AI VIET NAM – AI COURSE 2023

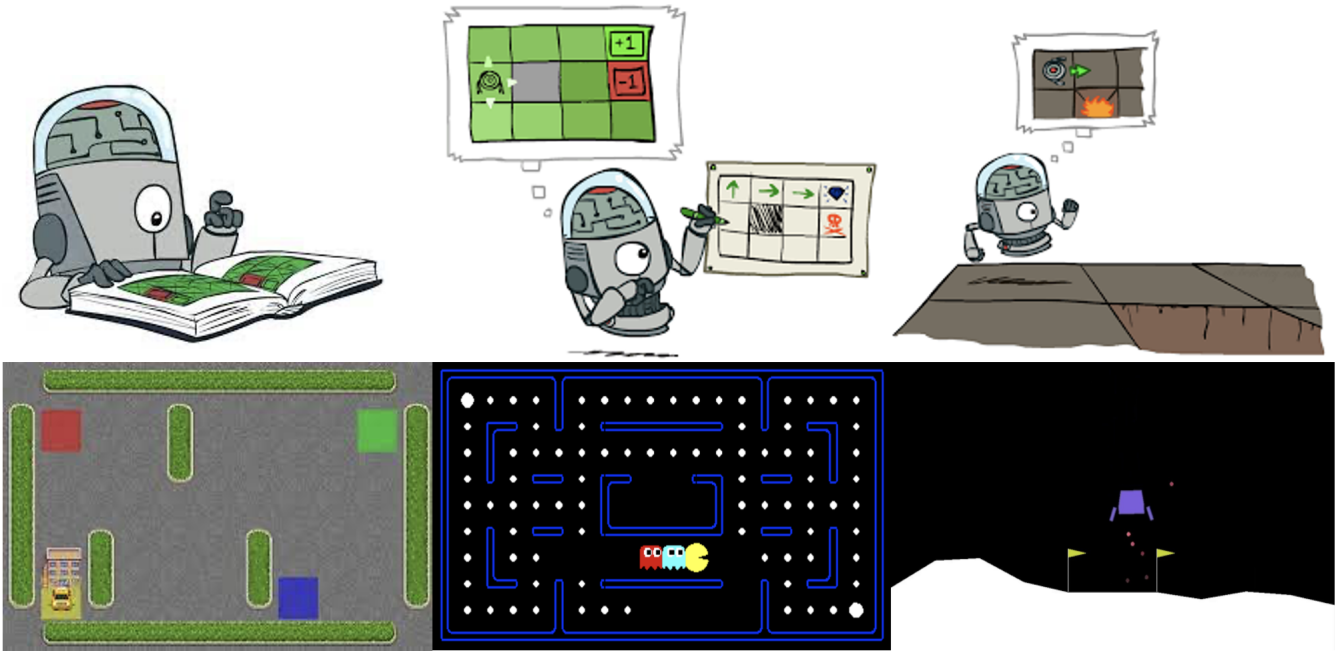
Reinforcement Learning - Exercise

Dinh-Thang Duong và Quang-Vinh Dinh
PR-Team: Hoàng-Nguyên Vũ, Đăng-Nhã Nguyễn và Minh-Châu Phạm

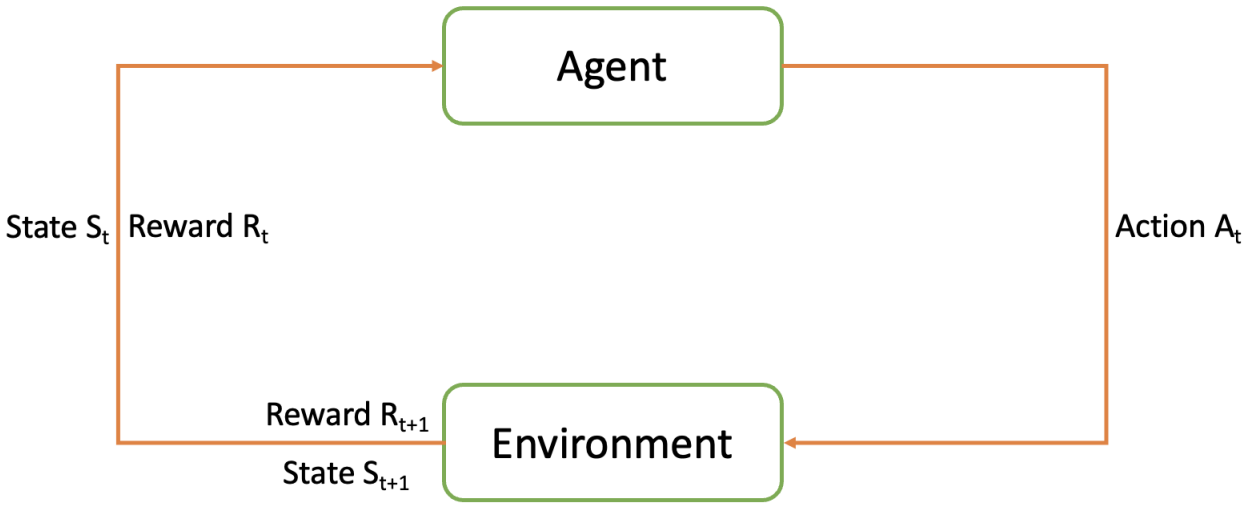
Ngày 18 tháng 4 năm 2024

Phần I: Giới thiệu

Reinforcement Learning (RL) (Tạm dịch: Học tăng cường), là một trong 3 nhánh chính trong Machine Learning (bên cạnh Supervised Learning và Unsupervised Learning). Đây là một nhánh học liên quan đến việc nghiên cứu cách để tạo ra một tác tử (Agents), có khả năng tương tác trong một môi trường giả lập (Environments) bằng một số các hành động (Actions) phù hợp, nhằm hoàn thành được mục tiêu của bài toán và tối đa điểm thưởng tích lũy kỳ vọng nhận được (Expected Cumulative Reward) qua mỗi hành động thực hiện. Reinforcement learning thường được thử nghiệm trong môi trường game giả lập, song vẫn góp mặt trong một vài các ứng dụng nổi tiếng như: AlphaGo, ChatGPT...



Để hiểu rõ hơn một số từ khóa sử dụng trong bài viết, chúng ta sẽ mô tả lại ý tưởng chính của một bài toán Học tăng cường qua framework sau:

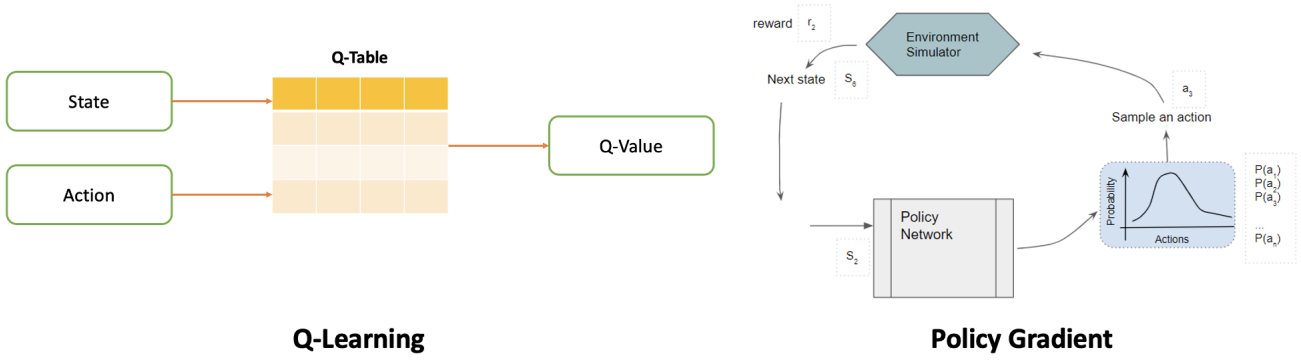


Hình 1: Reinforcement Learning Framework. Mô tả tổng quan về bài toán Học tăng cường.

Như đã nói ở trên, Reinforcement Learning liên quan đến việc triển khai một tác tử có khả năng tương tác với môi trường, từ đó học cách để tối ưu điểm thưởng tích lũy kỳ vọng nhận được qua mỗi nước đi. Với hình minh họa trên, ta có thể hình dung một ngữ cảnh khi quá trình huấn luyện bắt đầu (trò chơi bắt đầu) tại thời điểm $t = 0$ như sau:

1. Đầu tiên, tác tử nhận thông tin trạng thái S_0 (khởi tạo) từ môi trường.
2. Dựa vào thông tin trạng thái S_0, tác tử thực hiện hành động A_t lên môi trường.
3. Môi trường từ đó thay đổi sang trạng thái S_1.
4. Môi trường đồng thời trả về cho tác tử điểm thưởng R_1.

Và cứ như vậy, quá trình trên sẽ lặp đi lặp lại cho đến khi tác tử hoàn thành mục tiêu của trò chơi hoặc hết giờ (timeout), trò chơi sẽ kết thúc (gọi là terminate state).



Hình 2: Hai loại thuật toán Reinforcement Learning sẽ được tìm hiểu trong bài

Trong bài tập lần này, chúng ta sẽ cùng triển khai và chạy thử các thuật toán Reinforcement Learning cơ bản trên một số môi trường game, bao gồm các game trong thư viện gym của OpenAI (Taxi, LunarLander) và game PACMAN. Các thuật toán chúng ta sẽ triển khai bao gồm: Q-Learning, Approximate Q-Learning và Policy Gradient.

Phần II: Bài tập

Để có thể quan sát tác tử chơi game, các bạn cần chạy animation của game. Ở đây, chúng ta có thể hiển thị animation game trên cả 2 nền tảng code (máy local và Google Colab):

- **Đối với máy local:** Các bạn hãy sử dụng môi trường ảo conda (cách cài đặt conda các bạn hãy tham khảo tại đây) và cài đặt thư viện gym như sau:

```
$ pip3 install 'gym[all]'
```

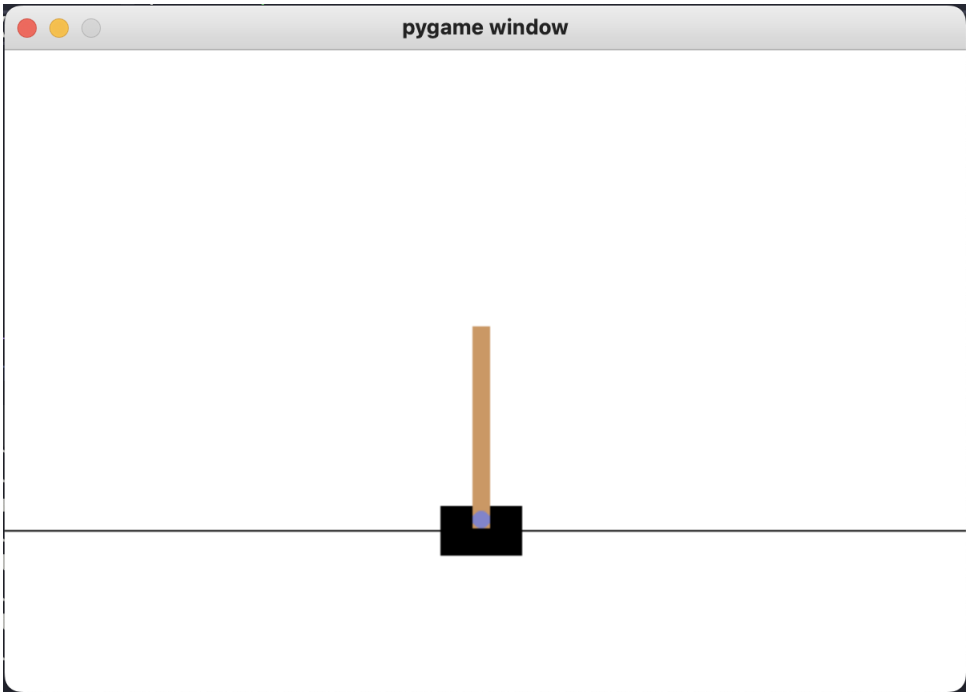
Sau khi quá trình tải hoàn tất, các bạn có thể test thư viện bằng cách chạy thử môi trường game **CartPole-v1** bằng cách copy đoạn lệnh này vào một file .py bất kì (giả sử **test.py**):

```
# test.py
import gym

env = gym.make('CartPole-v1', render_mode='human')
env.reset()
for _ in range(100):
    env.render()
    env.step(env.action_space.sample())
env.close()
```

Sau đó, thực thi file **test.py** trong terminal bằng lệnh:

```
$ python3 test.py
```

Nếu thực thi thành công, các bạn sẽ thấy một cửa sổ animation game CartPole hiện lên như sau:



Hình 3: Hình ảnh trực quan của môi trường CartPole-v1

- **Đối với Google Colab:** Mặc dù không thể chạy trực tiếp animation game như ở máy local, song vẫn có nhiều cách khác nhau giúp ta có thể quan sát animation game. Ở đây, chúng ta coi các

trạng thái của môi trường như các frame, sau đó lưu lại thành một file .gif, từ đó có thể dễ dàng hiển thị lên colab. Các bước thực hiện như sau:

1. **Cài đặt một số package cần thiết:**

```
!sudo apt-get update
!apt install imagemagick
!pip install 'gym[all]' pygame
```

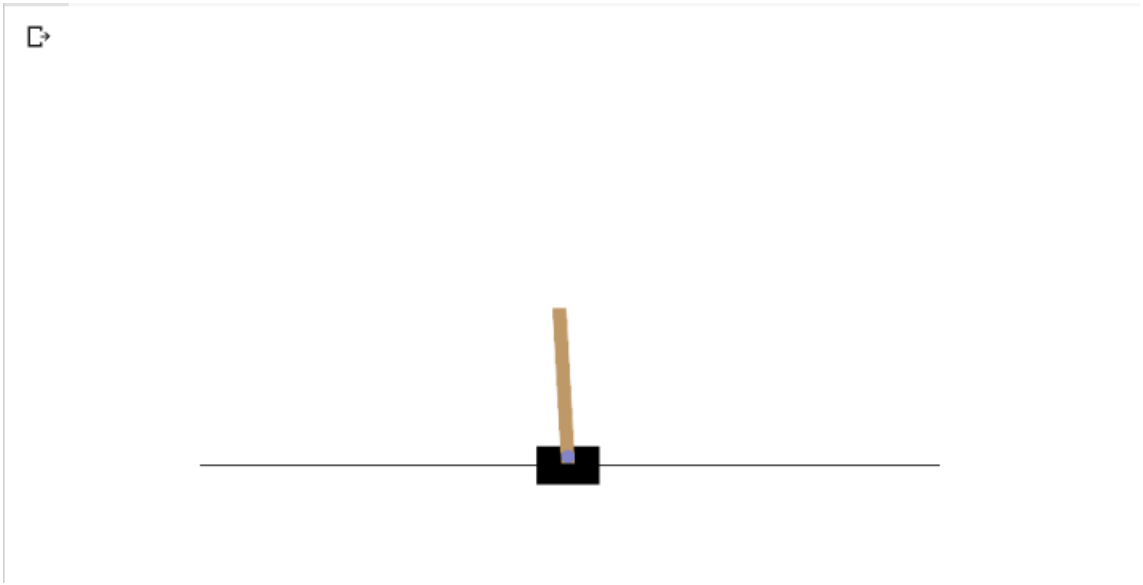
2. **Xây dựng hàm tạo ảnh gif:** Hàm này nhận vào danh sách các frame, sau đó xuất thành file có tên **demo.gif**:

```
import matplotlib.pyplot as plt
from matplotlib import animation

def save_frames_as_gif(
    frames,
    path='./',
    filename='demo.gif',
    fps=1
):
    temp_frame = frames[0]
    plt.figure(figsize=(temp_frame.shape[1] / 72.0, temp_frame.shape[0] /
    72.0), dpi=72)

    patch = plt.imshow(frames[0])
    plt.axis('off')

    def animate(i):
        patch.set_data(frames[i])

    anim = animation.FuncAnimation(plt.gcf(), animate, frames = len(frames),
    interval=50)
    anim.save(path + filename, writer='imagemagick', fps=fps)
    plt.close()
```

3. **Khởi tạo môi trường và kiểm thử:** Ta khởi tạo môi trường CartPole-v1 để kiểm thử hàm tạo ảnh gif trên bằng đoạn lệnh sau:

```
import gym

from IPython.display import Image

env_id = 'CartPole-v1'
env = gym.make(env_id, render_mode='rgb_array')
images = []
state, info = env.reset()
img = env.render()
images.append(img)

for _ in range(100):
    action = env.action_space.sample()
    state, reward, terminated, truncated, info = env.step(action)
    img = env.render()
    images.append(img)

    if terminated or truncated:
        break

env.close()
save_frames_as_gif(images, fps=10)
Image(open('demo.gif','rb').read())
```

Nếu toàn bộ các bước trên thành công, ta sẽ thấy animation game hiển thị trong Colab như hình sau:

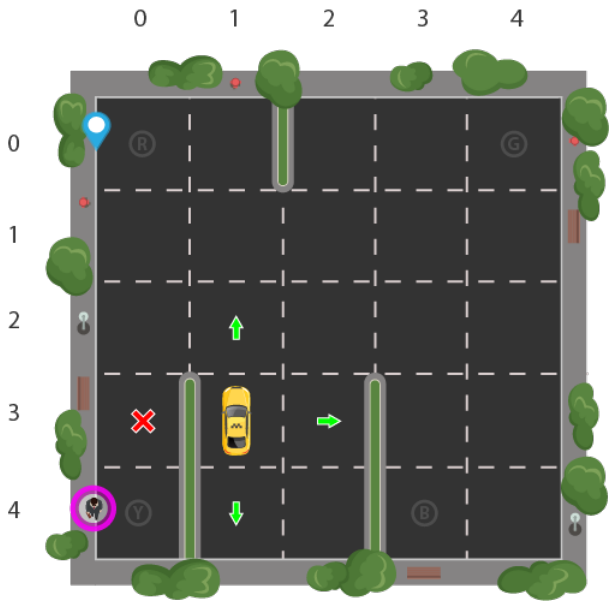


Hình 4: Hình ảnh trực quan của môi trường CartPole-v1 trong Google Colab

Để thuận tiện trong việc thực hiện bài tập này, bài tập 1 và 3 trong bài viết sẽ được thực hiện trong Google Colab.

Câu 1: Q-Learning

Cho môi trường game Taxi trong thư viện OpenAI Gym, với một số thông tin chính như sau (các bạn có thể đọc thêm thông tin chi tiết về môi trường này tại đây):



Hình 5: Hình ảnh trực quan của game Taxi

- **Kiểu môi trường:** Grid World (Deterministic Environment).
- **Không gian hành động (Action Space):** 6 (Discrete).
- **Không gian trạng thái (State Space):** 500 (Discrete).
- **Mục tiêu (Objective):** Di chuyển Taxi đến đón khách hàng và đưa họ đến một trong bốn vị trí trả khách.
- **Trạng thái kết thúc (Terminate State):** Taxi trả khách đúng nơi đã định.
- **Hàm điểm thưởng (Reward Function):**
  - **+20 điểm** nếu trả hàng khách đúng nơi.
  - **-1 điểm** mỗi một lần Taxi di chuyển.
  - **-10 điểm** nếu đón và trả hàng khách không đúng nơi đã định.

Các bạn hãy cài đặt thuật toán Q-Learning để huấn luyện tác tử hoàn thành nhiệm vụ, cũng như đạt điểm thưởng tích lũy tối đa trên môi trường Taxi này. Để triển khai thuật toán này, các bạn có thể tham khảo một số bước hướng dẫn sau:

1. **Khai báo các thư viện cần thiết:**

```
import numpy as np
import gymnasium as gym
import os
import tqdm
import matplotlib.pyplot as plt
```

```
6
7 from IPython.display import Image
8 from matplotlib import animation
9 from tqdm.notebook import tqdm
```

2. **Khởi tạo môi trường Taxi-v3:**

```
1 env_id = 'Taxi-v3'
2 env = gym.make(env_id, render_mode='rgb_array')
```

3. **Xây dựng hàm khởi tạo Q-Table:**

```
1 def init_q_table(state_space, action_space):
2     q_table = np.zeros((state_space, action_space))
3
4     return q_table
```

4. **Xây dựng hàm khởi tạo Greedy Policy:**

```
1 def greedy_policy(q_table, state):
2     action = np.argmax(q_table[state, :])
3
4     return action
```

5. **Xây dựng hàm khởi tạo Epsilon-greedy Policy:**

```
1 def epsilon_greedy_policy(q_table, state, epsilon):
2     rand_n = float(np.random.uniform(0, 1))
3
4     if rand_n > epsilon:
5         action = greedy_policy(q_table, state)
6     else:
7         action = np.random.choice(q_table.shape[1])
8
9     return action
```

6. **Khai báo một số siêu tham số cần thiết:**

```
1 n_training_episodes = 30000
2 n_eval_episodes = 100
3 lr = 0.7
4
5 max_steps = 99
6 gamma = 0.95
7 eval_seed = range(n_eval_episodes)
8
9 max_epsilon = 1.0
10 min_epsilon = 0.05
11 decay_rate = 0.0005
```

7. **Xây dựng hàm training:**

```
1 def train(
2     env,
3     max_steps,
4     q_table,
5     n_training_episodes,
6     min_epsilon,
7     max_epsilon,
8     decay_rate,
```

```
    lr,
    gamma
):
    for episode in tqdm(range(n_training_episodes)):
        epsilon = min_epsilon + (max_epsilon - min_epsilon) * np.exp(-decay_rate * episode)

        state, info = env.reset()
        step = 0
        terminated = False
        truncated = False

        for step in range(max_steps):
            action = epsilon_greedy_policy(q_table, state, epsilon)
            new_state, reward, terminated, truncated, info = env.step(action)

            q_table[state, action] = q_table[state, action] + lr * (reward + gamma * np.max(q_table[new_state]) - q_table[state, action])

            if terminated or truncated:
                break

            state = new_state

    return q_table
```

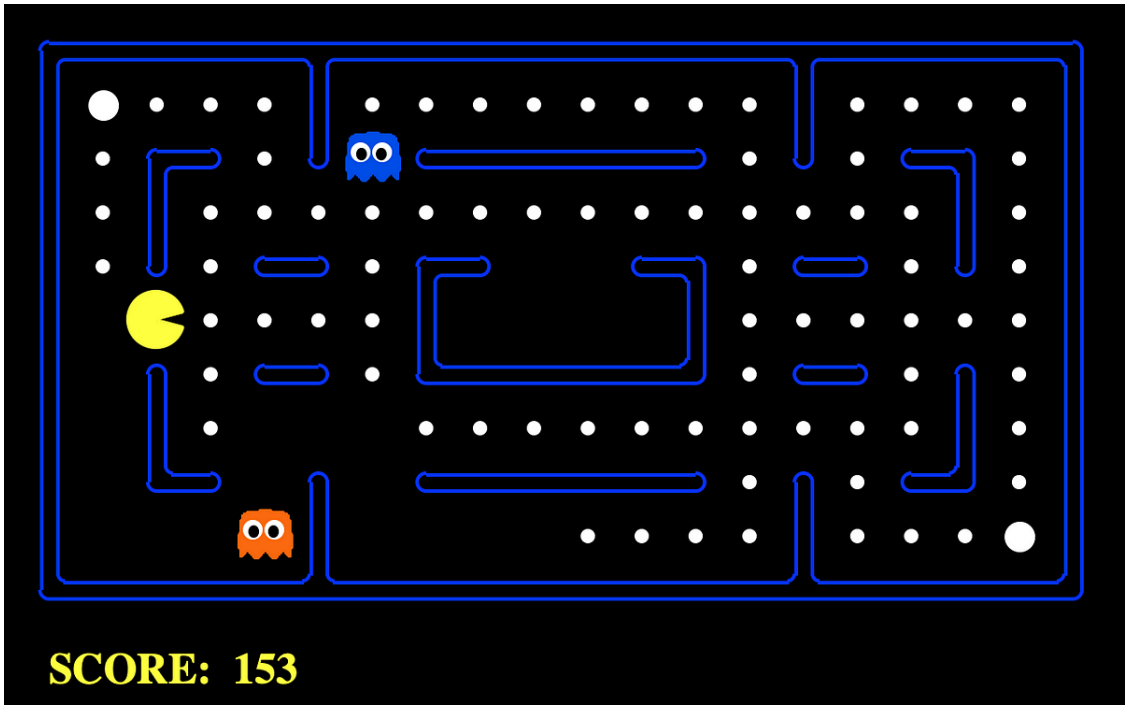
8. **Thực hiện huấn luyện:**

```
q_table = init_q_table(state_space, action_space)
trained_q_table = train(
    env,
    max_steps,
    q_table,
    n_training_episodes,
    min_epsilon,
    max_epsilon,
    decay_rate,
    lr,
    gamma
)
```

Sau khi quá trình training hoàn tất, ta sẽ có một Q-table với các giá trị Q-value đã được cập nhật.

Câu 2: Approximate Q-Learning

Cho môi trường game Pacman với một số thông tin chính như sau:



Hình 6: Hình ảnh trực quan của game Pacman

- **Kiểu môi trường:** Grid World (Stochastic Environment).
- **Không gian hành động (Actions Space):** 5 (Discrete).
- **Mục tiêu (Objective):** Di chuyển Pacman ăn toàn bộ tất cả các đồ ăn (chấm màu trắng) trên bản đồ đồng thời tránh né các ghosts.
- **Trạng thái kết thúc (Terminate State):** Pacman ăn được toàn bộ các đồ ăn trên bản đồ.
- **Hàm điểm thưởng (Reward Function):**
  - **+10 điểm** nếu ăn được 1 đồ ăn.
  - **+200 điểm** nếu tiêu diệt được 1 ghost.
  - **-500 điểm** nếu bị tiêu diệt bởi ghosts.
  - **-1 điểm** mỗi một lần thực hiện hành động bất kì.

Các bạn hãy tải source code của game Pacman theo đường dẫn này và cài đặt thuật toán Approximate Q-Learning để huấn luyện tác tử hoàn thành nhiệm vụ, cũng như đạt được điểm thưởng tích lũy tối đa trên môi trường game Pacman này. Bên cạnh đó, dựa vào chức năng recording game trong source code, các bạn hãy thực nghiệm với 4 bản demo tác tử chơi 1 ván game như sau:

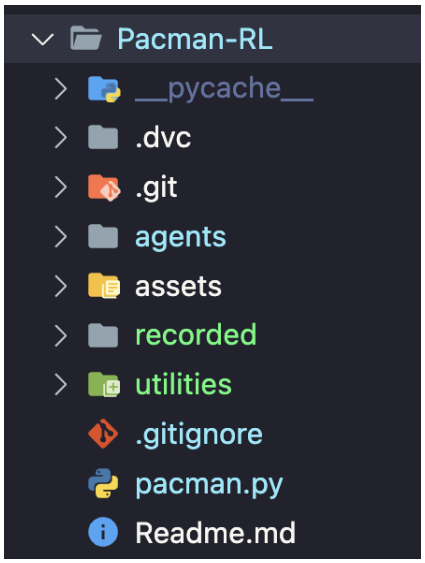
- **Demo 1:** Tác tử khi không được huấn luyện.
- **Demo 2:** Tác tử khi được huấn luyện qua 10 episodes.

- **Demo 3:** Tác tử khi được huấn luyện qua 100 episodes.
- **Demo 4:** Tác tử khi được huấn luyện qua 1000 episodes.

Để thực hiện được bài tập này, các bạn có thể tham khảo cách làm sau:

1. **Cài đặt source code game Pacman:** Dựa vào đường dẫn đã cung cấp ở trên, các bạn hãy tải source code game Pacman thông qua lệnh sau:

```
$ git clone https://dagshub.com/DavidN/Pacman-RL.git
```



Hình 7: Cấu trúc cây thư mục của mã nguồn game Pacman

Để nhanh chóng tiếp cận, các bạn cần lưu ý một vài file/thư mục chính như sau:

- **./pacman.py:** File main dùng để thực hiện chạy chương trình game Pacman cũng như huấn luyện tác tử. Các bạn có thể chạy thử game này bằng lệnh:

```
$ python3 pacman.py -n 1 -z 2
```

  Nếu thực thi thành công, cửa sổ game sẽ mở và các bạn có thể dùng nút lên xuống trái phải của bàn phím để chơi game.

- **./agents:** Thư mục chứa các class định nghĩa các kiểu tác tử, trong đó bao gồm Pacman và Ghosts. Đối với tác tử Pacman, hiện tại có 3 phiên bản gồm:
  - **KeyboardAgent:** Phiên bản cho phép ta điều khiển tác tử. Đây là mặc định của game.
  - **LeftTurnAgent:** Phiên bản Pacman sẽ luôn tự động chọn hướng rẽ trái để di chuyển.
  - **GreedyAgent:** Phiên bản tác tử sẽ luôn tự động chọn hành động mang lại điểm thưởng tức thời cao nhất.

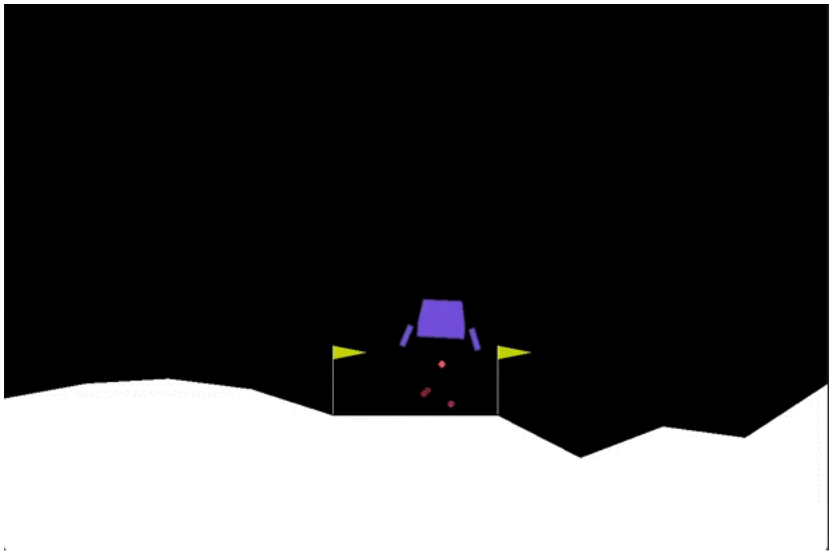
  Ví dụ, để sử dụng GreedyAgent, các bạn sử dụng lệnh sau:

```
$ python3 pacman.py -p GreedyAgent -n 1 -z 2
```

Có thể thấy rằng, để tạo một phiên bản tác tử mới, ta cần định nghĩa một class mới trong các file trong thư mục **./agents**. Để dễ dàng tiếp cận bài toán, các bạn hãy tham khảo file code Approximate Q-Learning trong repo này và tích hợp vào mã nguồn của đề bài.

Câu 3: Policy Gradient

Cho môi trường game LunarLander-v2 trong thư viện OpenAI Gym, với một số thông tin chính như sau (các bạn có thể đọc thêm thông tin chi tiết về môi trường này tại đây):



Hình 8: Hình ảnh trực quan của môi trường LunarLander

- **Kiểu môi trường:** Box2D (Stochastic Environment).
- **Không gian hành động (Actions Space):** 4 (Discrete).
- **State Shape:** (8,).
- **Mục tiêu (Objective):** Điều khiển tàu tên lửa đáp xuống đúng vị trí (càng sát càng tốt) đã định trên màn hình, trong khi giữ cho lượng nguyên liệu tiêu thụ ít nhất, đặc biệt không để tàu phát nổ.
- **Trạng thái kết thúc (Terminate State):** Tàu tên lửa đáp xuống thành công, hoặc phát nổ, hoặc bay khỏi giới hạn màn hình.
- **Hàm điểm thưởng (Reward Function):**
  - **+100 điểm** nếu đáp xuống bãi đáp thành công.
  - **-100 điểm** nếu tàu tên lửa phát nổ.
  - **-0.3 điểm** cho mỗi đơn vị thời gian trôi qua.
  - **+10 điểm** cho mỗi chân tàu nằm gọn bên trong bãi đáp.

Các bạn hãy triển khai thuật toán **REINFORCE** (một trong những thuật toán policy gradient) và huấn luyện tác tử có thể hoàn thành được trò chơi này, tối thiểu phải có điểm thưởng tích lũy tối đa là dương. Các bạn có thể tham khảo các bước thực hiện như sau:

Lưu ý: Để tăng tốc độ training, các bạn hãy **sử dụng GPU** cho bài tập này.

1. **Khai báo các thư viện cần thiết:**

```
import numpy as np
import gym
import os
import tqdm
import matplotlib.pyplot as plt

import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.distributions import Categorical

from collections import deque
from IPython.display import Image
from matplotlib import animation
from tqdm.notebook import tqdm
```

2. **Khởi tạo môi trường LunarLander-v2:**

```
env_id = 'LunarLander-v2'
env = gym.make(env_id, new_step_api=True)

state_space = env.observation_space.shape[0]
action_space = env.action_space.n
```

3. **Xây dựng Policy Network:**

```
class Policy(nn.Module):
    def __init__(self, s_size, a_size, h_size):
        super(Policy, self).__init__()
        self.fc1 = nn.Linear(s_size, h_size)
        self.fc2 = nn.Linear(h_size, h_size * 2)
        self.fc3 = nn.Linear(h_size * 2, a_size)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)

        return F.softmax(x, dim=1)

    def act(self, state):
        state = torch.from_numpy(state).float().unsqueeze(0).to(device)
        probs = self.forward(state).cpu()
        m = Categorical(probs)
        action = m.sample()

        return action.item(), m.log_prob(action)
```

4. **Xây dựng hàm training:**

```
def reinforce(
    policy,
    optimizer,
    n_training_episodes,
    max_steps,
    gamma,
    print_every
):
    scores_deque = deque(maxlen=100)
```

```
    scores = []

    for i_episode in range(1, n_training_episodes + 1):
        saved_log_probs = []
        rewards = []
        state = env.reset()

        for t in range(max_steps):
            action, log_prob = policy.act(state)
            saved_log_probs.append(log_prob)
            state, reward, done, _, info = env.step(action)
            rewards.append(reward)
            if done:
                break
        scores_deque.append(sum(rewards))
        scores.append(sum(rewards))

        returns = deque(maxlen=max_steps)
        n_steps = len(rewards)

        for t in range(n_steps)[::-1]:
            disc_return_t = returns[0] if len(returns) > 0 else 0
            returns.appendleft(gamma * disc_return_t + rewards[t])

        eps = np.finfo(np.float32).eps.item()

        returns = torch.tensor(returns)
        returns = (returns - returns.mean()) / (returns.std() + eps)

        policy_loss = []
        for log_prob, disc_return in zip(saved_log_probs, returns):
            policy_loss.append(-log_prob * disc_return)
        policy_loss = torch.cat(policy_loss).sum()

        optimizer.zero_grad()
        policy_loss.backward()
        optimizer.step()

        if i_episode % print_every == 0:
            print("Episode {}\tAverage Score: {:.2f}".format(i_episode, np.mean(
    scores_deque)))

    return scores
```

5. **Khai báo policy network và optimizer:**

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

policy = Policy(
    s_size=state_space,
    a_size=action_space,
    h_size=h_size,
).to(device)

optimizer = optim.Adam(policy.parameters(), lr=lr)
```

6. **Thực hiện huấn luyện:**

```
scores = reinforce(
    policy,
```

```
    optimizer,
    n_training_episodes,
    max_steps,
    gamma,
    print_every=100
)
```

Phần III: Câu hỏi trắc nghiệm

1. Trong Reinforcement Learning, mục tiêu của tác tử là?
   (*a*) Tối ưu điểm thưởng nhận được tại mỗi trạng thái.
   (*b*) Tối ưu điểm thưởng tích lũy kỳ vọng.
   (*c*) Tối ưu thời gian hoàn thành mục tiêu.
   (*d*) Tối ưu tiền thưởng nhận được trong trò chơi.

2. Theo Reinforcement Learning Framework, sau khi tác tử thực hiện hành động A_t, môi trường sẽ trả lại cho tác tử những gì?
   (*a*) Điểm thưởng R_t và Trạng thái S_t.
   (*b*) Trạng thái S_{t+1}.
   (*c*) Điểm thưởng R_{t+1} và Trạng thái S_{t+1}.
   (*d*) Điểm thưởng R_{t+1}.

3. Đáp án nào sau đây là một dạng bài toán trong Reinforcement Learning?
   (*a*) Adversarial
   (*b*) Recursive
   (*c*) Dynamic
   (*d*) Episodic

4. Trong Reinforcement Learning, trạng thái (S) được hiểu là?
   (*a*) Mô tả các hành động thực hiện được của tác tử.
   (*b*) Mô tả toàn cục của môi trường.
   (*c*) Mô tả cục bộ của môi trường.
   (*d*) Mô tả điểm thưởng nhận được của tác tử.

5. Trong trò chơi Super Mario Bros phiên bản OpenAI Gym (có không gian trò chơi như ảnh minh họa trên), nhân vật Mario có thể thực hiện được các hành động bao gồm: Di chuyển Trái/Phải, Ngồi xuống, Nhảy lên và Đứng yên. Theo đó, không gian hành động (Action Space) của môi trường này là?

(a) Discrete(4)

(b) Discrete(5)

(c) Discrete(6)

(d) Discrete(7)

6. Trong Reinforcement Learning, Policy là gì?

(a) Hàm ánh xạ hành động sang điểm thưởng.
(b) Hàm ánh xạ trạng thái sang hành động.
(c) Hàm ánh xạ điểm thưởng sang hành động.
(d) Hàm ánh xạ hành động sang trạng thái.

7. Trong phương pháp Value-based, việc luôn lựa chọn hành động cho điểm thưởng tích lũy cao nhất còn được gọi là gì?

(a) Softmax Policy.

(b) Random Policy.

(c) ε-greedy Policy.

(d) Greedy Policy.

8. Dòng code nào sau đây biểu diễn chiến lược cân bằng giữa exploration và exploitation?

(a) `np.argmax(Q[s])`

(b) `np.random.choice(A)`

(c) `np.argmin(Q[s])`

(d) `np.argmax(Q[s]) if np.random.rand() > e else np.random.choice(A)`

9. Mục đích của yếu tố Hệ số chiết khấu (Discounting Factor) là gì?

(a) Gia tăng kích thước của không gian trạng thái.

(b) Giảm kích thước của không gian trạng thái.

(c) Tăng điểm thưởng tích lũy kỳ vọng nhận được.

(d) Cân bằng độ quan trọng giữa điểm thưởng nhận được tức khắc và tương lai.

10. Trong Q-learning, hàm nào sau đây miêu tả tìm kiếm optimal policy?

(a) $\pi^*(s) = \arg\max_a Q^*(s, a)$

(b) $\pi^*(s) = \arg\max_a V^*(s)$

(c) $\pi^*(s) = \arg\min_a Q^*(s, a)$

(d) $\pi^*(s) = \arg\min_a V^*(s)$

11. Trong Q-Learning, công thức nào dưới đây biểu diễn hàm cập nhật Q-Value của trạng thái **s** và hành động **a**?

(a) $Q(s, a) = Q(s, a) + \alpha[r + \gamma \sum_{a'} Q(s', a') - Q(s, a)]$

(b) $Q(s, a) = Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$

(c) $Q(s, a) = Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$

(d) $Q(s, a) = Q(s, a) + \alpha[r - \gamma \max_{a'} Q(s', a') - Q(s, a)]$

12. Hàm nào sau đây không phải là hàm điểm thưởng tích lũy kỳ vọng?

(a) $V^\pi(s) = \mathbb{E}^\pi_{\tau_t \sim p(\tau_t)}[\sum_{i=0}^{\infty} \gamma^i \cdot r_{t+i} | s_t = s]$

(b) $V^{\pi}(s) = \sum_{a \in A} Q^{\pi}(s, a)$

(c) $V^{\pi}(s) = \sum_{a \in A} \pi(a|s) Q^{\pi}(s, a)$

(d) $V^{\pi}(s) = \mathbb{E}^{\pi}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... | S_t = s]$

13. Thuật toán nào sau đây thuộc nhóm thuật toán Policy Gradient?

(a) REINFORCE

(b) Q-Learning

(c) SARSA

(d) Value Iteration

14. Trong Policy Gradient, hàm nào dưới đây biểu diễn stochastic policy?

(a) $\pi_\theta(s) = \arg\max_a Q^{\pi}(s, a)$

(b) $\pi_\theta(a|s) = \arg\max_a Q^{\pi}(s, a)$

(c) $\pi_\theta(s) = \arg\max_a V^{\pi}(s)$

(d) $\pi_\theta(a|s) = P(a|s; \theta)$

15. Cho đoạn code Q-Learning sau:

```
state = env.reset()
for t in range(max_steps):
    action = np.argmax(Q[state])
    next_state, reward, done, _ = env.step(action)
    Q[state, action] = Q[state, action] + alpha * (reward + gamma * np.max(Q[
    next_state]) - Q[state, action])
    state = next_state
    if done:
        break
```

Loại policy dùng để lựa chọn hành động tại step **t** đã được triển khai là?

(a) ε-greedy Policy

(b) Softmax Policy

(c) Greedy Policy

(d) Random Policy

- Hết -