, **i** define o número de iterações que o algoritmo deve rodar, **a** representa o valor da taxa de aprendizado  $\alpha$ , **g** representa o fator de desconto  $\gamma$ , **r** define o valor da recompensa de todos os estados (quadrados) sem recompensa previamente definida e **e** é equivalente ao valor de  $\epsilon$ , caso o  $\epsilon$ -greedy seja utilizado. Já na segunda linha, nós temos **N**, que corresponde às dimensões da matriz quadrada. A partir da terceira linha, há a definição da matriz.

Um exemplo de entrada seria:

```
500 0.1 0.9 -0.06
5
0 -1 0 0 7
0 0 0 0 0
10 -1 0 0 0
0 0 0 0 4
```

Note que, nesse exemplo, não há  $\epsilon$ .

A fim de conseguir aplicar o algoritmo e visualizar seus resultados de forma fácil e padronizada, será feito o uso da biblioteca **seaborn**. Por isso, cada valor presente na matriz passada de entrada terá seu próprio significado, como definido na tabela a seguir.

| Valor | Significado                                |
|-------|--|
| 10    | Agente                                     |
| -1    | Obstáculo                                  |
| 0     | Chão com recompensa definida por parâmetro |
| 7     | Chão com recompensa +1                     |
| 4     | Chão com recompensa -1                     |

A partir da matriz apresentada de exemplo anteriormente e a plotando como um **heatmap** com a **seaborn**, obtemos a imagem:



Esse plot pode ser obtido com o seguinte código:

```
import seaborn as sns
sns.heatmap(data, square = True, cbar = False)
```

em que **data** pode ser uma lista ou array contendo a matriz de entrada.

Além disso, outro parâmetro de entrada para o seu código será o nome que os arquivos de saída deverão ter. Dessa forma, o seu código deverá ser executado com a seguinte linha:

```
python TP2.py arquivodeentrada.txt saidateste
```

Em que **arquivodeentrada.txt** é o nome/caminho do arquivo de entrada explicado anteriormente e **saidateste** é o que deverá ser incluído nos nomes dos arquivos de saída descritos a seguir.

Como saída, o seu código deverá produzir um **gif** e uma **imagem**.

A fim de produzir o **gif**, o código a seguir deverá ser utilizado:

```

import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib import animation

fig = plt.figure()

# NxN é o tamanho da matriz em questão
def init(N):
    sns.heatmap(np.zeros((N, N)), square = True, cbar = False)

def animate(i):
    data = data_list[i]
    sns.heatmap(data, square = True, cbar = False)

# Lista de arrays 10x10
data_list = []
# PREENCHA A LISTA

anim = animation.FuncAnimation(fig, animate, init_func = init(N), frames = len(data_list), repeat = False)

pillowwriter = animation.PillowWriter(fps = 7)

# filename é o nome do arquivo a ser salvo. É necessário que a extensão seja ".gif"
anim.save(filename, writer = pillowwriter)

```

Note que esse código **não** está completo. Você deverá implementar o que falta, somente. Seguindo o exemplo de entrada, o nome do gif produzido deveria ser **saidateste.gif**.

A fim de produzir a imagem, o código a seguir deverá ser utilizado:

```

import seaborn as sns
import matplotlib.pyplot as plt
sns.heatmap(data, cbar = True, square = True, annot = labels, fmt = '')
# imagemame é o nome do arquivo a ser salvo. É necessário que a extensão seja ".png"
plt.savefig(imagemame)

```

Em que **labels** é uma lista ou array de **strings**, com mesma dimensão de **data**. A imagem gerada deve apresentar a melhor ação em cada “casa” da matriz. Em **labels** teremos uma letra, podendo ser “c”, “b”, “e”, “d” ou “n”, indicando que a melhor ação daquela casa é ir para cima, baixo, esquerda, direita ou nenhuma, respectivamente, tal que “nenhuma” só é válida para “casas” em que o chão não tem uma recompensa definida por parâmetro. Além disso, os valores presentes em **data** devem corresponder aos valores de Q para aquele estado e ação. Vale ressaltar que o valor para as “casas” com obstáculos deverá ser 0, enquanto o valor para as “casas” com recompensa +1 ou -1 deverá ser o próprio valor da recompensa. Dessa forma, ainda seguindo o exemplo anterior, a imagem com as melhores ações deveria ser salva como **saidateste\_acoes.png**.

Como é esperado que todos os trabalhos sigam o mesmo padrão de saída, será necessário que a implementação seja feita em Python.

Sobre o formato da entrega, deverá ser feito um .zip ou similar com a seguinte estrutura:

- src
- saidas
- TP2.py
- documentacao.pdf

Em que **src** e **saidas** são pastas. Na pasta **src** deverão ser colocados os arquivos criados para a implementação, sem contar com o **TP2.py**, caso existam, caso contrário, deixe a pasta vazia. A pasta **saidas** deverá estar inicialmente vazia, mas é nela que o **gif** e a **imagem** devem ser salvos. **TP2.py** é o arquivo que será executado na linha de comando. **documentacao.pdf** é o arquivo contendo a documentação.

A documentação deverá ter, no máximo, **10 páginas**, contando com os seguintes tópicos:

- Descrição das estruturas de dados e algoritmo utilizado;
- Discussão dos resultados obtidos, analisando diferentes conjuntos de configurações para os parâmetros e os efeitos que essas mudanças causam. Utilize tabelas e/ou gráficos comparativos. Em especial, mostre pelo menos um gráfico da recompensa média obtida em função do número de iterações de treinamento.

**Bom trabalho!**