INSPER 1^o semestre de 2024

Avaliação Final

Reinforcement Learning Prof. Fabrício Barth

Nome:

- *Quaisquer hipóteses relevantes devem ser* **explicitamente formuladas**. *Faz parte da avaliação da prova a* **correta interpretação** *das questões. A* **clareza** *e a* **objetividade** *das respostas serão consideradas na avaliação.*

1. Considere um agente que atua em um ambiente onde o conjunto de estados é definido por $S = s_1, s_2, s_3, s_4, s_5$, o conjunto de ações por $A = a_1, a_2, a_3$ e o espaço de estados é definido pelo grafo abaixo:

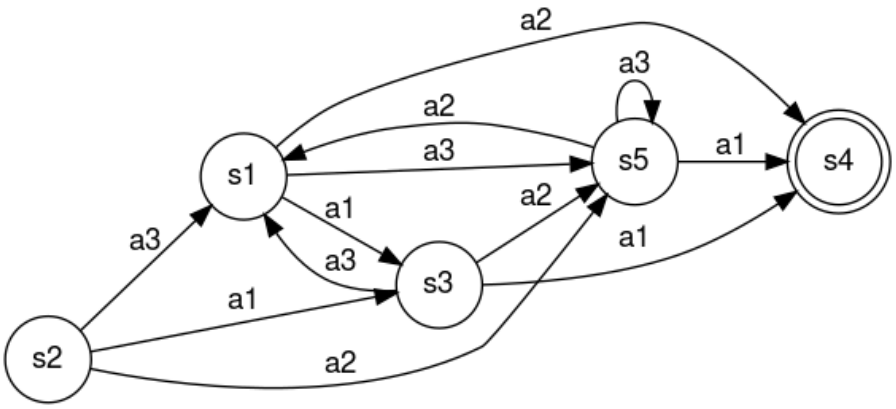


Figura 1: Espaço de estados do ambiente

Este mesmo agente foi treinado para atuar neste ambiente usando o algoritmo **Q-Learning**. Ao final do treinamento foi gerada a seguinte **Q-table**:

	a_1	a_2	a_3
s_1	-0.1	0.01	0.1
s_2	0.02	0.03	0.01
s_3	0.003	0.002	0.004
s_4	0	0	0
s_5	1	0	-1

Considere que o agente inicia no estado s_2 e que o estado terminal é o s_4, qual é a sequência de ações que o agente irá executar para chegar ao estado final?

2. No ambiente Cliff Walking a função de reward é definida como:

$$f(x) = \begin{cases} -1, & \text{para cada step} \\ -100, & \text{se o agente cair no penhasco.} \end{cases} \tag{1}$$

E o único estado terminal é quando o agente alcança o objetivo. Caso o agente caia no penhasco ele retorna para o estado inicial, mas o episódio não termina.

Existe diferença no comportamento de um agente treinado usando o algoritmo Q-Learning e um agente treinado usando SARSA neste ambiente? Justifique a sua resposta.

3. O ambiente **Simple Grid** é um ambiente onde o agente sabe executar 4 ações (cima, baixo, esquerda e direita) em um mapa pré-configurado com obstáculos (figura 2).

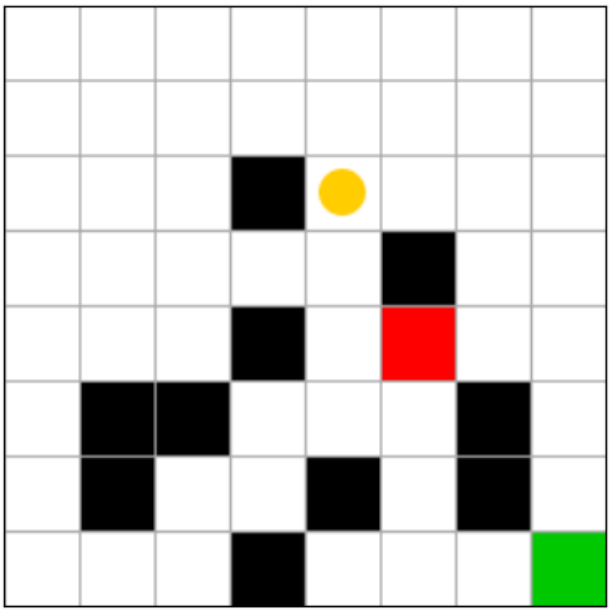


Figura 2: Exemplo de ambiente simple grid

O agente é treinado sempre no mesmo mapa com as mesmas dimensões e obstáculos. Os únicos dados que mudam são o estado inicial e final. O estado inicial é representado pelo quadrado vermelho e o estado final é representado pelo quadrado verde. A forma como os estados são tratados pelo agente também é muito simples. Por exemplo, para a figura 2 existem 64 estados possíveis porque o ambiente é um ambiente 8 por 8. Cada estado é representado por um número inteiro. Se o agente está na casa mais a esquerda e no topo do ambiente então o estado é representado pelo número 1. Na figura 2 o estado final é representado pelo número 64.

A função de **reward** é definida por:

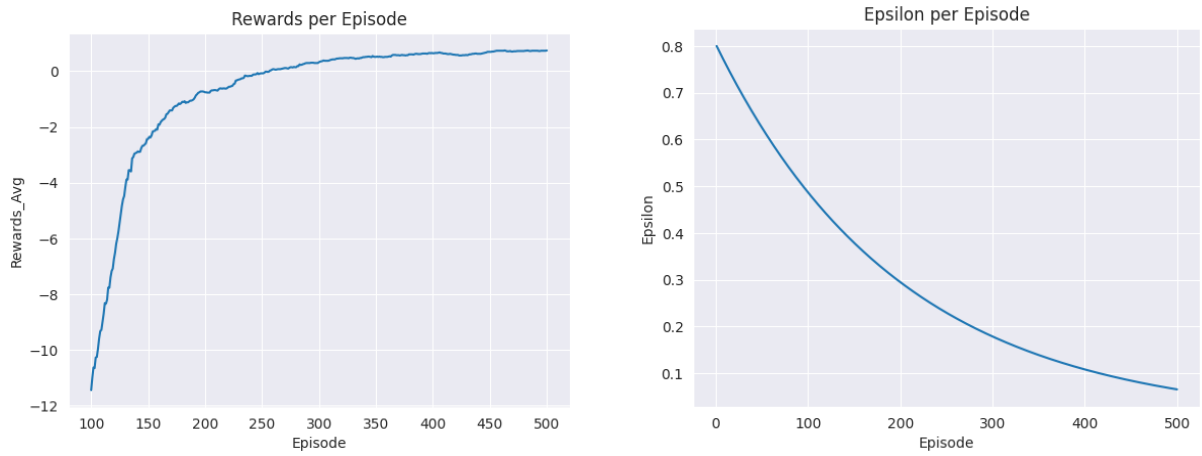
```python
def get_reward(self, x: int, y: int) -> float:
  """
  Get the reward of a given cell.
  """
  if not self.is_in_bounds(x, y):
    # if the agent tries to exit the grid, it receives a negative reward
    return -1.0
  elif not self.is_free(x, y):
    # if the agent tries to walk over a wall, it receives a negative reward
    return -1.0
  elif (x, y) == self.goal_xy:
    # if the agent reaches the goal, it receives a positive reward
    return 1.0
  else:
    # otherwise, it receives no reward
    return 0.0
```

Exemplo de código 1: Função de reward para o ambiente Simple Grid

Depois do agente treinado usando o seguinte algoritmo com os seguintes hiperparâmetros:

```python
qlearn = QLearning(
  env, alpha=0.1, gamma=0.999, epsilon=0.8,
  epsilon_min=0.05, epsilon_dec=0.999, episodes=5000)
```

E apresentando o seguinte comportamento ao longo do treinamento (figuras 3 e 3b).



(a) Curva de aprendizado

(b) Decaimento do ϵ

Figura 3: Figuras relacionadas com o aprendizado do agente no ambiente Simple Grid

Ao terminar o treinamento do agente e usá-lo no ambiente da seguinte forma:

```python
env = gym.make('SimpleGrid-8x8-v0', render_mode='human')
state, infor = env.reset()
done = False
rewards = 0
actions = 0

while not done and actions < 70:
  action = np.argmax(q_table[state])
  state, reward, done, truncated, info = env.step(action)

  rewards = rewards + reward
  actions = actions + 1

print("Actions taken: {}".format(actions))
print("Rewards: {}".format(rewards))
```

Exemplo de código 2: Uso do agente no ambiente Simple Grid

Ele tem o seguinte resultado:

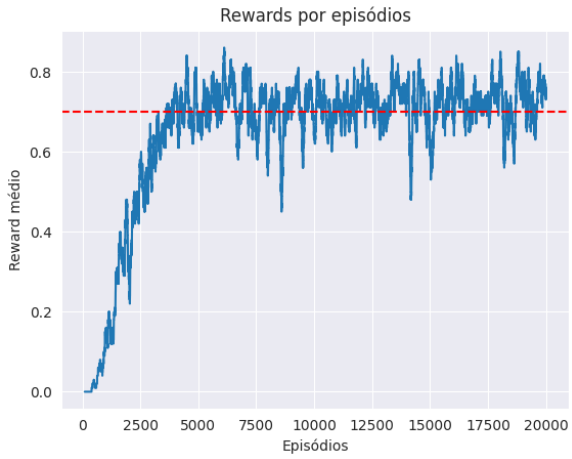
```
(venv) ?  simple_grid python TreinandoAgente.py
Actions taken: 70
Rewards: 0.0

(venv) ?  simple_grid python TreinandoAgente.py
Actions taken: 70
Rewards: 0.0

(venv) ?  simple_grid python TreinandoAgente.py
Actions taken: 70
Rewards: 0.0
```

Qual é a explicação?

4. O ambiente *Frozen Lake* é um exemplo de ambiente não determinístico que vimos nesta disciplina. Ao treinarmos um agente para atuar neste ambiente usando a seguinte configuração:

```
env = gym.make('FrozenLake-v1', render_mode='ansi').env
qlearn = QLearning(env, alpha=0.1, gamma=0.99, epsilon=0.8, epsilon_min=0.01, epsilon_dec
    =0.999, episodes=20_000)
```

temos uma curva de aprendizado como a apresentada na figura 4a, onde a linha pontilhada significa uma meta a ser atingida.



(a) Curva de aprendizado do agente para o ambiente Frozen Lake

(b) Segunda versão da curva de aprendizado do agente para o ambiente Frozen Lake

Figura 4: Figuras relacionadas com o aprendizado do agente no ambiente Frozen Lake

Além disso, depois de testar o agente treinado usando uma rotina com 100 testes que executam 100 episódios no ambiente, temos uma **média de** 78.56 **vezes que o agente consegue chegar ao destino, com um desvio padrão de** 4.73.

Ao mudarmos as configurações para:

```
env = gym.make('FrozenLake-v1', render_mode='ansi').env
qlearn = QLearning(env, alpha=0.1, gamma=0.99, epsilon=0.8, epsilon_min=0.01,
  epsilon_dec=1, episodes=20_000)
```

temos uma curva de aprendizado como a apresentada na figura 4b. Mas ao testar o agente treinado, usando o mesmo método descrito acima, temos **uma média de** 81.99 **vezes que o agente consegue chegar ao destino, com um desvio padrão de** 3.75.

Explique o que aconteceu.

5. Os algoritmos Deep Q-learning ou Deep Q-networks (DQN) são algoritmos que substituem a Q-table por uma rede neural. Talvez a arquitetura de rede neural mais utilizada e com certeza a mais simples é a *full-connected*, muitas vezes chamada de *multilayer perceptron* (MLP). Estas redes geralmente têm uma camada de entrada, uma ou duas camadas intermediárias e uma camada de saída. O que difere entre uma implementação e

outra é a quantidade de nodos de entrada, quantidade de nodos nas camadas intermediárias e quantidade de nodos na camada de saída. A quantidade de nodos nas camadas intermediárias é uma quantidade definida de forma empirica, mas que dependende da quantidade de nodos na primeira e última camadas. **O que define a quantidade de nodos na primeira camada da MLP e o que define a quantidade de nodos na última camada da MLP?**

6. Todos os algoritmos da família Deep Q-Learning tem o conceito de *experience replay*. O tamanho deste *experience replay* é definido por um hiperparâmetro. Na implementação do DQN na biblioteca *stable baseline* este hiperparâmetro se chama *replay buffer*. Na figura 5 é possível ver várias curvas de aprendizado para agentes atuando no ambiente *Lunar Lander*.

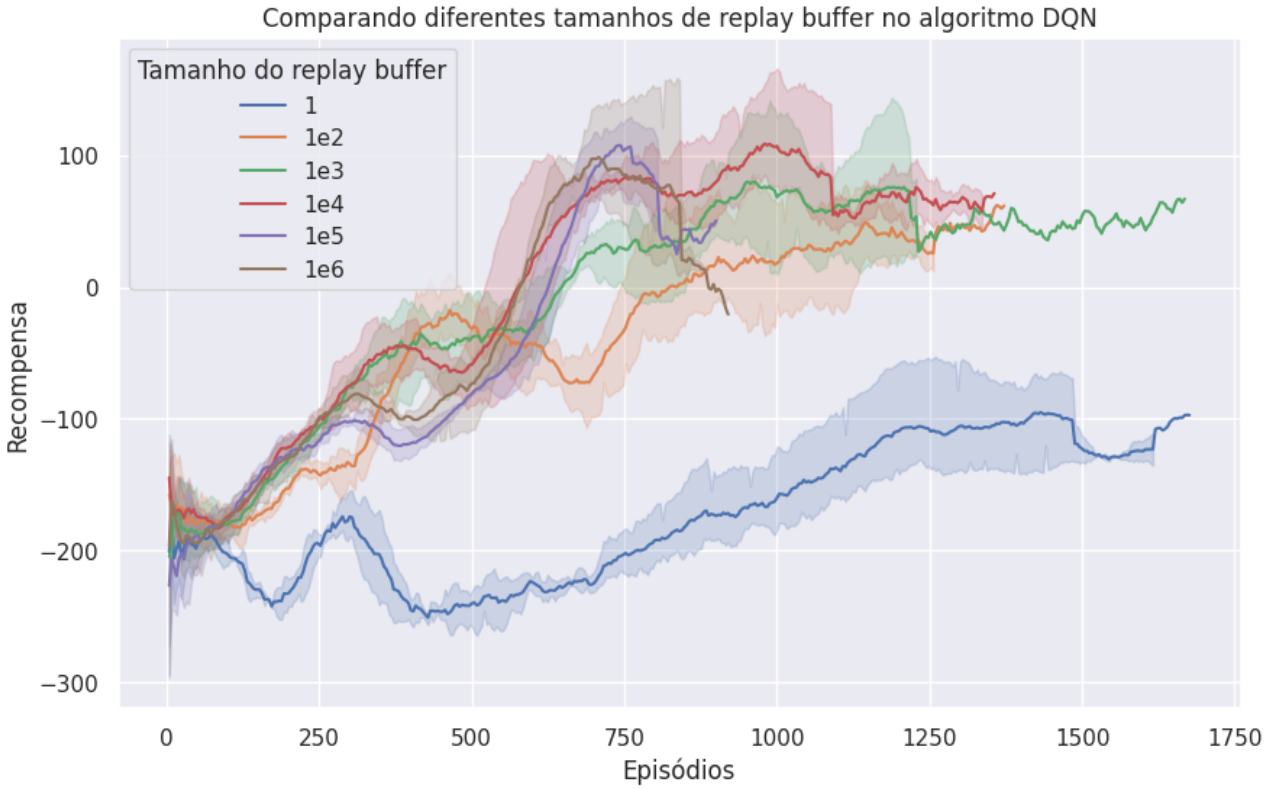


Figura 5: Análise do impacto do replay buffer no aprendizado do agente

Estes agentes foram treinados usando os hiperparâmetros padrão do DQN variando apenas o $buffer\_size$ $D = [1, 10^2, 10^3, 10^4, 10^5, 10^6]$. O objetivo desta análise é verificar o impacto do tamanho do *experience replay* no treinamento do agente.

A partir da análise do gráfico disponível em 5, quais são as conclusões que podemos chegar? Por que o desempenho do agente com $buffer\_size = 1$ é tão diferente dos demais?

7. O Algoritmo Double Deep Q-Learning, além de receber os hiperparâmetros normalmente utilizados no algoritmo Q-Learning (α, γ, ϵ, ϵ_{min}, ϵ_{dec}, quantidade de episódios), também tem como hiperparâmetros: tamanho da memória do *experience replay*, *batch size* e o tamanho do intervalo para atualização da *target network*.

   Considere uma implementação que faz uso da configuração de hiperparâmetros e da rede neural definidos no código 3 e no código 4, respectivamente.

```
gamma = 0.99
epsilon = 1.0
epsilon_min = 0.01
epsilon_dec = 0.99
episodes = 1000
batch_size = 64
learning_rate=0.001
memory = deque(maxlen=10000)
ma_steps = 1500
```

Exemplo de código 3: Lista de hiperparâmetros

```
model = Sequential()
model.add(Dense(512, activation=relu, input_dim=env.observation_space.shape[0]))
model.add(Dense(256, activation=relu))
model.add(Dense(env.action_space.n, activation=linear))
```

Exemplo de código 4: Denifição da Rede Neural

O algoritmo DQN está disponível na biblioteca **stable_baselines3**. O construtor deste algoritmo recebe os seguintes parâmetros:

```
class stable_baselines3.dqn.DQN(
  policy,
  env,
  learning_rate=0.0001,
  buffer_size=1000000,
  batch_size=32,
  tau=1.0,
  gamma=0.99,
  optimize_memory_usage=False,
  target_update_interval=10000,
```

```
  exploration_fraction=0.1,
  exploration_initial_eps=1.0,
  exploration_final_eps=0.05)
```

Exemplo de código 5: Parâmetros da classe DQN

onde o parâmetro *policy* pode receber três valores possíveis: *MlpPolicy*, *CnnPolicy* ou *MultiInputPolicy*.

Considerando o código em 3, 4 e 5, complete o código abaixo:

```
env = gym.make("LunarLander-v2")

model = DQN(
policy=_____________________,
env=env,
----------
"outros parâmetros que você julgar relevante"
---------
)
```

Utilize os hiperparâmetros definidos em 3 e a rede definida em 4 para definir uma nova instância de DQN. Escreva o código abaixo e justifique a sua tomada de decisão: