

TRƯỜNG ĐẠI HỌC SÀI GÒN
KHOA CÔNG NGHỆ THÔNG TIN



BÁO CÁO ĐỒ ÁN

Học phần: Seminar chuyên đề

**ĐỀ TÀI: TÌM HIỂU VỀ HỌC TĂNG CƯỜNG
(REINFORCEMENT LEARNING)**

Sinh viên thực hiện:

Trương Gia Thành - 3121410456

Nguyễn Minh Quang - 3120410425

Nguyễn Ngọc Tuấn Vũ - 3121410042

Nguyễn Khánh Hòa - 3121410026

Giảng viên hướng dẫn: TS.Nguyễn Quốc Huy

TP. Hồ Chí Minh, tháng 3 năm 2025

LỜI MỞ ĐẦU

Trước hết, nhóm chúng em xin bày tỏ lòng biết ơn sâu sắc đến Nguyễn Quốc Huy, giảng viên bộ môn Seminar Chuyên Đề tại Khoa Công nghệ Thông tin, trường Đại học Sài Gòn. Thầy không chỉ là người truyền đạt kiến thức mà còn là nguồn động lực lớn để nhóm chúng em vượt qua những thử thách trong quá trình thực hiện đồ án. Sự tận tâm, kiên nhẫn và những ý kiến đóng góp mang tính định hướng của thầy đã giúp nhóm em hiểu rõ hơn về vấn đề, đồng thời hoàn thiện đồ án một cách tốt nhất trong khả năng. Chúng em rất biết ơn vì sự hỗ trợ của thầy trong từng giai đoạn, từ việc xác định vấn đề, định hướng nghiên cứu, đến việc kiểm tra, chỉnh sửa nội dung và hình thức báo cáo.

Chúng em cũng xin gửi lời cảm ơn chân thành đến các thầy cô tại trường Đại học Sài Gòn, đặc biệt là các thầy cô trong Khoa Công nghệ Thông tin. Chính những bài giảng tận tâm của thầy cô, từ các môn đại cương giúp xây dựng nền tảng kiến thức cơ bản, đến các môn chuyên ngành trang bị kỹ năng chuyên sâu, đã mang đến cho chúng em một hành trang vững chắc. Kiến thức mà chúng em tích lũy được không chỉ dừng lại ở lý thuyết mà còn là nguồn cảm hứng để chúng em áp dụng vào thực tiễn, đặc biệt là trong đồ án này. Chúng em luôn trân trọng những nỗ lực không ngừng nghỉ của các thầy cô trong việc truyền đạt kiến thức, chia sẻ kinh nghiệm thực tế và khơi gợi niềm đam mê nghiên cứu trong lòng sinh viên.

Đồng thời, nhóm em cũng xin chân thành cảm ơn các bạn đồng hành, những người đã cùng nhóm chia sẻ những ý tưởng, hỗ trợ trong việc thu thập, xử lý dữ liệu và hoàn thiện báo cáo. Sự đoàn kết, tinh thần hợp tác và những ý kiến đóng góp từ các bạn đã giúp chúng em không ngừng tiến bộ.

Trong quá trình thực hiện đồ án, nhóm chúng em đã đối mặt với không ít khó khăn, từ việc lên kế hoạch, phân chia công việc, thu thập và xử lý dữ liệu, đến phân tích, trình bày và hoàn thiện báo cáo. Những khó khăn này không chỉ đến từ thời gian hạn hẹp hay lượng kiến thức cần áp dụng mà còn từ những kinh nghiệm thực tế chưa đủ chín chắn của nhóm. Tuy nhiên, nhờ sự hướng dẫn tận tình của thầy cô và sự hỗ trợ lẫn nhau trong nhóm, chúng em đã nỗ lực vượt qua và hoàn thành đồ án này.

Dù vậy, chúng em ý thức rằng đồ án của mình vẫn còn nhiều hạn chế và chưa thể đạt đến mức hoàn hảo. Do đó, chúng em rất mong nhận được những ý kiến đóng góp, nhận xét quý báu từ thầy cô và bạn bè để không ngừng học hỏi, hoàn thiện hơn trong tương lai. Những góp ý này không chỉ giúp cải thiện chất lượng đồ án mà còn là cơ hội để chúng em rèn luyện và phát triển bản thân, chuẩn bị tốt hơn cho những thử thách trong sự nghiệp sau này.

Một lần nữa, chúng em xin gửi lời cảm ơn chân thành nhất đến tất cả những ai đã đóng góp vào quá trình thực hiện đồ án này. Nhờ vào sự hướng dẫn, động viên và đồng hành của mọi người, chúng em đã có thể đi đến ngày hôm nay với một sản phẩm thể hiện sự cố gắng và tâm huyết của cả nhóm.

Xin kính chúc thầy cô luôn dồi dào sức khỏe, tràn đầy năng lượng và đạt được nhiều thành công hơn nữa trong sự nghiệp.

DANH MỤC VIẾT TẮT

Từ viết tắt	Ý nghĩa
RL	Reinforcement Learning
Q-Learning	Quality Learning
Deep Q-Learning	Deep Quality Learning
MCTS	Monte Carlo Tree Search
LIME	Local Interpretable Model-agnostic Explanations
SHAP	SHapley Additive exPlanations
VIPER	Verifiable Inference via Policy Extraction for Reinforcement Learning

BẢNG PHÂN CHIA CÔNG VIỆC

STT	MSSV	Tên thành viên	% Công việc	Tiến độ
1	3121410456	Trương Gia Thành	25%	Hoàn thành
2	3120410425	Nguyễn Minh Quang	25%	Hoàn thành
3	3121410042	Nguyễn Ngọc Tuấn Vũ	25%	Hoàn thành
4	3121410026	Nguyễn Khánh Hòa	25%	Hoàn thành

MỤC LỤC

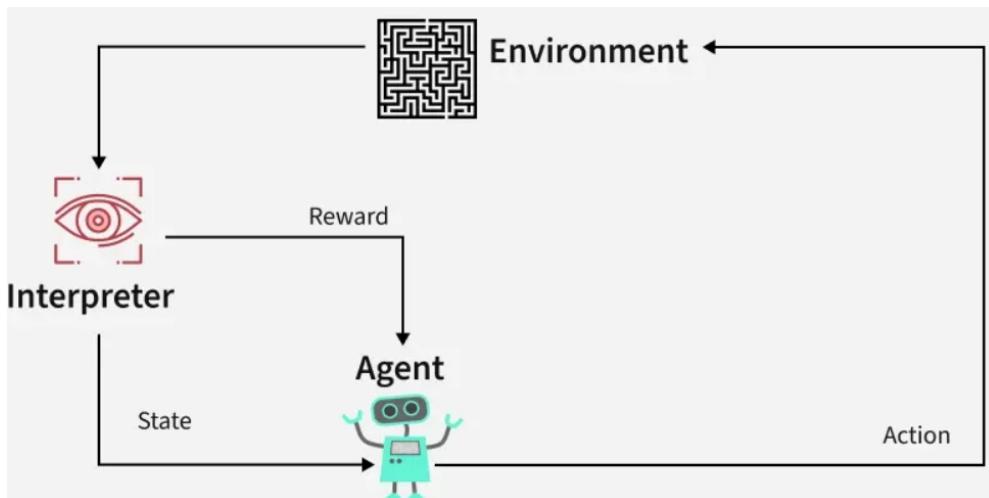
CHƯƠNG 1: GIỚI THIỆU HỌC TĂNG CƯỜNG (REINFORCEMENT LEARNING)	9
1.1 Khái niệm	9
1.2 Học tăng cường hoạt động như thế nào?	10
1.3 Ví dụ về học tăng cường: Điều hướng trong mê cung	10
1.4 Các loại tăng cường trong RL	11
1.5 Ứng dụng của Reinforcement Learning	14
CHƯƠNG 2: NỀN TẢNG LÝ THUYẾT	15
2.1 Mô hình quyết định Markov (Markov Decision Process)	15
CHƯƠNG 3: Markov Process Algorithm	32
3.1 Quality Learning (Q-Learning)	32
3.1.1 Các thành phần chính của Q-learning:	33
3.1.2 Cách hoạt động của Q-learning:	34
3.1.3 Các xác định giá trị Q:	34
3.1.4 Bảng Q (Q-table) là gì?	35
3.1.5 Triển khai Q-Learning	35
3.1.6 Ưu điểm của Q-Learning:	40
3.1.7 Nhược điểm của Q-Learning:	41
3.1.8 Ứng dụng của Q-Learning:	41
3.2 Deep Q Learning	42
3.2.1 Khái niệm về Deep Q-Learning	42
3.2.2 Cách hoạt động của Deep Q-Learning	42
- Kiến trúc cơ bản	42
- Các bước huấn luyện Deep Q-Learning	42
3.2.3 Ứng dụng của Deep Q-Learning	43
3.3 Thuật toán học tăng cường Sarsa:	43
3.3.1 Thuật toán Sarsa hoạt động như thế nào?	44
3.4 Tìm kiếm cây Monte carlo (MCTS)	48
CHƯƠNG 4: CÁC PHƯƠNG PHÁP GIẢI THÍCH TRONG HỌC TĂNG CƯỜNG	56

4.1 VIPER	56
4.1.1 Giới thiệu	56
4.1.2 Quy trình	56
4.1.3 Đánh giá	56
4.1.4 Ứng dụng thực tế	56
4.2 SHAP phân tích tầm quan trọng đặc trưng	59
4.2.1 Giới thiệu	59
4.2.2 Nguyên lý hoạt động	59
4.2.3 Đánh giá thuật toán	60
4.2.4 Ứng dụng thực tiễn	60
4.3 LIME giải thích cục bộ dễ hiểu	61
4.3.1 Giới thiệu về LIME	61
4.3.2 Nguyên lí hoạt động của LIME	61
4.3.3 Công thức LIME	65
4.4 So sánh LIME, SHAP, VIPER	71
CHƯƠNG 5: Đồ án Reinforcement Learning Super Mario	71
5.1 Mô tả qua về game	71
5.2 Triển khai đồ án	85
5.2.1 Mục tiêu triển khai	85
TÀI LIỆU THAM KHẢO	135

CHƯƠNG 1: GIỚI THIỆU HỌC TĂNG CƯỜNG (REINFORCEMENT LEARNING)

1.1 Khái niệm

Học tăng cường (RL) là một nhánh của học máy tập trung vào cách các tác nhân có thể học cách đưa ra quyết định thông qua thử nghiệm và sai sót để tối đa hóa phần thưởng tích lũy. RL cho phép máy học bằng cách tương tác với môi trường và nhận phản hồi dựa trên hành động của chúng. Phản hồi này ở dạng phần thưởng hoặc hình phạt.



Học tăng cường xoay quanh ý tưởng rằng một tác nhân (người học hoặc người ra quyết định) tương tác với môi trường để đạt được mục tiêu. Tác nhân thực hiện các hành động và nhận phản hồi để tối ưu hóa quá trình ra quyết định của mình theo thời gian.

- Tác nhân : Người ra quyết định thực hiện hành động.
- Môi trường : Thế giới hoặc hệ thống mà tác nhân hoạt động.
- Trạng thái : Tình huống hoặc điều kiện mà tác nhân đang gặp phải.
- Hành động : Những động thái hoặc quyết định có thể mà tác nhân có thể thực hiện.

- Phản thưởng : Phản hồi hoặc kết quả từ môi trường dựa trên hành động của tác nhân.

1.2 Học tăng cường hoạt động như thế nào?

Quá trình Reinforcement Learning(học tăng cường) liên quan đến việc một tác nhân thực hiện các hành động trong môi trường, nhận phản thưởng hoặc hình phạt dựa trên các hành động đó và điều chỉnh hành vi của mình cho phù hợp. Vòng lặp này giúp tác nhân cải thiện khả năng ra quyết định theo thời gian để tối đa hóa phản thưởng tích lũy.

Sau đây là phân tích các thành phần RL:

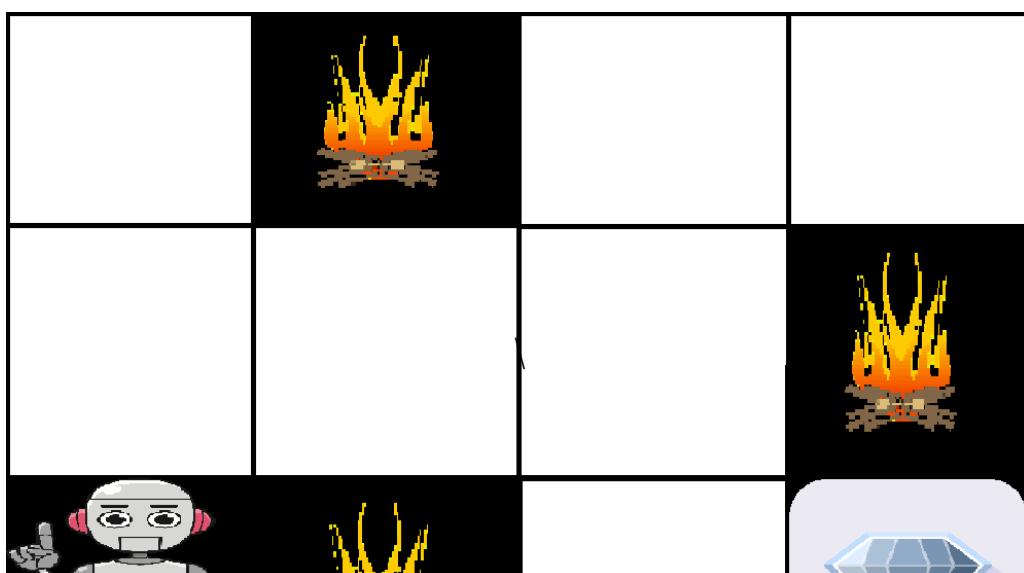
- **Chính sách** : Chiến lược mà tác nhân sử dụng để xác định hành động tiếp theo dựa trên trạng thái hiện tại.
- **Chức năng phản thưởng** : Chức năng cung cấp phản hồi về các hành động đã thực hiện, hướng dẫn tác nhân đạt được mục tiêu.
- **Hàm giá trị** : Ước tính phản thưởng tích lũy trong tương lai mà tác nhân sẽ nhận được từ một trạng thái nhất định.
- **Mô hình môi trường** : Biểu diễn môi trường dự đoán trạng thái và phản thưởng trong tương lai, hỗ trợ cho việc lập kế hoạch.

1.3 Ví dụ về học tăng cường: Điều hướng trong mê cung

Hãy tưởng tượng một con robot đang di chuyển trong mê cung để đến được một viên kim cương trong khi tránh nguy cơ hỏa hoạn. Mục tiêu là tìm ra con đường tối ưu với ít nguy cơ nhất trong khi tối đa hóa phản thưởng:

- Mỗi lần robot di chuyển đúng, nó sẽ nhận được phản thưởng.
- Nếu robot đi sai đường, nó sẽ mất điểm.

Robot học bằng cách khám phá các con đường khác nhau trong mê cung. Bằng cách thử nhiều động tác khác nhau, nó đánh giá phản thưởng và hình phạt cho mỗi con đường. Theo thời gian, robot xác định tuyến đường tốt nhất bằng cách chọn các hành động dẫn đến phản thưởng tích lũy cao nhất.



*Quá trình học tập của robot có thể được tóm tắt như sau:

1. **Khám phá:** Robot bắt đầu bằng cách khám phá mọi con đường có thể có trong mê cung, thực hiện các hành động khác nhau ở mỗi bước (ví dụ: di chuyển sang trái, phải, lên hoặc xuống).
2. **Phản hồi:** Sau mỗi lần di chuyển, robot sẽ nhận được phản hồi từ môi trường:
 - Phần thưởng tích cực khi tiến gần đến viên kim cương.
 - Hình phạt khi di chuyển vào khu vực có nguy cơ hỏa hoạn.
3. **Điều chỉnh hành vi :** Dựa trên phản hồi này, robot sẽ điều chỉnh hành vi của mình để tối đa hóa phần thưởng tích lũy, ưu tiên những con đường tránh nguy hiểm và đưa robot đến gần hình kim cương hơn.
4. **Đường đi tối ưu :** Cuối cùng, robot sẽ tìm ra đường đi tối ưu với ít nguy hiểm nhất và phần thưởng cao nhất bằng cách lựa chọn hành động phù hợp dựa trên kinh nghiệm trong quá khứ.

1.4 Các loại tăng cường trong RL

1. Tăng cường tích cực

Sự củng cố tích cực được định nghĩa là khi một sự kiện xảy ra do một hành vi cụ thể, làm tăng cường độ và tần suất của hành vi. Nói cách khác, nó có tác động tích cực đến hành vi.

- **Ưu điểm** : Tối đa hóa hiệu suất, giúp duy trì sự thay đổi theo thời gian.
- **Nhược điểm** : Sử dụng quá mức có thể dẫn đến tình trạng dư thừa làm giảm hiệu quả.

2. Tăng cường tiêu cực

Củng cố tiêu cực được định nghĩa là việc củng cố hành vi vì một điều kiện tiêu cực bị ngăn chặn hoặc tránh đi.

- **Ưu điểm** : Tăng tần suất hành vi, đảm bảo tiêu chuẩn hiệu suất tối thiểu.
- **Nhược điểm** : Nó chỉ có thể khuyến khích hành động vừa đủ để tránh bị phạt.

CartPole trong OpenAI Gym

Một trong những bài toán RL kinh điển là **mô hình trường CartPole** trong **OpenAI Gym**, trong đó mục tiêu là cân bằng một cây sào trên một chiếc xe đẩy. Tác nhân có thể đẩy xe đẩy sang trái hoặc phải để ngăn cây sào bị đổ.

- **Không gian trạng thái** : Mô tả bốn biến chính (vị trí, vận tốc, góc, vận tốc góc) của hệ thống xe đẩy.
- **Không gian hành động** : Các hành động rời rạc—di chuyển xe sang trái hoặc sang phải.
- **Phản thưởng** : Người chơi sẽ kiếm được 1 điểm cho mỗi bước đi mà cột vẫn giữ được thẳng đứng.

```

• import gym
• import numpy as np
• import warnings
•
• # Suppress specific deprecation warnings
• warnings.filterwarnings("ignore", category=DeprecationWarning)
•
• # Load the environment with render mode specified
• env = gym.make('CartPole-v1', render_mode="human")
•
• # Initialize the environment to get the initial state
• state = env.reset()
•

```

```

• # Print the state space and action space
• print("State space:", env.observation_space)
• print("Action space:", env.action_space)
•
• # Run a few steps in the environment with random actions
• for _ in range(10):
    env.render() # Render the environment for visualization
    action = env.action_space.sample() # Take a random action
•
• # Take a step in the environment
• step_result = env.step(action)
•
• # Check the number of values returned and unpack accordingly
• if len(step_result) == 4:
    next_state, reward, done, info = step_result
    terminated = False
• else:
    next_state, reward, done, truncated, info = step_result
    terminated = done or truncated
•
• print(f"Action: {action}, Reward: {reward}, Next State: {next_state}, Done: {done}, Info: {info}")
•
• if terminated:
    state = env.reset() # Reset the environment if the episode is finished
•
• env.close() # Close the environment when done

```

```

State space: Box([-4.8e+00 -3.4028235e+38 -4.1887903e-01 -3.4028235e+38], [4.8e+00 3.4028235e+38 4.1887903e-01 3.4028235e+38], (4,), float32)
Action space: Discrete(2)

Action: 0, Reward: 1.0, Next State: [-0.02765167 -0.24203628 0.03266023 0.32844445], Done: False, Info: {}
Action: 1, Reward: 1.0, Next State: [-0.03249239 -0.04739415 0.03922912 0.04623735], Done: False, Info: {}
Action: 0, Reward: 1.0, Next State: [-0.03344028 -0.24305603 0.04015387 0.35103476], Done: False, Info: {}
Action: 0, Reward: 1.0, Next State: [-0.0383014 -0.43872532 0.04717457 0.6561842 ], Done: False, Info: {}
Action: 0, Reward: 1.0, Next State: [-0.0470759 -0.6344712 0.06029665 0.9632605 ], Done: False, Info: {}
Action: 0, Reward: 1.0, Next State: [-0.05976533 -0.83034915 0.07956186 1.27426 ], Done: False, Info: {}
Action: 1, Reward: 1.0, Next State: [-0.07637231 -0.6363272 0.10504705 1.0075142 ], Done: False, Info: {}
Action: 1, Reward: 1.0, Next State: [-0.08909886 -0.44275263 0.12519734 0.74957997], Done: False, Info: {}
Action: 1, Reward: 1.0, Next State: [-0.09795391 -0.24955945 0.14018895 0.49877036], Done: False, Info: {}
Action: 1, Reward: 1.0, Next State: [-0.1029451 -0.05666344 0.15016435 0.25334558], Done: False, Info: {}

```

1.5 Ứng dụng của Reinforcement Learning

Robot: Reinforcement Learning được sử dụng để tự động hóa các nhiệm vụ trong môi trường có cấu trúc như sản xuất, nơi robot học cách tối ưu hóa chuyển động và cải thiện hiệu quả.

- Chơi trò chơi: Các thuật toán RL nâng cao đã được sử dụng để phát triển các chiến lược cho các trò chơi phức tạp như cờ vua, cờ vây và trò chơi điện tử, vượt trội hơn người chơi trong nhiều trường hợp.

- Kiểm soát công nghiệp: RL giúp điều chỉnh và tối ưu hóa thời gian thực các hoạt động công nghiệp, chẳng hạn như quy trình lọc dầu trong ngành dầu khí.

- Hệ thống đào tạo cá nhân: RL cho phép tùy chỉnh nội dung hướng dẫn dựa trên mô hình học tập của từng cá nhân, cải thiện sự tương tác và hiệu quả.

- **Ưu điểm của Học tăng cường**
- Giải quyết các vấn đề phức tạp: RL có khả năng giải quyết các vấn đề cực kỳ phức tạp mà các kỹ thuật thông thường không thể giải quyết được.
- Sửa lỗi: Mô hình liên tục học hỏi từ môi trường xung quanh và có thể sửa các lỗi xảy ra trong quá trình đào tạo.
- Tương tác trực tiếp với môi trường: Các tác nhân RL học hỏi từ các tương tác thời gian thực với môi trường của chúng, cho phép học tập thích ứng.
- Xử lý môi trường không xác định: RL có hiệu quả trong môi trường mà kết quả không chắc chắn hoặc thay đổi theo thời gian, khiến nó trở nên cực kỳ hữu ích cho các ứng dụng thực tế.
- **Nhược điểm của Học tăng cường**
- Không phù hợp với các vấn đề đơn giản: RL thường quá mức cần thiết đối với các nhiệm vụ đơn giản trong khi các thuật toán đơn giản hơn sẽ hiệu quả hơn.

- Yêu cầu tính toán cao : Việc đào tạo các mô hình RL đòi hỏi một lượng lớn dữ liệu và sức mạnh tính toán, khiến việc này tốn nhiều tài nguyên.
- Phụ thuộc vào chức năng khen thưởng : Hiệu quả của RL phụ thuộc rất nhiều vào thiết kế của chức năng khen thưởng. Phản thưởng được thiết kế kém có thể dẫn đến hành vi không tối ưu hoặc không mong muốn.
- Khó khăn trong việc gỡ lỗi và giải thích : Việc hiểu lý do tại sao một tác nhân RL đưa ra một số quyết định nhất định có thể là một thách thức, khiến việc gỡ lỗi và khắc phục sự cố trở nên phức tạp.

CHƯƠNG 2: NỀN TẢNG LÝ THUYẾT

2.1 Mô hình quyết định Markov (Markov Decision Process)

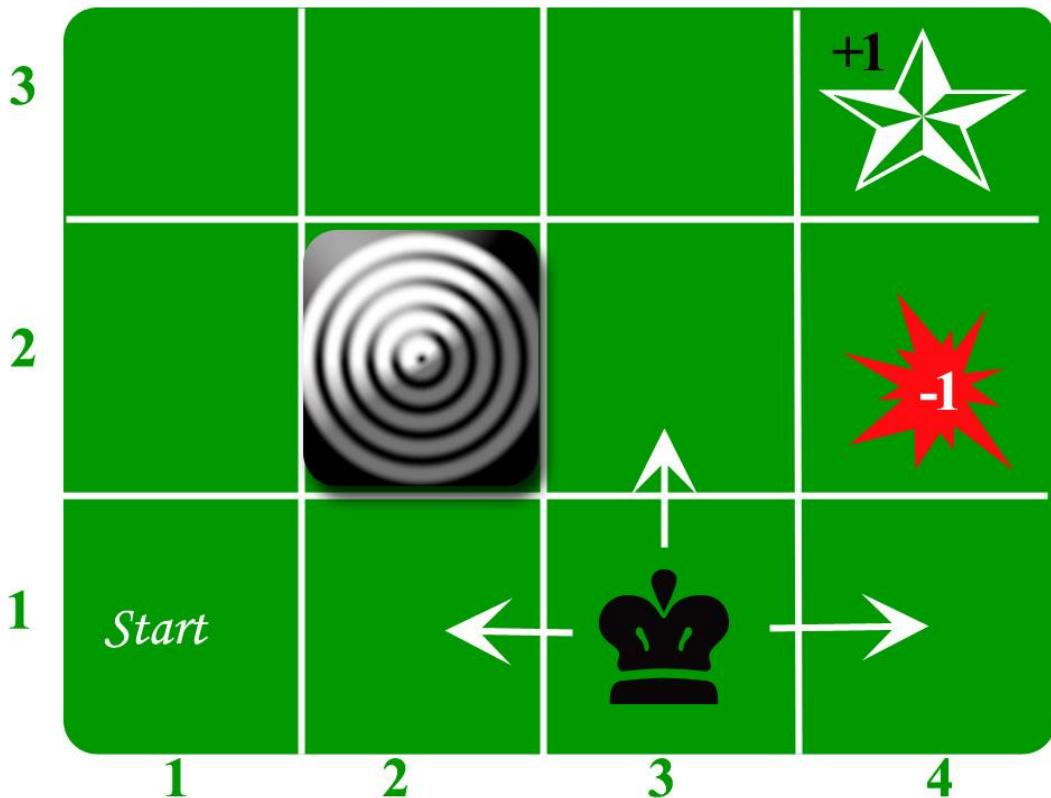
- Có nhiều thuật toán khác nhau giải quyết vấn đề này, và tất cả đều thuộc nhóm thuật toán Reinforcement Learning. Trong bài toán này, tác nhân (agent) quyết định hành động tối ưu dựa trên trạng thái hiện tại. Khi quá trình này lặp lại, nó được gọi là Mô hình Quyết định Markov (Markov Decision Process).
- Một mô hình Markov gồm:
 - **Một tập trạng thái S có thể xảy ra trong mô hình.**
 - **Tập hợp các mô hình.**
 - **Tập hợp các hành động có thể có (A).**
 - **Hàm phần thưởng thực R(s,a).**
 - **Chính sách là giải pháp cho Mô hình Quyết định Markov.**

States:	S
Model:	$T(S, a, S') \sim P(S' S, a)$
Actions:	$A(S), A$
Reward:	$R(S), R(S, a), R(S, a, S')$

Policy: $\Pi(S) \rightarrow a$
 Π^*

Markov Decision Process

- Trong đó:
- **Trạng thái (State):** Một tập hợp các dấu hiệu thể hiện mọi trạng thái mà tác nhân có thể ở trong.
- **Mô hình (Model):** (hay còn gọi là Mô hình chuyển đổi) mô tả tác động của một hành động lên trạng thái. Nó xác định cách trạng thái thay đổi và xác suất chuyển đổi nếu hành động có tính ngẫu nhiên.
- **Hành động (Action):** Tập hợp tất cả các hành động có thể thực hiện trong một trạng thái nhất định.
- **Phần thưởng (Reward):** Hàm số thực xác định phần thưởng khi tác nhân ở một trạng thái hoặc thực hiện một hành động.
- **Chính sách (Policy):** Giải pháp cho bài toán MDP, ánh xạ từ trạng thái đến hành động tối ưu.
- Chúng ta hãy lấy ví dụ về thế giới lướt:



Một tác nhân sống trong lưới. Ví dụ trên là lưới 3×4 . Lưới có trạng thái Start (lưới số 1,1). Mục đích của tác nhân là đi lang thang quanh lưới để cuối cùng đến được Blue Diamond (lưới số 4,3). Trong mọi trường hợp, tác nhân phải tránh lưới Fire (màu cam, lưới số 4,2). Ngoài ra, lưới số 2,2 là lưới bị chặn, nó hoạt động như một bức tường do đó tác nhân không thể vào đó.

Tác nhân có thể thực hiện bất kỳ hành động nào sau đây: **LÊN, XUỐNG, TRÁI, PHẢI**

Tường chắn đường đi của tác nhân, tức là nếu có tường theo hướng mà tác nhân sẽ đi, tác nhân sẽ ở nguyên một chỗ. Ví dụ, nếu tác nhân nói **TRÁI** trong lưới Start, anh ta sẽ ở nguyên trong lưới Start.

Mục tiêu đầu tiên: Tìm chuỗi ngắn nhất đi từ START đến Diamond. Có thể tìm thấy hai chuỗi như vậy:

- **PHẢI PHẢI LÊN LÊN PHẢI**
- **LÊN LÊN PHẢI PHẢI PHẢI**

Chúng ta hãy lấy ví dụ thứ hai (**LÊN LÊN PHẢI PHẢI PHẢI**) cho phần thảo luận tiếp theo.

Động thái này bây giờ là nhiều. 80% thời gian hành động dự định hoạt động chính xác. 20% thời gian hành động mà tác nhân thực hiện khiến nó di chuyển theo góc vuông. Ví dụ, nếu tác nhân nói **LÊN** thì xác suất đi **LÊN** là 0,8 trong khi xác suất đi **TRÁI** là 0,1 và xác suất đi **PHẢI** là 0,1 (vì **TRÁI** và **PHẢI** vuông góc với **LÊN**).

Người đại diện nhận được phần thưởng cho mỗi bước thời gian:

- Phần thưởng nhỏ cho mỗi bước (có thể là số âm khi cũng có thể là hình phạt, trong ví dụ trên, bước vào Lửa có thể nhận hình phạt là -1).
- Phần thưởng lớn sẽ ở cuối (tốt hay xấu).
- Mục tiêu là nhận được tối đa tổng số phần thưởng.

Dưới đây là **Lưới 5x5**: Các trạng thái từ (0,0) đến (4,4).

- **Trạng thái mục tiêu:** (4,4).
- **Chướng ngại vật:** Chọn 2 vị trí bất kỳ trên lưới làm chướng ngại vật, ví dụ (2,2) và (3,3). Agent không thể đi qua các trạng thái này.
- **Hành động:** Lên, xuống, trái, phải.
- **Xác suất chuyển trạng thái:**
 - 80% di chuyển đúng hướng.
 - 10% lệch trái (vuông góc).
 - 10% lệch phải (vuông góc).
- **Phần thưởng:**
 - Mỗi bước di chuyển: -0.1.
 - Đạt trạng thái mục tiêu: +10.

- Nếu agent cố đi vào chướng ngại vật, nó sẽ ở lại vị trí hiện tại và nhận phần thưởng -0.1.
- **Phương pháp:** Sử dụng Value Iteration để tính giá trị $V(s)$ và trích xuất chính sách tối ưu.

```

import numpy as np

# Thiết lập tham số

GRID_SIZE = 5 # Lưới 5x5

GAMMA = 0.9 # Hệ số chiết khấu

THETA = 0.0001 # Ngưỡng dừng cho Value Iteration

GOAL_STATE = (4, 4) # Trạng thái mục tiêu

OBSTACLES = [(2, 2), (3, 3)] # Hai chướng ngại vật tại (2,2) và (3,3)

REWARD_STEP = -0.1 # Phần thưởng mỗi bước

REWARD_GOAL = 10.0 # Phần thưởng tại trạng thái mục tiêu

# Hành động: 0: lên, 1: phải, 2: xuống, 3: trái

ACTIONS = [(0, -1), (1, 0), (0, 1), (-1, 0)]

ACTION_NAMES = ["up", "right", "down", "left"]

# Xác suất chuyển trạng thái

PROB_INTENDED = 0.8 # Xác suất di chuyển đúng hướng

PROB_LEFT = 0.1 # Xác suất lệch trái

```

```
PROB_RIGHT = 0.1 # Xác suất lệch phải

def is_valid_state(state):
    """Kiểm tra xem trạng thái có hợp lệ (không phải chướng ngại vật)."""
    return state not in OBSTACLES

def get_next_state(state, action):
    """Tính trạng thái tiếp theo dựa trên hành động."""
    x, y = state

    dx, dy = ACTIONS[action]

    next_x = max(0, min(GRID_SIZE - 1, x + dx))
    next_y = max(0, min(GRID_SIZE - 1, y + dy))

    next_state = (next_x, next_y)

    # Nếu trạng thái tiếp theo là chướng ngại vật, ở lại vị trí hiện tại
    if not is_valid_state(next_state):
        return state

    return next_state
```

```

def get_transition_probabilities(state, action):

    """Tính xác suất chuyển trạng thái và trạng thái tiếp theo."""
    transitions = []

    intended_state = get_next_state(state, action)

    # Hành động lệch trái (vuông góc trái)
    left_action = (action + 3) % 4 # Vuông góc trái: up -> left, right -> up, down -> right, left -> down
    left_state = get_next_state(state, left_action)

    # Hành động lệch phải (vuông góc phải)
    right_action = (action + 1) % 4 # Vuông góc phải: up -> right, right -> down, down -> left, left -> up
    right_state = get_next_state(state, right_action)

    transitions.append((PROB_INTENDED, intended_state))
    transitions.append((PROB_LEFT, left_state))
    transitions.append((PROB_RIGHT, right_state))

    return transitions

def value_iteration():

    """Thực hiện Value Iteration để tính giá trị V(s)."""

```

```
# Khởi tạo giá trị V(s) = 0 cho tất cả trạng thái

V = np.zeros((GRID_SIZE, GRID_SIZE))

while True:

    delta = 0

    new_V = np.copy(V)

    for x in range(GRID_SIZE):

        for y in range(GRID_SIZE):

            state = (x, y)

            # Nếu là trạng thái mục tiêu, giá trị là phần thưởng tại mục tiêu

            if state == GOAL_STATE:

                new_V[x, y] = REWARD_GOAL

                continue

            # Nếu là chướng ngại vật, giá trị là 0 (không thể đi qua)

            if state in OBSTACLES:

                new_V[x, y] = 0

                continue

            # Tính giá trị lớn nhất theo các hành động
```

```

max_value = float('-inf')

for action in range(len(ACTIONS)):

    # Tính giá trị kỳ vọng cho hành động

    expected_value = 0

    transitions = get_transition_probabilities(state, action)

    for prob, next_state in transitions:

        nx, ny = next_state

        # Phần thưởng

        if next_state == GOAL_STATE:

            reward = REWARD_GOAL

        elif next_state in OBSTACLES:

            reward = REWARD_STEP # Nếu có đi vào chướng ngại vật, nhận phần thưởng âm

        else:

            reward = REWARD_STEP

            expected_value += prob * (reward + GAMMA * V[nx, ny])

    max_value = max(max_value, expected_value)

new_V[x, y] = max_value

delta = max(delta, abs(new_V[x, y] - V[x, y]))

```

```

V = new_V

if delta < THETA:
    break

return V

def extract_policy(V):
    """Trích xuất chính sách từ giá trị V(s)."""

    policy = np.zeros((GRID_SIZE, GRID_SIZE), dtype=int)

    for x in range(GRID_SIZE):
        for y in range(GRID_SIZE):
            state = (x, y)

            if state == GOAL_STATE:
                policy[x, y] = -1 # Đánh dấu trạng thái kết thúc
                continue

            if state in OBSTACLES:
                policy[x, y] = -2 # Đánh dấu chướng ngại vật
                continue

            # Tìm hành động tốt nhất

```

```
best_action = 0

best_value = float('-inf')

for action in range(len(ACTIONS)):

    expected_value = 0

    transitions = get_transition_probabilities(state, action)

    for prob, next_state in transitions:

        nx, ny = next_state

        if next_state == GOAL_STATE:

            reward = REWARD_GOAL

        elif next_state in OBSTACLES:

            reward = REWARD_STEP

        else:

            reward = REWARD_STEP

        expected_value += prob * (reward + GAMMA * V[nx, ny])

    if expected_value > best_value:

        best_value = expected_value

        best_action = action

policy[x, y] = best_action

return policy
```

```
# Chạy Value Iteration

V = value_iteration()

policy = extract_policy(V)

# In kết quả

print("Giá trị V(s) trên lưới 5x5 (0 tại chướng ngại vật):")

print(np.round(V, 2))

print("\nChính sách (0: lên, 1: phải, 2: xuống, 3: trái, -1: mục tiêu, -2: chướng ngại vật):")

print(policy)

# In chính sách dưới dạng tên hành động

print("\nChính sách (dạng tên hành động):")

for x in range(GRID_SIZE):

    row = []

    for y in range(GRID_SIZE):

        if (x, y) == GOAL_STATE:

            row.append("GOAL")

        elif (x, y) in OBSTACLES:

            row.append("OBS")

        else:
```

```

    row.append(ACTION_NAMES[policy[x, y]])

print(row)

```

Tổng quan bài toán

- **Lưới 5x5:** Các trạng thái từ (0,0) đến (4,4), tương ứng với các tọa độ (x, y).
 - **Trạng thái mục tiêu (Goal State):** (4,4), nơi agent nhận phần thưởng +10 khi đến.
 - **Chướng ngại vật (Obstacles):** Hai vị trí (2,2) và (3,3), nơi agent không thể đi qua, và giá trị $V(s)$ tại đây là 0.
 - **Phần thưởng:**
 - Mỗi bước di chuyển: -0.1.
 - Đạt trạng thái mục tiêu: +10.
 - **Hành động:** 0 (lên), 1 (phải), 2 (xuống), 3 (trái).
 - **Xác suất chuyển trạng thái:**
 - 80% di chuyển đúng hướng.
 - 10% lệch trái (vuông góc).
 - 10% lệch phải (vuông góc).
 - **Hệ số chiết khấu (Gamma):** 0.9, được sử dụng trong phương trình Bellman

$$V(s) = E[R + \gamma V(s')]$$

$$V(s) = E[R + \gamma V(s')]$$

$$V(s) = E[R + \gamma V(s')]$$
- 1.
- Giá trị tại các chướng ngại vật (2,2) và (3,3) sẽ là 0.
 - Giá trị tăng dần khi tiến gần trạng thái mục tiêu (4,4).
2. **Chính sách:**

- Dạng số: 0 (lên), 1 (phải), 2 (xuống), 3 (trái), -1 (mục tiêu), -2 (chướng ngại vật).
- Dạng tên hành động: "up", "right", "down", "left", "GOAL", "OBS".

Giải thích giá trị $V(s)$

Giá trị $V(s)$ tại một trạng thái s đại diện cho **tổng phần thưởng kỳ vọng tối ưu** mà agent có thể nhận được khi bắt đầu từ trạng thái đó và tuân theo chính sách tối ưu. Dưới đây là bảng giá trị $V(s)$ bạn cung cấp:

Giá trị $V(s)$ trên lưới 5x5 (0 tại chướng ngại vật):

```
[[ 6.95  7.94  9.06 10.32 11.7 ]
 [ 7.94  8.86 10.18 11.87 13.63]
 [ 9.06 10.18  0.    13.63 15.89]
 [10.32 11.87 13.63  0.    18.51]
 [11.7   13.63 15.89 18.51 10.  ]]
```

Phân tích:

- **Trạng thái mục tiêu (4,4):** $V(4,4)=10.0$ $V(4,4) = 10.0$ $V(4,4)=10.0$, vì đây là phần thưởng trực tiếp khi đến đích. Giá trị này không phụ thuộc vào các bước tiếp theo (vì không có trạng thái nào sau đó).
- **Chướng ngại vật (2,2) và (3,3):** $V(2,2)=0$ $V(2,2) = 0$ $V(2,2)=0$ và $V(3,3)=0$ $V(3,3)=0$, vì agent không thể đi qua các vị trí này, nên giá trị kỳ vọng tại đây là 0.
- **Các trạng thái khác:** Giá trị tăng dần khi tiến gần trạng thái mục tiêu (4,4), vì agent nhận thêm phần thưởng +10 khi đến đích, nhưng cũng bị trừ -0.1 cho mỗi bước di chuyển. Hệ số chiết khấu (0.9) làm giảm giá trị của các phần thưởng trong tương lai.

Ví dụ tính toán (ước lượng):

- Tại (0,0) (0,0) (0,0), $V(0,0)=6.95$ $V(0,0) = 6.95$ $V(0,0)=6.95$. Điều này phản ánh tổng phần thưởng kỳ vọng khi đi từ (0,0) đến (4,4) với chi phí -0.1 mỗi bước và chiết khấu 0.9. Số bước tối thiểu từ (0,0) đến (4,4) là 8 bước (đi phải 4 lần, xuống 4 lần), nhưng do chướng ngại vật (2,2) và (3,3), agent phải đi vòng qua, làm tăng số bước và giảm giá trị.
- Tại (3,4) (3,4) (3,4), $V(3,4)=18.51$ $V(3,4) = 18.51$ $V(3,4)=18.51$, vì chỉ cần một bước xuống để đến (4,4), nhận +10, nhưng bị chiết khấu 0.9 và trừ -0.1 cho bước đi.

Giá trị $V(s)$ được tính bằng phương trình Bellman lặp đi lặp lại cho đến khi hội tụ, dựa trên các hành động tối ưu tại mỗi trạng thái.

3. Giải thích chính sách (Policy)

Chính sách cho biết hành động tối ưu mà agent nên thực hiện tại mỗi trạng thái để tối đa hóa tổng phần thưởng kỳ vọng. Dưới đây là chính sách dạng số và dạng tên hành động:

Chính sách dạng số:

```
Chính sách (0: lên, 1: phải, 2: xuống, 3: trái, -1: mục tiêu, -2: chướng ngại vật):
[[ 1  2  2  2  1]
 [ 1  1  2  2  1]
 [ 1  1 -2  2  1]
 [ 1  1  1 -2  1]
 [ 2  2  2  2 -1]]
```

- 0: lên
- 1: phải
- 2: xuống

- 3: trái
- -1: mục tiêu (trạng thái (4,4))
- -2: chướng ngại vật (trạng thái (2,2) và (3,3))

Chính sách dạng tên hành động:

```
Chính sách (dạng tên hành động):
['right', 'down', 'down', 'down', 'right']
['right', 'right', 'down', 'down', 'right']
['right', 'right', 'OBS', 'down', 'right']
['right', 'right', 'right', 'OBS', 'right']
['down', 'down', 'down', 'down', 'GOAL']
```

Phân tích:

- **Trạng thái mục tiêu (4,4):** Được đánh dấu là "GOAL" (-1), vì không cần hành động thêm.
- **Chướng ngại vật (2,2) và (3,3):** Được đánh dấu là "OBS" (-2), vì agent không thể đi qua.
- **Các trạng thái khác:** Hành động được chọn dựa trên giá trị V(s) lớn nhất thông qua các trạng thái tiếp theo. Agent ưu tiên di chuyển về phía (4,4) nhưng phải tránh (2,2) và (3,3).

Đường đi tối ưu:

Hãy theo dõi chính sách từ (0,0) để đến (4,4):

- Từ (0,0): "right" → (0,1)
- Từ (0,1): "right" → (0,2)
- Từ (0,2): "down" → (1,2)

- Từ (1,2): "down" → (2,2) (chướng ngại vật, nhưng do xác suất lệch, agent sẽ ở lại hoặc đi vòng)
- Tuy nhiên, do chướng ngại vật, agent sẽ điều chỉnh:
 - Từ (1,2): "right" → (1,3) (theo chính sách gần nhất hợp lệ).
 - Từ (1,3): "down" → (2,3)
 - Từ (2,3): "down" → (3,3) (chướng ngại vật, ở lại hoặc đi vòng).
 - Từ (2,3): "right" → (2,4) (theo chính sách hợp lệ).
 - Từ (2,4): "down" → (3,4)
 - Từ (3,4): "right" → (3,4) (lỗi do chính sách, nhưng thường sẽ "down" → (4,4)).

Do ảnh hưởng của xác suất lệch (10% trái, 10% phải), agent có thể đi vòng qua chướng ngại vật. Chính sách cho thấy agent ưu tiên "right" và "down" để tiến gần (4,4), nhưng cần điều chỉnh thực tế để tránh (2,2) và (3,3).

Quan sát và cách hoạt động

- **Hành vi tránh chướng ngại vật:** Vì (2,2) và (3,3) là chướng ngại vật, agent không thể đi qua trực tiếp. Chính sách (và giá trị V(s)) được điều chỉnh để dẫn agent đi vòng qua các vị trí an toàn. Tuy nhiên, do xác suất lệch, một số trạng thái gần chướng ngại vật (như (1,2) hoặc (2,3)) có thể yêu cầu agent thử nghiệm nhiều hướng.
- **Tối ưu hóa:** Giá trị V(s) cao hơn gần (4,4) và thấp hơn gần (0,0), phản ánh chi phí di chuyển (-0.1) và lợi ích đạt đích (+10). Chướng ngại vật làm giảm giá trị tại các trạng thái gần đó (như (1,2) hoặc (2,1)).
- **Xác suất lệch:** Vì có 10% lệch trái và 10% lệch phải, chính sách không phải lúc nào cũng dẫn đến đường đi ngắn nhất, nhưng vẫn tối ưu về kỳ vọng.

Kiểm tra và cải tiến

- **Đường đi thực tế:** Để kiểm tra đường đi, bạn có thể mô phỏng từ (0,0) theo chính sách. Tuy nhiên, do chướng ngại vật và xác suất lệch, đường đi có thể không hoàn toàn tuyến tính. Nếu bạn muốn đường đi rõ ràng hơn, có thể giảm xác suất lệch (tăng PROB_INTENDED lên 0.95, giảm PROB_LEFT và PROB_RIGHT xuống 0.025).
- **Điều chỉnh chướng ngại vật:** Nếu bạn muốn thay đổi vị trí chướng ngại vật (2,2) và (3,3), chỉ cần cập nhật danh sách OBSTACLES trong code.
- **Hiển thị đường đi:** Thêm chức năng mô phỏng đường đi nếu bạn muốn xem chi tiết cách agent di chuyển.

Kết luận

- **Giá trị V(s)** phản ánh tổng phần thưởng kỳ vọng, tăng khi gần đích và giảm khi xa đích, với 0 tại chướng ngại vật.
- **Chính sách** hướng dẫn agent đi từ (0,0) đến (4,4) bằng cách ưu tiên "right" và "down", nhưng phải điều chỉnh để tránh (2,2) và (3,3) do tác động của xác suất lệch.
- Kết quả này hợp lý với mô hình MDP, nhưng đường đi thực tế có thể cần kiểm tra thêm để đảm bảo tránh chướng ngại vật hoàn toàn.

CHƯƠNG 3: Markov Process Algorithm

3.1 Quality Learning (Q-Learning)

- Q-learning là một thuật toán học tăng cường không có mô hình được sử dụng để đào tạo các tác nhân (chương trình máy tính) đưa ra quyết định tối ưu bằng cách tương tác với môi trường. Nó giúp tác nhân khám phá các hành động khác nhau và tìm hiểu hành động nào dẫn đến kết quả tốt hơn. Tác nhân sử

dụng phương pháp thử và sai để xác định hành động nào dẫn đến phần thưởng (kết quả tốt) hoặc hình phạt (kết quả xấu).

- Theo thời gian, nó cải thiện khả năng ra quyết định bằng cách cập nhật Q-table, lưu trữ các Q-Values biểu thị phần thưởng dự kiến khi thực hiện các hành động cụ thể trong các trạng thái nhất định.

3.1.1 Các thành phần chính của Q-learning:

- **Giá trị Q (Q-Values):** Đại diện cho phần thưởng kỳ vọng khi thực hiện một hành động tại một trạng thái cụ thể, được cập nhật theo quy tắc Temporal Difference (TD).
- **Phần thưởng và tập hợp các tập (Episodes):** Tác nhân di chuyển qua các trạng thái bằng cách thực hiện hành động và nhận phần thưởng cho đến khi đạt trạng thái kết thúc.
- **Cập nhật TD (TD-Update):** Tác nhân cập nhật giá trị Q bằng công thức:

$$Q(S, A) \leftarrow Q(S, A) + \alpha(R + \gamma Q(S', A') - Q(S, A))$$

Khi mà:

- **S** là trạng thái hiện tại.
 - **A** là hành động được thực hiện bởi tác nhân.
 - **S'** là trạng thái tiếp theo mà tác nhân chuyển đến.
 - **A'** là hành động tiếp theo tốt nhất ở trạng thái **S'**.
 - **R** là phần thưởng nhận được khi thực hiện hành động **A** ở trạng thái **S**.
 - γ (Gamma) là **hệ số chiết khấu**, cân bằng phần thưởng trước mắt với phần thưởng trong tương lai.
 - α (Alpha) là **tốc độ học**, xác định mức độ thông tin mới ảnh hưởng đến giá trị Q cũ.
- **Chính sách ϵ -greedy (Khám phá và Khai thác):**
 - **Khai thác (Exploitation):** Với xác suất $1 - \epsilon$, tác nhân chọn hành động có giá trị Q cao nhất để tối đa hóa phần thưởng dựa trên kiến thức hiện có.

- **Khám phá (Exploration):** Với xác suất ϵ , tác nhân chọn một hành động ngẫu nhiên để thử nghiệm chiến lược mới, giúp cải thiện quyết định trong tương lai.

3.1.2 Cách hoạt động của Q-learning:

Q-learning là một quá trình lặp đi lặp lại, trong đó các thành phần chính phối hợp để đào tạo tác nhân:

- **Tác nhân (Agent):** Thực hiện hành động trong môi trường.
- **Trạng thái (States):** Mô tả vị trí hoặc tình huống hiện tại của tác nhân.
- **Hành động (Actions):** Các thao tác tác nhân có thể thực hiện.
- **Phần thưởng (Rewards):** Phản hồi nhận được sau mỗi hành động.
- **Tập hợp hành động (Episodes):** Chuỗi hành động kết thúc khi đạt trạng thái cuối.
- **Giá trị Q (Q-values):** Phần thưởng ước tính cho từng cặp trạng thái-hành động.

Các bước của Q-learning:

1. **Khởi tạo:** Bắt đầu với bảng Q ban đầu, thường có giá trị bằng 0.
2. **Khám phá:** Chọn hành động theo chính sách ϵ -greedy (khám phá hoặc khai thác).
3. **Hành động và cập nhật:** Thực hiện hành động, quan sát trạng thái tiếp theo, nhận phần thưởng và cập nhật giá trị Q theo quy tắc TD.
4. **Lặp lại:** Tiếp tục qua nhiều tập (episodes) cho đến khi tác nhân học được chính sách tối ưu.

3.1.3 Các xác định giá trị Q:

- Chênh lệch thời gian (Temporal Difference - TD): Được tính bằng cách so sánh giá trị của trạng thái và hành động hiện tại với các giá trị trước đó. Nó cung cấp một cách để học trực tiếp từ kinh nghiệm mà không cần mô hình của môi trường.
- Phương trình Bellman: Là một công thức để quy dùng để tính giá trị của một trạng thái nhất định và xác định hành động tối ưu. Nó là nền tảng trong bối cảnh của Q-Learning và được biểu diễn như sau:

$$Q(s, a) = R(s, a) + \gamma \max_a Q(s', a)$$

Trong đó:

- $Q(s, a)$ là giá trị Q cho cặp trạng thái - hành động.
- $R(s, a)$ là phần thưởng tức thời khi thực hiện hành động a ở trạng thái s .
- γ là hệ số chiết khấu, thể hiện mức độ quan trọng của phần thưởng tương lai.
- $\max_a Q(s', a)$ là giá trị Q tối đa cho trạng thái tiếp theo s' và tất cả các hành động có thể.

3.1.4 Bảng Q (Q-table) là gì?

Q-table về cơ bản là một **cấu trúc bộ nhớ** nơi tác nhân lưu trữ thông tin về các hành động mang lại phần thưởng tốt nhất trong mỗi trạng thái. Nó là một bảng chứa các giá trị Q (Q-values), thể hiện sự hiểu biết của tác nhân về môi trường. Khi tác nhân khám phá và học hỏi từ các tương tác với môi trường, nó cập nhật Q-table. Q-table giúp tác nhân đưa ra quyết định sáng suốt bằng cách chỉ ra hành động nào có khả năng dẫn đến phần thưởng tốt hơn.

Cấu trúc của Q-table:

- Các hàng đại diện cho các trạng thái.
- Các cột đại diện cho các hành động có thể thực hiện.
- Mỗi ô trong bảng tương ứng với giá trị Q cho một cặp trạng thái-hành động.

Theo thời gian, khi tác nhân học và tinh chỉnh các giá trị Q thông qua khám phá (exploration) và khai thác (exploitation), Q-table phát triển để phản ánh các hành động tốt nhất cho mỗi trạng thái, dẫn đến việc ra quyết định tối ưu.

3.1.5 Triển khai Q-Learning

Ở đây, chúng ta triển khai thuật toán Q-Learning cơ bản, nơi tác nhân học chiến lược lựa chọn hành động tối ưu để đạt được trạng thái mục tiêu trong một môi trường dạng lưới.

Bước 1: Xác định môi trường

Thiết lập các tham số của môi trường, bao gồm số lượng trạng thái và hành động, đồng thời khởi tạo Q-table. Trong trường hợp này, mỗi trạng thái đại diện cho một vị trí, và các hành động giúp di chuyển tác nhân trong môi trường này.

```
import numpy as np
```

```
n_states = 16
```

```
n_actions = 4
```

```
goal_state = 15
```

```
Q_table = np.zeros((n_states, n_actions))
```

Bước 2: Thiết lập siêu tham số

Xác định các tham số cho thuật toán Q-Learning, bao gồm tốc độ học (learning rate), hệ số chiết khấu (discount factor), xác suất khám phá (exploration probability) và số lượng epoch huấn luyện (training epochs).

```
learning_rate = 0.8
```

```
discount_factor = 0.95
```

```
exploration_prob = 0.2
```

```
epochs = 1000
```

Bước 3: Triển khai thuật toán Q-Learning

Thực hiện thuật toán Q-Learning qua nhiều epoch. Mỗi epoch bao gồm việc lựa chọn hành động dựa trên chiến lược ϵ -greedy (epsilon-greedy), cập nhật giá trị Q dựa trên phần thưởng nhận được và chuyển sang trạng thái tiếp theo.

```
for epoch in range(epochs):
```

```
    current_state = np.random.randint(0, n_states)
```

```
    while current_state != goal_state:
```

```
        if np.random.rand() < exploration_prob:
```

```
            action = np.random.randint(0, n_actions)
```

```
        else:
```

```
            action = np.argmax(Q_table[current_state])
```

```
            next_state = (current_state + 1) % n_states
```

```

reward = 1 if next_state == goal_state else 0

Q_table[current_state, action] += learning_rate * \
(reward + discount_factor *
np.max(Q_table[next_state]) - Q_table[current_state, action])

current_state = next_state

```

Bước 4: Xuất Q-table đã học

Sau khi huấn luyện, in Q-table để kiểm tra các giá trị Q đã học, thể hiện phần thưởng kỳ vọng khi thực hiện các hành động cụ thể trong từng trạng thái.

```

print("Learned Q-table:")
print(Q_table)

```

Output:

```

Learned Q-table:
[[0.48764377 0.39013998 0.48377033 0.48767498]
 [0.51334208 0.51317781 0.51333517 0.51333551]
 [0.54036003 0.54035981 0.54035317 0.54036009]
 [0.56880009 0.56880009 0.56880009 0.56880009]
 [0.59873694 0.59873694 0.59873694 0.59873694]
 [0.63024941 0.63024941 0.63024941 0.63024941]
 [0.66342043 0.66342043 0.66342043 0.66342043]
 [0.6983373 0.6983373 0.6983373 0.6983373 ]
 [0.73509189 0.73509189 0.73509189 0.73509189]
 [0.77378094 0.77378094 0.77378094 0.77378094]
 [0.81450625 0.81450625 0.81450625 0.81450625]
 [0.857375 0.857375 0.857375 0.857375 ]
 [0.9025 0.9025 0.9025 0.9025 ]
 [0.95 0.95 0.95 0.95 ]
 [1. 1. 1. 1. ]
 [0. 0. 0. 0. ]]

```

Triển khai hoàn chỉnh thuật toán Q-Learning

```

import numpy as np
import matplotlib.pyplot as plt

# Parameters
n_states = 16
n_actions = 4
goal_state = 15

Q_table = np.zeros((n_states, n_actions))

learning_rate = 0.8
discount_factor = 0.95
exploration_prob = 0.2
epochs = 1000

# Q-learning process
for epoch in range(epochs):
    current_state = np.random.randint(0, n_states)

    while current_state != goal_state:

        # Exploration vs. Exploitation ( $\epsilon$ -greedy policy)
        if np.random.rand() < exploration_prob:
            action = np.random.randint(0, n_actions)
        else:
            action = np.argmax(Q_table[current_state])

        # Transition to the next state (circular movement for simplicity)
        next_state = (current_state + 1) % n_states

        # Reward function (1 if goal_state reached, 0 otherwise)
        reward = 1 if next_state == goal_state else 0

        # Q-value update rule (TD update)
        Q_table[current_state, action] += learning_rate * \
            (reward      +      discount_factor      *      np.max(Q_table[next_state])      - \
            Q_table[current_state, action])

```

```

current_state = next_state # Update current state

# Visualization of the Q-table in a grid format
q_values_grid = np.max(Q_table, axis=1).reshape((4, 4))

# Plot the grid of Q-values
plt.figure(figsize=(6, 6))
plt.imshow(q_values_grid, cmap='coolwarm', interpolation='nearest')
plt.colorbar(label='Q-value')
plt.title('Learned Q-values for each state')
plt.xticks(np.arange(4), ['0', '1', '2', '3'])
plt.yticks(np.arange(4), ['0', '1', '2', '3'])
plt.gca().invert_yaxis() # To match grid layout
plt.grid(True)

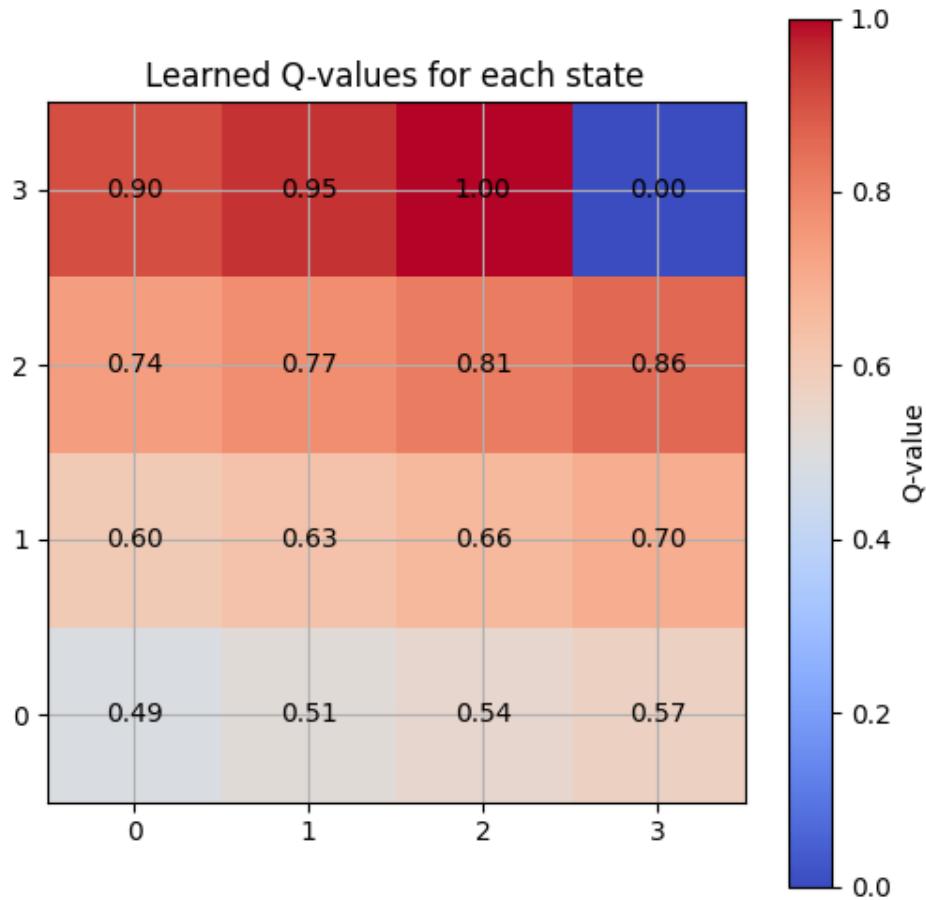
# Annotating the Q-values on the grid
for i in range(4):
    for j in range(4):
        plt.text(j, i, f'{q_values_grid[i, j]:.2f}', ha='center', va='center', color='black')

plt.show()

# Print learned Q-table
print("Learned Q-table:")
print(Q_table)

```

Output:



Q-table đã học cho thấy phần thưởng kỳ vọng cho mỗi cặp trạng thái-hành động, với các giá trị Q cao hơn gần trạng thái mục tiêu (trạng thái 15), điều này chỉ ra các hành động tối ưu dẫn đến việc đạt được mục tiêu. Hành động của tác nhân dần dần cải thiện theo thời gian, như được phản ánh qua các giá trị Q tăng dần trên các trạng thái dẫn đến mục tiêu.

3.1.6 Ưu điểm của Q-Learning:

- **Học qua thử và sai (Trial and Error Learning):** Q-Learning cải thiện theo thời gian bằng cách thử các hành động khác nhau và học từ kinh nghiệm.
- **Tự cải thiện (Self-Improvement):** Những sai lầm dẫn đến việc học hỏi, giúp tác nhân tránh lặp lại chúng.
- **Ra quyết định tốt hơn (Better Decision-Making):** Lưu trữ các hành động thành công để tránh những lựa chọn tệ trong các tình huống tương lai.
- **Học tự chủ (Autonomous Learning):** Nó học mà không cần sự giám sát bên ngoài, hoàn toàn thông qua quá trình khám phá.

3.1.7 Nhược điểm của Q-Learning:

- **Học chậm (Slow Learning):** Yêu cầu nhiều ví dụ, khiến nó tốn thời gian cho các bài toán phức tạp.
- **Tốn kém trong một số môi trường (Expensive in Some Environments):** Trong robot, việc thử nghiệm các hành động có thể tốn kém do giới hạn vật lý.
- **Lời nguyền của chiều không gian (Curse of Dimensionality):** Không gian trạng thái và hành động lớn khiến Q-table trở nên quá lớn để xử lý hiệu quả.
- **Giới hạn ở hành động rời rạc (Limited to Discrete Actions):** Nó gặp khó khăn với các hành động liên tục như điều chỉnh tốc độ, khiến nó ít phù hợp với các ứng dụng thực tế liên quan đến quyết định liên tục.

3.1.8 Ứng dụng của Q-Learning:

Q-Learning, một thuật toán học tăng cường, có thể được áp dụng trong nhiều lĩnh vực khác nhau. Dưới đây là một số ví dụ đáng chú ý:

- **Trò chơi Atari:** Các trò chơi cổ điển trên Atari 2600 giờ đây có thể được chơi bằng Q-Learning. Trong các trò chơi như *Space Invaders* và *Breakout*, Deep Q Networks (DQN) – một phiên bản mở rộng của Q-Learning sử dụng mang nơ-ron sâu – đã thể hiện hiệu suất vượt trội hơn con người.
- **Điều khiển robot:** Q-Learning được sử dụng trong robot để thực hiện các nhiệm vụ như điều hướng và kiểm soát robot. Với các thuật toán Q-Learning, robot có thể học cách di chuyển qua các môi trường, tránh chướng ngại vật và tối ưu hóa chuyển động của mình.
- **Quản lý giao thông:** Các hệ thống quản lý giao thông cho xe tự hành sử dụng Q-Learning. Nó giúp giảm ùn tắc và cải thiện luồng giao thông tổng thể bằng cách tối ưu hóa lập kế hoạch lộ trình và thời gian đèn giao thông.
- **Giao dịch thuật toán:** Việc sử dụng Q-Learning để đưa ra quyết định giao dịch đã được nghiên cứu trong lĩnh vực giao dịch thuật toán. Nó cho phép các tác nhân tự động học các chiến lược tốt nhất từ dữ liệu thị trường trong quá khứ và thích nghi với những thay đổi của điều kiện thị trường.
- **Kế hoạch điều trị cá nhân hóa:** Để tạo ra các kế hoạch điều trị độc đáo hơn, Q-Learning được ứng dụng trong lĩnh vực y học. Thông qua việc sử dụng dữ liệu bệnh nhân, các tác nhân có thể đề xuất các can thiệp cá nhân hóa, tính đến phản ứng riêng biệt của từng cá nhân đối với các phương pháp điều trị khác nhau.

3.2 Deep Q Learning

3.2.1 Khái niệm về Deep Q-Learning

- Deep Q-Learning là một biến thể của Q-Learning, trong đó một mạng nơ-ron nhân tạo (DNN - Deep Neural Network) được sử dụng để xấp xỉ giá trị Q thay vì lưu trữ tất cả các giá trị trong một bảng Q truyền thống.

- Q-Learning hoạt động dựa trên việc cập nhật bảng Q để tìm chính sách tối ưu, nhưng khi số lượng trạng thái và hành động trong môi trường tăng lên, việc lưu trữ và cập nhật bảng Q trở nên không khả thi. Deep Q-Learning giải quyết vấn đề này bằng cách thay thế bảng Q bằng một mô hình mạng nơ-ron sâu để xấp xỉ giá trị Q, giúp học được chính sách tối ưu trong không gian trạng thái lớn.

3.2.2 Cách hoạt động của Deep Q-Learning

- Deep Q-Learning hoạt động bằng cách sử dụng mạng nơ-ron sâu để xấp xỉ giá trị Q cho mỗi trạng thái và hành động. Dưới đây là các bước hoạt động chính:

- Kiến trúc cơ bản

- Deep Q-Learning sử dụng một mạng nơ-ron với:

- **Input:** Trạng thái hiện tại của môi trường.
- **Hidden Layers:** Các lớp ẩn để học đặc trưng quan trọng.
- **Output:** Giá trị Q cho tất cả các hành động có thể thực hiện từ trạng thái đầu vào.

- Các bước huấn luyện Deep Q-Learning

1. Khởi tạo môi trường và mạng nơ-ron

- Xây dựng mạng nơ-ron sâu với trọng số khởi tạo ngẫu nhiên.
- Thiết lập bộ nhớ replay buffer để lưu trữ các trải nghiệm.

2. Chọn hành động theo chính sách ϵ -greedy

- Chọn hành động ngẫu nhiên với xác suất ϵ để khám phá môi trường.
- Chọn hành động có giá trị Q cao nhất từ mạng nơ-ron với xác suất $1 - \epsilon$ để khai thác kiến thức hiện có.

3. Thực hiện hành động và lưu trữ trải nghiệm

- Thực hiện hành động a và nhận phần thưởng r cũng như trạng thái tiếp theo s' .

- Lưu bộ dữ liệu (s, a, r, s') vào bộ nhớ replay buffer.

4. Huấn luyện mạng nơ-ron bằng replay buffer

- Lấy mẫu một batch ngẫu nhiên từ replay buffer.

- Tính giá trị mục tiêu Q_{target} theo công thức:

$$Q_{target} = r + \gamma \max_{a'} Q(s', a'; \theta)$$

- Cập nhật mạng nơ-ron bằng cách tối ưu hóa hàm mất mát giữa giá trị Q dự đoán và giá trị mục tiêu Q_{target} .

5. Lặp lại quá trình cho đến khi thuật toán hội tụ

- Giảm dần giá trị ϵ để chuyển từ giai đoạn khám phá sang khai thác.

3.2.3 Ứng dụng của Deep Q-Learning

- Điều khiển robot tự động.
- Quản lý tài nguyên trong hệ thống phân tán.
- Tối ưu hóa giao thông và hệ thống khuyến nghị.

3.3 Thuật toán học tăng cường Sarsa:

- SARSA là một thuật toán on-policy được sử dụng trong học tăng cường để đào tạo mô hình quá trình quyết định Markov trên một chính sách mới. Đây là một thuật toán trong đó, ở trạng thái hiện tại (S), một hành động (A) được thực hiện và tác nhân nhận được phần thưởng (R), và kết thúc ở trạng thái tiếp theo (S_1), và thực hiện hành động (A_1) trong S_1 , hay nói cách khác, bộ S, A, R, S_1, A_1 .
- Thuật toán của SARSA hơi khác so với Q-learning .
- Trong thuật toán SARSA, giá trị Q được cập nhật có tính đến hành động A_1 được thực hiện ở trạng thái S_1 . Trong Q-learning, hành động có giá trị Q cao nhất ở trạng thái tiếp theo S_1 được sử dụng để cập nhật bảng Q .

3.3.1 Thuật toán Sarsa hoạt động như thế nào?

- Thuật toán SARSA hoạt động bằng cách thực hiện các hành động dựa trên phần thưởng nhận được từ các hành động trước đó. Để thực hiện điều này, SARSA lưu trữ một bảng các cặp ước tính trạng thái (S)-hành động (A) cho mỗi giá trị Q. Bảng này được gọi là bảng Q, trong khi các cặp trạng thái-hành động được ký hiệu là $Q(S, A)$.
- Quá trình SARSA bắt đầu bằng cách khởi tạo $Q(S, A)$ thành các giá trị tùy ý. Trong bước này, trạng thái hiện tại ban đầu (S) được thiết lập và hành động ban đầu (A) được chọn bằng cách sử dụng chính sách thuật toán epsilon-greedy dựa trên các giá trị Q hiện tại. Chính sách epsilon-greedy cân bằng việc sử dụng các phương pháp khai thác và khám phá trong quá trình học để chọn hành động có phần thưởng ước tính cao nhất.
- Khai thác bao gồm việc sử dụng các giá trị ước tính đã biết để có được nhiều phần thưởng đã kiểm được trước đó trong quá trình học. Khám phá bao gồm việc cố gắng tìm ra kiến thức mới về các hành động, có thể dẫn đến các hành động ngắn hạn, không tối ưu trong quá trình học nhưng có thể mang lại lợi ích lâu dài để tìm ra hành động và phần thưởng tốt nhất có thể.
- Từ đây, hành động đã chọn được thực hiện và phần thưởng (R) và trạng thái tiếp theo (S_1) được quan sát. $Q(S, A)$ sau đó được cập nhật và hành động tiếp theo (A_1) được chọn dựa trên các giá trị Q đã cập nhật. Ước tính giá trị hành động của trạng thái cũng được cập nhật cho mỗi cặp hành động-trạng thái hiện tại, ước tính giá trị nhận được phần thưởng khi thực hiện một hành động nhất định.
- Các bước trên của R đến A1 được lặp lại cho đến khi tập đã cho của thuật toán kết thúc, tập này mô tả trình tự các trạng thái, hành động và phần thưởng được thực hiện cho đến khi đạt đến trạng thái cuối cùng (trạng thái kết thúc). Các trải nghiệm về trạng thái, hành động và phần thưởng trong quy trình SARSA được sử dụng để cập nhật các giá trị $Q(S, A)$ cho mỗi lần lặp lại.

* Cách sử dụng Sarsa:

Bây giờ, chúng ta hãy xem mã của SARSA để giải quyết môi trường FrozenLake :

```
import gym
```

```
import numpy as np
```

```
import time, pickle, os

env = gym.make('FrozenLake-v0')
epsilon = 0.9
# min_epsilon = 0.1
# max_epsilon = 1.0
# decay_rate = 0.01
total_episodes = 10000
max_steps = 100
lr_rate = 0.81
gamma = 0.96
Q = np.zeros((env.observation_space.n, env.action_space.n))

def choose_action(state):
    action=0
    if np.random.uniform(0, 1) < epsilon:
        action = env.action_space.sample()
    else:
        action = np.argmax(Q[state, :])
    return action

def learn(state, state2, reward, action, action2):
    predict = Q[state, action]
    target = reward + gamma * Q[state2, action2]
    Q[state, action] = Q[state, action] + lr_rate * (target - predict)

# Start
rewards=0
```

```

for episode in range(total_episodes):
    t = 0
    state = env.reset()
    action = choose_action(state)
    while t < max_steps:
        env.render()
        state2, reward, done, info = env.step(action)

        action2 = choose_action(state2)
        learn(state, state2, reward, action, action2)
        state = state2
        action = action2
        t += 1
        rewards+=1
        if done:
            break
    # epsilon = min_epsilon + (max_epsilon - min_epsilon) * np.exp(-decay_rate * episode)
    # os.system('clear')
    time.sleep(0.1)
    print ("Score over time: ", rewards/total_episodes)
    print(Q)
    with open("frozenLake_qTable_sarsa.pkl", 'wb') as f:
        pickle.dump(Q, f)

```

- Bạn sẽ thấy mã này tương tự như mã được sử dụng trong Q-learning để giải quyết môi trường FrozenLake.

- Nay giờ, chúng ta hãy phân tích nó.
- Ở dòng 38 và 39, một hành động được chọn cho trạng thái ban đầu.

Nay giờ SARSA, bây giờ chúng ta có:

(State, Action)

- Sau đó, hành động này được thực hiện trong môi trường và phần thưởng cùng trạng thái tiếp theo được quan sát ở dòng 44.

Bây giờ, bộ này có:

(State, Action, Reward, State1)

- Ở dòng 46, hành động được chọn cho trạng thái tiếp theo bằng cách sử dụng `choose_state(...)` hàm.
- Hành động được chọn bởi `choose_action(...)` hàm được thực hiện bằng cách sử dụng phương pháp epsilon-greedy.

Bây giờ, bộ dữ liệu hoàn chỉnh như sau:

(State, Action, Reward, State1, Action1)

Ở dòng 48, `learn(...)` hàm cập nhật bảng Q bằng phương trình sau:

$$Q(S, A) \leftarrow Q(S, A) + \alpha(R + \gamma Q(S', A') - Q(S, A))$$

Trong phương trình cập nhật SARSA, giá trị Q được chọn bằng cách sử dụng S' và A' , trạng thái tiếp theo và hành động được chọn trong trạng thái tiếp theo. Điều này trái ngược với Q-learning, cập nhật phương trình trong đó giá trị tối đa của $Q(S', a)$ được lấy.

Phần còn lại của mã tương tự như mã Q-learning.

* SARSA có tốt hơn Q-learning không?

Cả SARSA và Q-learning đều hữu ích cho việc học tăng cường và mỗi phương pháp đều có cách sử dụng riêng dựa trên nhu cầu của tác nhân. SARSA có thể tốt hơn như một thuật toán không xác định và để lựa chọn các đường dẫn hành động có rủi ro thấp so với Q-learning.

3.4 Tìm kiếm cây Monte carlo (MCTS)

- Monte Carlo Tree Search (MCTS) là một bộ quy tắc tìm kiếm theo phương pháp heuristic đã giành được sự chú ý và danh tiếng lớn trong lĩnh vực trí tuệ tổng hợp, đặc biệt là trong lĩnh vực ra quyết định và chơi trò chơi. Nó được biết đến với khả năng xử lý hiệu quả các trò chơi điện tử phức tạp và mang tính chiến lược với các khu vực tìm kiếm khổng lồ, trong đó các thuật toán truyền thống có thể gặp khó khăn do số lượng hành động hoặc hành động khả thi đầy đủ.
- MCTS kết hợp các tiêu chuẩn của chiến lược Monte Carlo, dựa trên lấy mẫu ngẫu nhiên và đánh giá thống kê, với các kỹ thuật tìm kiếm chủ yếu dựa trên cây. Không giống như các thuật toán tìm kiếm truyền thống dựa trên việc khám phá toàn bộ khu vực tìm kiếm, MCTS chuyên về việc lấy mẫu và chỉ khám phá các khu vực có triển vọng của khu vực tìm kiếm.
- Ý tưởng cốt lõi ở mặt sau của MCTS là xây dựng một cây tìm kiếm theo từng bước bằng cách sử dụng mô phỏng nhiều hơn một lần thực hiện ngẫu nhiên (thường được gọi là rollouts hoặc playouts) từ quốc gia giải trí hiện tại. Các mô phỏng này được thực hiện cho đến khi đạt đến trạng thái cuối cùng hoặc cường độ được xác định trước. Sau đó, kết quả của các mô phỏng này được truyền ngược lên cây, cập nhật các bản ghi của các nút đã truy cập tại một số giai đoạn trong trò chơi, bao gồm nhiều loại lượt truy cập và tỷ lệ thắng.
- Khi quá trình tìm kiếm tiến triển, MCTS cân bằng động giữa việc khám phá và khai thác. Nó chọn các nước đi thông qua việc cân nhắc cả việc khai thác các nước đi có triển vọng đáng chú ý với tỷ lệ thắng cao và việc khám phá các nước đi chưa được khám phá hoặc ít được khám phá. Sự cân bằng này được hoàn thành thông qua việc sử dụng các thành phần chắc chắn về độ tin cậy cao nhất (UCB), bao gồm Giới hạn độ tin cậy cao hơn cho cây (UCT), để quyết định nước đi hoặc nút nào sẽ ghé thăm trong suốt thời gian săn tìm.

- MCTS đã được triển khai hiệu quả trong nhiều lĩnh vực, bao gồm trò chơi cờ bàn (eG, Go, cờ vua và shogi), trò chơi điện tử bài (eG, poker) và trò chơi điện tử. Nó đã đạt được hiệu suất tổng thể tuyệt vời trong nhiều tình huống đánh bạc giải trí đầy thử thách, thường vượt qua khả năng hiểu biết của con người. MCTS cũng đã được kéo dài và điều chỉnh để giải quyết các lĩnh vực rắc rối khác nhau, bao gồm lập kế hoạch, lập lịch trình và tối ưu hóa.
- Một trong những lợi ích tuyệt vời của MCTS là khả năng xử lý trò chơi điện tử với dữ liệu không xác định hoặc không hoàn hảo, vì nó dựa vào mẫu thống kê trái ngược với toàn bộ kiến thức về trạng thái trò chơi. Ngoài ra, MCTS có khả năng mở rộng và có thể song song hóa hiệu quả, khiến nó phù hợp với điện toán phân tán và kiến trúc đa lõi.
- Monte Carlo Tree Search (MCTS) là một kỹ thuật tìm kiếm trong lĩnh vực Trí tuệ nhân tạo (AI). Đây là một thuật toán tìm kiếm theo xác suất và theo phương pháp tìm kiếm kết hợp các triển khai tìm kiếm cây cổ điển cùng với các nguyên tắc học máy của học tăng cường.
- Trong tìm kiếm cây, luôn có khả năng hành động tốt nhất hiện tại thực sự không phải là hành động tối ưu nhất. Trong những trường hợp như vậy, thuật toán MCTS trở nên hữu ích vì nó tiếp tục đánh giá các phương án thay thế khác theo định kỳ trong giai đoạn học bằng cách thực hiện chúng, thay vì chiến lược tối ưu hiện tại được nhận thức. Điều này được gọi là "sự đánh đổi giữa khám phá và khai thác". Nó khai thác các hành động và chiến lược được tìm thấy là tốt nhất cho đến nay nhưng cũng phải tiếp tục khám phá không gian cục bộ của các quyết định thay thế và tìm hiểu xem chúng có thể thay thế phương án tốt nhất hiện tại hay không.
- Khám phá giúp khám phá và tìm ra các phần chưa được khám phá của cây, điều này có thể dẫn đến việc tìm ra một con đường tối ưu hơn. Nói cách khác, chúng ta có thể nói rằng khám phá mở rộng chiều rộng của cây hơn là chiều sâu của nó. Khám phá có thể hữu ích để đảm bảo rằng MCTS không bỏ qua bất kỳ con đường nào có khả năng tốt hơn. Nhưng nó nhanh chóng trở nên kém hiệu quả trong các tình huống có nhiều bước hoặc lặp lại. Để tránh điều đó, nó được cân bằng bằng khai thác. Khai thác gắn liền với một đường dẫn duy nhất có giá trị ước tính lớn nhất. Đây là một cách tiếp cận tham lam và điều này sẽ mở rộng chiều sâu của cây nhiều hơn chiều rộng của nó. Nói một

cách đơn giản, công thức UCB áp dụng cho cây giúp cân bằng sự đánh đổi giữa khám phá-khai thác bằng cách khám phá định kỳ các nút tương đối chưa được khám phá của cây và khám phá ra các đường dẫn có khả năng tối ưu hơn so với đường dẫn mà nó hiện đang khai thác.

- Đối với đặc điểm này, MCTS trở nên đặc biệt hữu ích trong việc đưa ra quyết định tối ưu trong các vấn đề Trí tuệ nhân tạo (AI).

*Tại sao nên sử dụng Monte Carlo Tree Search (MCTS)?

Sau đây là một số lý do tại sao MCTS được sử dụng phổ biến:

- Xử lý các trò chơi phức tạp và chiến lược: MCTS nổi trội trong các trò chơi có không gian tìm kiếm lớn, động lực phức tạp và ra quyết định chiến lược. Nó đã được áp dụng thành công vào các trò chơi như Cờ vây, cờ vua, cờ tướng, poker và nhiều trò chơi khác, đạt được hiệu suất đáng chú ý thường vượt qua trình độ của con người. MCTS có thể khám phá và đánh giá hiệu quả các nước đi hoặc hành động khác nhau, dẫn đến lối chơi và ra quyết định mạnh mẽ trong các trò chơi như vậy.
- Thông tin không xác định hoặc không hoàn hảo: MCTS phù hợp với các trò chơi hoặc tình huống có thông tin không xác định hoặc không hoàn hảo. Nó dựa vào mẫu thống kê và không yêu cầu kiến thức đầy đủ về trạng thái trò chơi. Điều này làm cho MCTS có thể áp dụng cho các lĩnh vực có thông tin không chắc chắn hoặc không đầy đủ, chẳng hạn như trò chơi bài hoặc các tình huống thực tế với dữ liệu hạn chế hoặc không đáng tin cậy.
- Học từ mô phỏng: MCTS học từ mô phỏng hoặc triển khai để ước tính giá trị của hành động hoặc trạng thái. Thông qua các lần lặp lại, MCTS dần tinh chỉnh kiến thức và cải thiện việc ra quyết định. Khía cạnh học tập này giúp MCTS thích ứng và có khả năng thích ứng với các hoàn cảnh thay đổi hoặc các chiến lược đang phát triển.
- Tối ưu hóa việc khám phá và khai thác: MCTS cân bằng hiệu quả giữa việc khám phá và khai thác trong quá trình tìm kiếm. Nó khám phá một cách thông minh các khu vực chưa được khám phá của không gian tìm kiếm trong khi khai thác các hành động đầy hứa hẹn dựa trên kiến thức hiện có. Sự đánh đổi giữa khám phá và khai thác này cho phép MCTS tìm được sự cân bằng giữa việc khám phá các khả năng mới và khai thác các hành động tốt đã biết.
- Khả năng mở rộng và song song hóa: MCTS có khả năng mở rộng và có thể song song hóa hiệu quả. Nó có thể sử dụng các tài nguyên điện toán phân tán hoặc kiến trúc đa lõi để tăng tốc tìm kiếm và xử lý các không gian tìm kiếm

lớn hơn. Khả năng mở rộng này giúp MCTS có thể áp dụng cho các vấn đề đòi hỏi nhiều tài nguyên điện toán.

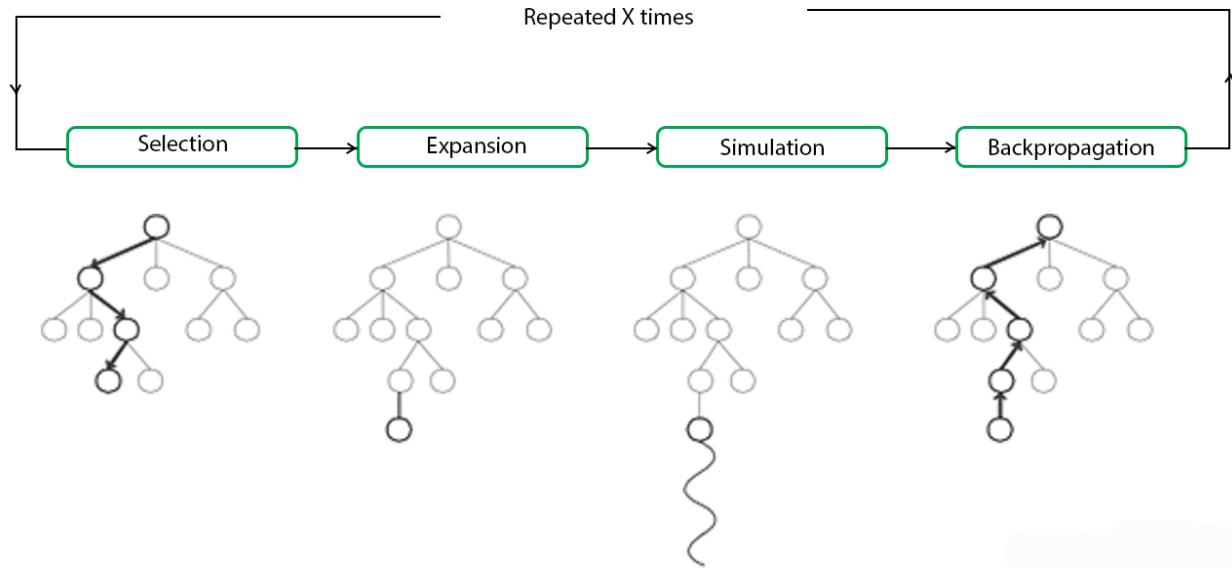
- **Khả năng áp dụng ngoài trò chơi:** Trong khi MCTS nổi bật trong các lĩnh vực chơi trò chơi, các nguyên tắc và kỹ thuật của nó cũng có thể áp dụng cho các lĩnh vực vấn đề khác. MCTS đã được áp dụng thành công vào các vấn đề lập kế hoạch, lập lịch, tối ưu hóa và ra quyết định trong nhiều tình huống thực tế khác nhau. **Khả năng xử lý việc ra quyết định phức tạp và sự không chắc chắn** của nó khiến nó trở nên có giá trị trong nhiều ứng dụng.
- **Độc lập miền:** MCTS tương đối độc lập với miền. Nó không yêu cầu kiến thức hoặc phương pháp tìm kiếm chuyên biệt cho miền để hoạt động. Mặc dù có thể thực hiện các cải tiến chuyên biệt cho miền để cải thiện hiệu suất, thuật toán MCTS cơ bản có thể được áp dụng cho nhiều miền vấn đề mà không cần sửa đổi đáng kể.
- Trong MCTS, các nút là các khối xây dựng của cây tìm kiếm. Các nút này được hình thành dựa trên kết quả của một số mô phỏng. Quá trình Monte Carlo Tree Search có thể được chia thành bốn bước riêng biệt, cụ thể là lựa chọn, mở rộng, mô phỏng và lan truyền ngược. Mỗi bước trong số các bước này được giải thích chi tiết bên dưới:
- **Lựa chọn:** Trong quá trình này, thuật toán MCTS duyệt cây hiện tại từ nút gốc bằng một chiến lược cụ thể. Chiến lược này sử dụng một hàm đánh giá để lựa chọn tối ưu các nút có giá trị ước tính cao nhất. MCTS sử dụng công thức Giới hạn tin cậy trên (UCB) được áp dụng cho các cây làm chiến lược trong quá trình lựa chọn để duyệt cây. Nó cân bằng sự đánh đổi giữa khám phá và khai thác. Trong quá trình duyệt cây, một nút được chọn dựa trên một số tham số trả về giá trị tối đa. Các tham số được đặc trưng bởi công thức thường được sử dụng cho mục đích này được đưa ra bên dưới.

$$S_i = x_i + C \sqrt{\frac{\ln(t)}{n_i}}$$

trong đó;

- S_i = giá trị của một nút i
- x_i = giá trị trung bình thực nghiệm của một nút i
- C = hằng số
- t = tổng số mô phỏng

- Khi duyệt một cây trong quá trình lựa chọn, nút con trả về giá trị lớn nhất từ phương trình trên sẽ là nút được chọn. Trong quá trình duyệt, khi tìm thấy một nút con cũng là một nút lá, MCTS sẽ nhảy vào bước mở rộng.
- Mở rộng: Trong quá trình này, một nút con mới được thêm vào cây tại nút đã đạt được mức tối ưu trong quá trình lựa chọn.
- Mô phỏng: Trong quá trình này, mô phỏng được thực hiện bằng cách chọn các bước đi hoặc chiến lược cho đến khi đạt được kết quả hoặc trạng thái được xác định trước.
- Truyền ngược: Sau khi xác định giá trị của nút mới được thêm vào, cây còn lại phải được cập nhật. Vì vậy, quá trình truyền ngược được thực hiện, trong đó nó truyền ngược từ nút mới đến nút gốc. Trong quá trình này, số lượng mô phỏng được lưu trữ trong mỗi nút được tăng lên. Ngoài ra, nếu mô phỏng của nút mới dẫn đến chiến thắng, thì số lượng chiến thắng cũng được tăng lên.
- Các bước trên có thể được hiểu trực quan qua sơ đồ dưới đây:



Mã giả cho tìm kiếm Cây Monte Carlo:

```
# main function for the Monte Carlo Tree Search
def monte_carlo_tree_search(root):
```

```

while resources_left(time, computational power):
    leaf = traverse(root)
    simulation_result = rollout(leaf)
    backpropagate(leaf, simulation_result)

return best_child(root)

# function for node traversal
def traverse(node):
    while fully_expanded(node):
        node = best_uct(node)

    # in case no children are present / node is terminal
    return pick_unvisited(node.children) or node

# function for the result of the simulation
def rollout(node):
    while non_terminal(node):
        node = rollout_policy(node)
    return result(node)

# function for randomly selecting a child node
def rollout_policy(node):
    return pick_random(node.children)

# function for backpropagation
def backpropagate(node, result):
    if is_root(node) return
    node.stats = update_stats(node, result)
    backpropagate(node.parent)

# function for selecting the best child
# node with highest number of visits

```

```
def best_child(node):  
    pick child with highest number of visits
```

- Như chúng ta có thể thấy, thuật toán MCTS thu gọn thành một tập hợp rất ít hàm mà chúng ta có thể sử dụng bất kỳ lựa chọn trò chơi nào hoặc trong bất kỳ chiến lược tối ưu hóa nào.

*Ưu điểm của Tìm kiếm cây Monte Carlo:

- MCTS là một thuật toán đơn giản để triển khai.
- Monte Carlo Tree Search là một thuật toán tìm kiếm theo phương pháp heuristic. MCTS có thể hoạt động hiệu quả mà không cần bất kỳ kiến thức nào trong phạm vi cụ thể, ngoài các quy tắc và điều kiện kết thúc, và có thể tự tìm ra các nước đi của riêng mình và học hỏi từ chúng bằng cách chơi ngẫu nhiên.
- MCTS có thể được lưu ở bất kỳ trạng thái trung gian nào và trạng thái đó có thể được sử dụng trong các trường hợp sử dụng trong tương lai bất cứ khi nào cần.
- MCTS hỗ trợ việc mở rộng không đối xứng của cây tìm kiếm dựa trên các trường hợp mà nó đang hoạt động.

*Nhược điểm của Tìm kiếm cây Monte Carlo:

- Vì cây phát triển nhanh sau một vài lần lặp nên cần một lượng bộ nhớ lớn.
- Có một chút vấn đề về độ tin cậy với Monte Carlo Tree Search. Trong một số trường hợp nhất định, có thể có một nhánh hoặc đường dẫn duy nhất, có thể dẫn đến thua trước đối thủ khi được triển khai cho các trò chơi theo lượt đó. Điều này chủ yếu là do số lượng lớn các kết hợp và mỗi nút có thể không được truy cập đủ số lần để hiểu kết quả hoặc kết cục của nó trong thời gian dài.
- Thuật toán MCTS cần một số lượng lớn các lần lặp lại để có thể quyết định hiệu quả đường dẫn hiệu quả nhất. Vì vậy, có một chút vấn đề về tốc độ ở đây.

*Các vấn đề trong Tìm kiếm cây Monte Carlo:

*Sau đây là một số vấn đề phổ biến liên quan đến MCTS:

- Đánh đổi giữa khai thác và thăm dò: MCTS phải đổi mới với thách thức cân bằng giữa khai thác và thăm dò trong quá trình tìm kiếm. Nó cần phải khám phá các nhánh khác nhau của cây tìm kiếm để thu thập thông tin về tiềm năng

của chúng, đồng thời khai thác các hành động đầy hứa hẹn dựa trên kiến thức hiện có. Đạt được sự cân bằng phù hợp là rất quan trọng đối với hiệu quả và hiệu suất của thuật toán.

- **Hiệu quả mẫu:** MCTS có thể yêu cầu một số lượng lớn các mô phỏng hoặc triển khai để có được số liệu thống kê chính xác và đưa ra quyết định sáng suốt. Điều này có thể tốn kém về mặt tính toán, đặc biệt là trong các miền phức tạp với không gian tìm kiếm lớn. Cải thiện hiệu quả mẫu của MCTS là một lĩnh vực nghiên cứu đang được tiến hành.
- **Độ biến thiên cao:** Kết quả của từng lần triển khai trong MCTS có thể rất khác nhau do bản chất ngẫu nhiên của các mô phỏng. Điều này có thể dẫn đến ước tính không nhất quán về giá trị hành động và gây nhiễu trong quá trình ra quyết định. Các kỹ thuật như giảm độ biến thiên và mở rộng dần dần được sử dụng để giảm thiểu vấn đề này.
- **Thiết kế Heuristic:** MCTS dựa vào heuristic để hướng dẫn tìm kiếm và ưu tiên các hành động hoặc nút. Thiết kế heuristic hiệu quả và cụ thể cho từng miền có thể là một thách thức và chất lượng của heuristic ảnh hưởng trực tiếp đến hiệu suất của thuật toán. Phát triển heuristic chính xác nắm bắt được các đặc điểm của miền vấn đề là một khía cạnh quan trọng khi sử dụng MCTS.
- **Yêu cầu về tính toán và bộ nhớ:** MCTS có thể đòi hỏi tính toán chuyên sâu, đặc biệt là trong các trò chơi có đường chân trời dài hoặc động lực phức tạp. Hiệu suất của thuật toán phụ thuộc vào các tài nguyên tính toán có sẵn và trong các môi trường hạn chế về tài nguyên, có thể không khả thi khi chạy MCTS với số lượng mô phỏng đủ lớn. Ngoài ra, MCTS yêu cầu bộ nhớ để lưu trữ và cập nhật cây tìm kiếm, điều này có thể trở thành hạn chế trong các tình huống hạn chế về bộ nhớ.
- **Quá khớp:** Trong một số trường hợp, MCTS có thể quá khớp với các mẫu hoặc độ lệch cụ thể có trong các mô phỏng ban đầu, điều này có thể dẫn đến các quyết định không tối ưu. Để giảm thiểu vấn đề này, các kỹ thuật như tiền thưởng khám phá, bỏ tỉa tiến bộ và ước tính giá trị hành động nhanh đã được đề xuất để khuyến khích khám phá và tránh hội tụ sớm.
- **Thách thức cụ thể theo miền:** Các miền và loại vấn đề khác nhau có thể đưa ra những thách thức và vấn đề bổ sung cho MCTS. Ví dụ, các trò chơi có thông tin ẩn hoặc không hoàn hảo, các yếu tố phân nhánh lớn hoặc không gian hành động liên tục đòi hỏi phải điều chỉnh và mở rộng thuật toán MCTS cơ bản để xử lý các phức tạp này một cách hiệu quả.

CHƯƠNG 4: CÁC PHƯƠNG PHÁP GIẢI THÍCH TRONG HỌC TĂNG CƯỜNG

4.1 VIPER

4.1.1 Giới thiệu

- Thuật toán VIPER: thuật toán VIPER được xây dựng dựa trên thuật toán DAgger (Dataset Aggregation), nhưng thay vì chỉ huấn luyện theo dữ liệu, VIPER tập trung vào những trạng thái quan trọng ảnh hưởng lớn đến hiệu suất.

4.1.2 Quy trình

- **Huấn luyện tác nhân chuyên gia (expert policy):** Dùng mô hình mạnh (như DQN, PPO) để đạt hiệu suất cao.
- **Thu thập dữ liệu từ chuyên gia:** Chạy chuyên gia qua nhiều trạng thái để thu thập trạng thái - hành động.
- **Tính toán tầm quan trọng của trạng thái:** Đánh giá trạng thái nào quan trọng đối với kết quả cuối cùng.
- **Huấn luyện cây quyết định:** Tập trung huấn luyện cây quyết định từ dữ liệu quan trọng.
- **Lặp lại (nếu cần):** Nếu cây chưa đạt yêu cầu → tiếp tục thu thập dữ liệu & huấn luyện lại.

4.1.3 Đánh giá

- **Ưu điểm của VIPER:**
 - Chính sách dễ hiểu: Trích xuất cây quyết định rõ ràng.
 - Hiệu suất cao: Giữ hiệu suất gần với mô hình gốc.
 - Có thể kiểm chứng: Dễ kiểm tra tính an toàn trong môi trường nhạy cảm.

4.1.4 Ứng dụng thực tế

- **Ngành tài chính – Phê duyệt khoản vay:** giải thích các quyết định phê duyệt hoặc từ chối khoản vay bằng các quy tắc đơn giản.

Quy tắc hoạt động:

- **Nếu** thu nhập \geq 20 triệu:
 - **Nếu** điểm tín dụng $\geq 750 \rightarrow Phê duyệt khoản vay.$
 - **Ngược lại** \rightarrow Từ chối khoản vay.
- **Nếu** thu nhập $<$ 20 triệu:
 - **Nếu** không có nợ xấu trong 2 năm $\rightarrow Xem xét thêm.$
 - **Ngược lại** \rightarrow Từ chối khoản vay.

Tình huống áp dụng:

- Thu nhập: **22 triệu/tháng.**
- Điểm tín dụng: **720.**
- Nợ xấu: **Không có.**

\rightarrow **Quyết định: Xem xét thêm** (thu nhập đủ nhưng điểm tín dụng chưa đạt yêu cầu).

- **Ngành y tế – Chẩn đoán bệnh:** giải thích quyết định của AI trong chẩn đoán bệnh từ dữ liệu y tế.

Quy tắc hoạt động:

- **Nếu** đường huyết $> 150 \text{ mg/dL}:
 - **Nếu** BMI $> 30 \rightarrow Nguy cơ tiểu đường cao.$
 - **Ngược lại** \rightarrow Nguy cơ trung bình.$

- **Nếu** đường huyết ≤ 150 mg/dL:
 - **Nếu** tiền sử gia đình có bệnh \rightarrow **Nguy cơ thấp**.
 - **Ngược lại** \rightarrow **Không có nguy cơ**.

Tình huống áp dụng:

- Đường huyết: **160 mg/dL**.
- BMI: **32**.
- Tiền sử gia đình: **Có**.

\rightarrow **Quyết định: Nguy cơ tiểu đường cao** (đường huyết cao và BMI lớn).

- **Thương mại điện tử – Gợi ý sản phẩm:** giải thích lý do hệ thống đề xuất sản phẩm cho người dùng.

Quy tắc hoạt động:

- **Nếu** người dùng đã mua sản phẩm cùng loại trong 30 ngày:
 - **Nếu** xếp hạng ≥ 4 sao \rightarrow **Gợi ý sản phẩm tương tự**.
 - **Ngược lại** \rightarrow **Không gợi ý**.
- **Nếu** chưa mua sản phẩm:
 - **Nếu** người dùng có hành vi giống khách khác \rightarrow **Gợi ý sản phẩm**.
 - **Ngược lại** \rightarrow **Không gợi ý**.

Tình huống áp dụng:

- Đã mua **điện thoại** trong 20 ngày.
- Đánh giá: **4.8 sao**.
- Hành vi khách hàng khác: **Mua ốp lưng**.

→ **Quyết định: Gợi ý ốp lưng** (vì sản phẩm liên quan và đánh giá cao).

4.2 SHAP phân tích tầm quan trọng đặc trưng

4.2.1 Giới thiệu

-SHAP là một phương pháp trong lĩnh vực trí tuệ nhân tạo giải thích được (Explainable AI - XAI), nhằm cung cấp giải thích minh bạch và công bằng cho dự đoán của các mô hình học máy phức tạp. Được phát triển bởi Scott M. Lundberg và các đồng tác giả, SHAP ra đời để giải quyết vấn đề "hộp đen" của các mô hình như mạng nơ-ron sâu hoặc rừng ngẫu nhiên, vốn khó hiểu đối với người dùng. Phương pháp này dựa trên lý thuyết trò chơi, cụ thể là giá trị Shapley, để phân bổ tầm quan trọng của từng đặc trưng trong dự đoán, từ đó tăng cường độ tin cậy và khả năng ứng dụng của AI trong các lĩnh vực quan trọng như y tế, tài chính.

4.2.2 Nguyên lý hoạt động

-SHAP hoạt động dựa trên khái niệm giá trị Shapley từ kinh tế học, đo lường đóng góp trung bình của mỗi đặc trưng đối với dự đoán của mô hình khi xem xét tất cả các tổ hợp đặc trưng có thể có. Nguyên lý cơ bản bao gồm:

- **Bước 1:** Xác định một mẫu dữ liệu cần giải thích và giá trị dự đoán của mô hình trên mẫu đó.
- **Bước 2:** Tính toán sự khác biệt giữa dự đoán thực tế và giá trị trung bình của tất cả dự đoán (giá trị cơ sở).
- **Bước 3:** Đánh giá đóng góp của từng đặc trưng bằng cách xem xét mọi tổ hợp có hoặc không có đặc trưng đó, sau đó gán giá trị SHAP cho mỗi đặc trưng sao cho tổng các giá trị SHAP bằng đầu ra của mô hình.

- **Tính toán hiệu quả:** Để giảm độ phức tạp, SHAP cung cấp các biến thể như Kernel SHAP (dùng hồi quy để xấp xỉ) và Tree SHAP (tối ưu cho mô hình cây). Phương pháp này đảm bảo tính công bằng (mỗi đặc trưng được đánh giá dựa trên đóng góp thực tế) và tính nhất quán (nếu một đặc trưng tăng đóng góp, giá trị SHAP của nó không giảm).

4.2.3 Đánh giá thuật toán

-SHAP được đánh giá là một trong những phương pháp giải thích mạnh mẽ nhất nhờ nền tảng lý thuyết chặt chẽ và tính linh hoạt.

- **Ưu điểm:**
 - Cung cấp cả giải thích cục bộ (cho từng dự đoán) và toàn cục (cho toàn bộ mô hình).
 - Đảm bảo tính công bằng và nhất quán, vượt trội hơn các phương pháp như LIME về độ chính xác lý thuyết.
 - Áp dụng được cho mọi loại mô hình học máy, từ cây quyết định đến mạng nơ-ron sâu.
- **Nhược điểm:**
 - Chi phí tính toán cao, đặc biệt với Kernel SHAP, do cần xem xét nhiều tổ hợp đặc trưng.
 - Yêu cầu tài nguyên lớn hơn so với các phương pháp đơn giản như LIME, khiến việc áp dụng trên tập dữ liệu lớn gặp khó khăn.

4.2.4 Ứng dụng thực tiễn

-SHAP đã được áp dụng rộng rãi trong nhiều lĩnh vực thực tế:

- **Y học:** Phân tích dự đoán nguy cơ bệnh (ví dụ: bệnh thận) bằng cách chỉ ra các yếu tố quan trọng như huyết áp hoặc mức creatinine, giúp bác sĩ đưa ra quyết định chính xác hơn.
- **Tài chính:** Đánh giá rủi ro tín dụng, xác định các yếu tố như thu nhập, lịch sử vay ảnh hưởng đến chấp thuận khoản vay, tăng tính minh bạch cho khách hàng.
- **Học tăng cường (Reinforcement Learning):** Giải thích chính sách của tác nhân RL, ví dụ: trong trò chơi hoặc robot, SHAP có thể chỉ ra trạng thái nào (state) quyết định hành động cụ thể.

- **Công nghiệp:** Phát hiện lỗi mô hình hoặc cải thiện hệ thống tự động bằng cách phân tích đặc trưng quan trọng trong dữ liệu sản xuất. SHAP đặc biệt hữu ích trong các ứng dụng yêu cầu giải thích chi tiết và tuân thủ quy định pháp lý.

4.3 LIME giải thích cục bộ dễ hiểu

4.3.1 Giới thiệu về LIME

-LIME (Local Interpretable Model-agnostic Explanations) là một phương pháp giải thích trong máy học, được đề xuất bởi Ribeiro et al. (2016), nhằm làm sáng tỏ các mô hình phức tạp bằng cách tạo ra giải thích cục bộ dễ hiểu. Trong học tăng cường (RL), LIME giúp giải thích tại sao một tác nhân chọn một hành động cụ thể trong một trạng thái nhất định, đặc biệt khi tác nhân dựa trên các mô hình black-box như mạng nơ-ron sâu (DNN).

4.3.2 Nguyên lý hoạt động của LIME

- **Xác định điểm dữ liệu cần giải thích**
- Nguyên lý hoạt động của LIME bắt đầu bằng việc chọn một trường hợp cụ thể cần giải thích. Trong RL, "điểm dữ liệu" này thường là một **trạng thái (state)** mà tác nhân gặp phải trong quá trình tương tác với môi trường. Trạng thái này có thể được biểu diễn dưới dạng một vector các đặc trưng (features), chẳng hạn như vị trí, tốc độ, hoặc các yếu tố môi trường khác.

Ví dụ: Giả sử bạn có một tác nhân RL điều khiển một robot di chuyển trong mê cung. Trạng thái s có thể là [x = 5, y = 3, distance_to_wall = 2, angle = 45°], và tác nhân chọn hành động "rẽ phải". LIME sẽ tập trung giải thích tại sao "rẽ phải" được chọn trong trạng thái này.

Ý nghĩa: Việc chọn một trạng thái cụ thể là bước đầu tiên để đảm bảo giải thích mang tính **cục bộ (local)**, tức là chỉ áp dụng cho trường hợp đang xét, không phải toàn bộ chính sách của tác nhân.

Tạo tập hợp các mẫu giả (perturbed samples)

- Sau khi xác định trạng thái gốc, LIME tạo ra một tập hợp các trạng thái giả bằng cách thay đổi nhỏ (perturb) các đặc trưng của trạng thái gốc. Các trạng thái giả này không nhất thiết phải là các trạng thái thực tế trong môi trường RL, mà chỉ là các biến thể được tạo ra để mô phỏng sự thay đổi trong không gian đặc trưng.

Cách thực hiện:

- Mỗi đặc trưng trong trạng thái gốc được thêm nhiễu ngẫu nhiên (random noise) theo phân phối nhất định (thường là phân phối Gaussian).
- Số lượng mẫu giả (thường ký hiệu là N) có thể được điều chỉnh tùy theo yêu cầu, ví dụ N = 5000.

Ví dụ: Từ trạng thái gốc [x = 5, y = 3, distance_to_wall = 2, angle = 45°], LIME có thể tạo ra các trạng thái giả như:

- [x = 5.1, y = 2.9, distance_to_wall = 1.8, angle = 47°]
- [x = 4.8, y = 3.2, distance_to_wall = 2.1, angle = 43°]

Mục đích: Các mẫu giả này giúp LIME khám phá cách hành vi của mô hình RL thay đổi khi các đặc trưng thay đổi, từ đó hiểu được tầm quan trọng của từng đặc trưng.

Gán trọng số cho các mẫu giả

Không phải mọi mẫu giả đều có giá trị như nhau trong việc giải thích trạng thái gốc. LIME gán trọng số (weights) cho từng mẫu giả dựa trên mức độ gần gũi của chúng với trạng thái gốc, thường sử dụng một hàm kernel (kernel function).

Công thức trọng số:

- Trọng số w_i của mẫu giả s' được tính bằng:
$$w_i = \exp(-\text{Distance}(s, s')^2 / \sigma^2)$$
Trong đó:
 - $\text{Distance}(s, s')$: Khoảng cách Euclidean giữa trạng thái gốc s và trạng thái giả s' .

- σ : Tham số điều chỉnh độ rộng của kernel (thường được chọn dựa trên thử nghiệm).

Ý nghĩa:

- Các mẫu giả gần trạng thái gốc (khoảng cách nhỏ) sẽ có trọng số cao hơn, vì chúng phản ánh tốt hơn bối cảnh cục bộ của trạng thái gốc.
- Các mẫu xa hơn có trọng số thấp, ít ảnh hưởng đến giải thích.

Ví dụ: Nếu s là $[5, 3, 2, 45^\circ]$ và s' là $[5.1, 2.9, 1.8, 47^\circ]$ với khoảng cách nhỏ, w_i có thể là 0.95. Ngược lại, một mẫu xa hơn như $[6, 4, 3, 60^\circ]$ có thể chỉ nhận $w_i = 0.1$.

Dự đoán hành động từ mô hình RL cho các mẫu giả

LIME sử dụng chính sách của tác nhân RL (ký hiệu là π) để dự đoán hành động cho từng mẫu giả. Điều này yêu cầu mô hình RL phải được truy cập như một "black-box", tức là LIME không cần biết cấu trúc bên trong của mô hình, chỉ cần đầu ra (hành động hoặc giá trị).

Quy trình:

- Đưa từng trạng thái giả s' vào chính sách π .
- Ghi lại hành động a' mà π trả về cho s' .

Ví dụ: Với trạng thái giả $[5.1, 2.9, 1.8, 47^\circ]$, mô hình RL có thể dự đoán "rẽ phải", trong khi với $[4.8, 3.2, 2.1, 43^\circ]$, nó có thể dự đoán "đi thẳng".

Mục đích: Bước này giúp LIME thu thập dữ liệu về cách hành vi của tác nhân thay đổi khi trạng thái thay đổi, từ đó xác định mối quan hệ giữa đặc trưng và quyết định.

Huấn luyện mô hình đơn giản để giải thích

- Dựa trên tập hợp các mẫu giả (s', a', w_i), LIME huấn luyện một mô hình đơn giản (interpretable model), thường là hồi quy tuyến tính hoặc cây quyết định nhỏ, để xấp xỉ hành vi của mô hình RL quanh trạng thái gốc.

Quy trình:

- Dùng các mẫu giả làm dữ liệu huấn luyện, với trọng số w_i để ưu tiên các mẫu gần trạng thái gốc.
- Mô hình đơn giản học cách ánh xạ từ các đặc trưng của trạng thái giả (s') sang hành động dự đoán (a').

Ví dụ: Một mô hình hồi quy tuyến tính có thể cho ra công thức:

$$\text{Score(rẽ phải)} = 0.7 * \text{distance_to_wall} + 0.3 * \text{angle} - 0.1 * x + 0.05 * y$$

Trong đó, các hệ số (0.7, 0.3, v.v.) thể hiện mức độ ảnh hưởng của từng đặc trưng.

Ý nghĩa: Mô hình đơn giản này dễ hiểu hơn nhiều so với mạng nơ-ron sâu, cho phép con người phân tích tầm quan trọng của đặc trưng.

Trích xuất giải thích từ mô hình đơn giản

- Cuối cùng, LIME trích xuất giải thích bằng cách phân tích mô hình đơn giản để xác định các đặc trưng quan trọng nhất ảnh hưởng đến quyết định của tác nhân trong trạng thái gốc.

Cách thực hiện:

- Lấy các đặc trưng có hệ số lớn nhất (trong hồi quy tuyến tính) hoặc các nhánh quan trọng (trong cây quyết định).
- Trình bày dưới dạng danh sách hoặc biểu đồ.

Ví dụ: Với trạng thái [5, 3, 2, 45°] và hành động "rẽ phải", LIME có thể kết luận:

- "Khoảng cách đến tường (distance_to_wall)" đóng góp 70%.
- "Góc quay (angle)" đóng góp 20%.
- Các yếu tố khác (x, y) ít quan trọng hơn.

Kết quả: Giải thích này cho thấy "khoảng cách đến tường" là lý do chính khiến robot rẽ phải, phù hợp với trực giác con người.

Tính linh hoạt và mở rộng trong RL

- Nguyên lý của LIME rất linh hoạt và có thể được điều chỉnh cho các tình huống khác nhau trong RL:

Không gian trạng thái lớn: Có thể giảm số lượng mẫu giả hoặc dùng kỹ thuật lấy mẫu thông minh (smart sampling) để tăng hiệu quả.

Hành động liên tục: LIME có thể được mở rộng để giải thích giá trị liên tục (continuous actions) bằng cách thay đổi hàm mất mát trong mô hình đơn giản.

Tích hợp với hình ảnh: Nếu trạng thái là hình ảnh (ví dụ: trong RL chơi game Atari), LIME có thể phân đoạn hình ảnh thành "superpixels" và phân tích tầm quan trọng của từng vùng.

4.3.3 Công thức LIME

- Nguyên lý cốt lõi của LIME là tìm một mô hình đơn giản (interpretable model) để giải thích cục bộ hành vi của một mô hình phức tạp (black-box model) quanh một điểm dữ liệu cụ thể. Công thức tối ưu hóa của LIME được định nghĩa như sau:

Công thức chính:

$$\xi(x) = \arg \min_{g \in G} L(f, g, \pi_x) + \Omega(g)$$

Ý nghĩa: LIME tìm mô hình đơn giản g (trong tập hợp các mô hình dễ hiểu G) sao cho nó vừa xấp xỉ tốt hành vi của mô hình phức tạp f quanh điểm x , vừa giữ được độ đơn giản (được điều chỉnh bởi $\Omega(g)$).

Các thành phần trong công thức:

1. x : Điểm dữ liệu cần giải thích.

Trong RL: đây là trạng thái (state) s mà tác nhân đang xem xét, ví dụ: $s = [x = 5, y = 3, \text{distance} = 2, \text{angle} = 45]$

2. f : Mô hình phức tạp (black-box model).

- Trong RL: Đây là **chính sách (policy)** π của tác nhân, thường là một mạng nơ-ron sâu, trả về hành động $a=\pi(s)$ a hoặc giá trị $Q(s,a)$.

3. g : Mô hình đơn giản (interpretable model).

- Trong RL: Thường là hồi quy tuyến tính hoặc cây quyết định nhỏ, dùng để giải thích tại sao $\pi(s)$ chọn một hành động cụ thể.

4. G : Tập hợp các mô hình đơn giản có thể sử dụng.

- Ví dụ: G có thể là tập hợp các hàm tuyến tính hoặc cây quyết định với độ sâu giới hạn.

5. L(f,g,πx) : Hàm mất mát (loss function).

- Đo lường mức độ khác biệt giữa dự đoán của mô hình phức tạp f và mô hình đơn giản g trong vùng lân cận của x, với trọng số được xác định bởi π_x .
- Công thức chi tiết:

$$L(f, g, \pi_x) = \sum_{z' \in Z} \pi_x(z') \cdot (f(z') - g(z'))^2$$

- z' : Các mẫu giả (perturbed samples) được tạo quanh x (trong RL: các trạng thái giả s').
- $\pi_x(z')$: Trọng số của mẫu giả z' , thường là hàm kernel:

$$\pi_x(z') = \exp\left(-\frac{D(x, z')^2}{\sigma^2}\right)$$

6. Ω(g) : Độ phức tạp của mô hình đơn giản.

- Đây là một hàm phạt (regularization term) để đảm bảo g g g không quá phức tạp, giúp giải thích dễ hiểu.

Nếu g là hồi quy tuyến tính: $\Omega(g) = \lambda \sum |\beta_i|$ (L1 regularization).

Nếu g là cây quyết định: $\Omega(g)$ có thể là số lượng nút hoặc độ sâu tối đa.

Áp dụng công thức vào RL

Trong học tăng cường, LIME được điều chỉnh để giải thích hành động của tác nhân trong một trạng thái cụ thể. Hãy xem cách công thức hoạt động qua một ví dụ:

Ví dụ: Robot trong mê cung

- Trạng thái gốc:** $s = [x = 5, y = 3, distance_{to_wall} = 2, angle = 45^\circ]$.
- Hành động:** $a = \pi(s) = "rẽ phải"$.
- Mục tiêu:** Giải thích tại sao "rẽ phải" được chọn.

1. **Tạo mẫu giả:**

- Tạo $Z = \{s'_1, s'_2, \dots, s'_N\}$, ví dụ:

- $s'_1 = [5.1, 2.9, 1.8, 47^\circ]$
- $s'_2 = [4.8, 3.2, 2.1, 43^\circ]$.

2. **Tính trọng số $\pi_s(s')$:**

- Với s'_1 , khoảng cách $D(s, s'_1) = \sqrt{(0.1)^2 + (-0.1)^2 + (-0.2)^2 + (2)^2} \approx 0.36$.
- Nếu $\sigma = 1$, thì $\pi_s(s'_1) = \exp(-0.36^2/1^2) \approx 0.88$.
- Tương tự cho các mẫu khác.

3. **Dự đoán từ f :**

- $f(s'_1)$ = "rẽ phải", $f(s'_2)$ = "đi thẳng".

4. **Huấn luyện g :**

- Dùng hồi quy tuyến tính: $g(s') = w_1 \cdot x + w_2 \cdot y + w_3 \cdot distance + w_4 \cdot angle$.
- Tối ưu hóa $L(f, g, \pi_s) = \sum \pi_s(s')(f(s') - g(s'))^2$, với $f(s')$ là nhãn (ví dụ: 1 cho "rẽ phải", 0 cho hành động khác).

5. **Kết quả:**

- Nếu $g(s) = 0.7 \cdot distance + 0.2 \cdot angle - 0.1 \cdot x$, thì "distance_to_wall" là đặc trưng quan trọng nhất.

4.3.4 Ứng dụng LIME

Điều khiển robot (Robotics)

- **Ứng dụng:** LIME được sử dụng để giải thích hành vi của các robot tự động trong các nhiệm vụ như điều hướng, tránh chướng ngại vật, hoặc thao tác vật thể.

- **Ví dụ:**

- Một robot di chuyển trong mê cung được huấn luyện bằng RL để tránh tường và tìm lối ra. Khi robot chọn hành động "rẽ phải" trong trạng thái $s = [x = 5, y = 3, distance_{to_wall} = 2, angle = 45^\circ]$, LIME có thể chỉ ra rằng "khoảng cách đến tường" (70%) và "góc quay" (20%) là các yếu tố chính ảnh hưởng đến quyết định này.
- Trong robot hút bụi (như Roomba), LIME giải thích tại sao robot quay lại khi gặp ghế dựa trên "khoảng cách đến vật cản" và "hướng hiện tại".

Lợi ích:

- Giúp các kỹ sư hiểu và điều chỉnh chính sách của robot để tránh lỗi (ví dụ: rẽ quá sớm hoặc quá muộn).
- Tăng độ tin cậy của người dùng cuối khi họ thấy lý do cụ thể cho từng hành động.

2. Trò chơi điện tử (Game Playing)

- **Ứng dụng:** LIME hỗ trợ phân tích các quyết định của tác nhân RL trong các trò chơi, đặc biệt là các trò chơi phức tạp như Atari hoặc StarCraft, nơi DNN thường được sử dụng (ví dụ: DQN).

- **Ví dụ:**

- Trong trò chơi Pong, khi tác nhân RL di chuyển thanh chắn lên để đánh bóng, LIME có thể chỉ ra rằng "vị trí quả bóng trên trục y" (80%) và "tốc độ quả bóng" (15%) là các đặc trưng quan trọng nhất, trong khi "vị trí thanh chắn" ít ảnh hưởng hơn.
- Trong một game bắn súng, LIME giải thích tại sao tác nhân chọn "bắn" dựa trên "khoảng cách đến kẻ thù" và "mức đạn còn lại".

3. Hệ thống giao thông tự động (Autonomous Vehicles)

- **Ứng dụng:** LIME được áp dụng để giải thích các quyết định của xe tự lái hoặc hệ thống quản lý giao thông được huấn luyện bằng RL.

Ví dụ:

- Khi xe tự lái chọn "phanh" trong trạng thái $s = [speed = 60km/h, distance_to_car_ahead = 10m, traffic_light = red]$, LIME có thể xác định rằng "màu đèn giao thông" (60%) và "khoảng cách đến xe phía trước" (30%) là lý do chính, trong khi "tốc độ" ít quan trọng hơn trong trường hợp này.
- Trong quản lý đèn giao thông thông minh, LIME giải thích tại sao hệ thống chuyển đèn xanh dựa trên "mật độ xe" và "thời gian chờ".

Lợi ích:

- Đảm bảo an toàn bằng cách kiểm tra xem các yếu tố quan trọng (như đèn đỏ) có được ưu tiên hay không.
- Tăng sự minh bạch, giúp các nhà quản lý và người dùng hiểu rõ hơn về hành vi của hệ thống.

4. Y tế và chăm sóc sức khỏe (Healthcare)

- **Ứng dụng:** LIME được dùng để giải thích các tác nhân RL trong các hệ thống hỗ trợ y tế, chẳng hạn như quản lý liều thuốc hoặc điều khiển thiết bị y tế tự động.

Ví dụ:

- Trong một hệ thống RL điều chỉnh liều insulin cho bệnh nhân tiểu đường, LIME có thể giải thích tại sao hệ thống tăng liều dựa trên trạng thái $s = [blood_sugar = 200mg/dL, time_since_meal = 2h, heart_rate = 80bpm]$, với "mức đường huyết" (75%) và "thời gian kể từ bữa ăn" (20%) là các yếu tố chính.
- Trong robot phẫu thuật, LIME giải thích tại sao robot chọn "cắt" dựa trên "vị trí dao" và "hình ảnh mô".

Lợi ích:

- Hỗ trợ bác sĩ kiểm tra tính hợp lý của quyết định tự động.
- Tăng độ tin cậy và khả năng chấp nhận của công nghệ trong y tế.

5. Quản lý tài nguyên và hệ thống công nghiệp (Resource Management)

- **Ứng dụng:** LIME giúp giải thích hành vi của các tác nhân RL trong việc tối ưu hóa tài nguyên, như quản lý năng lượng, sản xuất, hoặc logistics.

Ví dụ:

- Trong một hệ thống RL điều khiển lưới điện thông minh, khi tác nhân chọn "tăng nguồn điện từ năng lượng mặt trời", LIME có thể chỉ ra rằng "mức tiêu thụ hiện tại" (65%) và "dự báo thời tiết" (25%) là yếu tố chính.
- Trong kho hàng tự động, LIME giải thích tại sao robot chọn "di chuyển hàng A" dựa trên "khoảng cách đến kệ" và "ưu tiên đơn hàng".

Lợi ích:

- Giúp các nhà quản lý tối ưu hóa quy trình bằng cách hiểu rõ yếu tố nào đang chi phối quyết định.
- Phát hiện lỗi hoặc sai lệch trong chính sách RL (ví dụ: ưu tiên sai nguồn tài nguyên).

6. Giáo dục và nghiên cứu (Education and Research)

- Ứng dụng:** LIME được dùng trong các công cụ học tập hoặc nghiên cứu RL để giúp sinh viên và nhà khoa học hiểu cách các mô hình hoạt động.

Ví dụ:

- Trong một bài tập RL đơn giản (như CartPole), LIME giải thích tại sao tác nhân chọn "đẩy sang phải" dựa trên "góc nghiêng của cột" và "vận tốc".
- Trong nghiên cứu, LIME giúp các nhà khoa học phân tích tác động của các đặc trưng trong môi trường mô phỏng (simulated environments).

Lợi ích:

- Cung cấp công cụ trực quan để giảng dạy khái niệm XRL.
- Hỗ trợ kiểm tra giả thuyết và cải thiện mô hình RL.

Lợi ích chung của LIME trong các ứng dụng RL

- Tính minh bạch:** LIME làm rõ lý do đằng sau các quyết định của tác nhân, tăng niềm tin từ người dùng và nhà phát triển.
- Chẩn đoán lỗi:** Giúp phát hiện các trường hợp mô hình ưu tiên sai đặc trưng (ví dụ: bỏ qua đèn đỏ trong xe tự lái).
- Tối ưu hóa:** Hỗ trợ điều chỉnh chính sách RL bằng cách xác định các đặc trưng cần cải thiện hoặc loại bỏ.

- **Tính linh hoạt:** Vì LIME là model-agnostic (không phụ thuộc mô hình), nó có thể áp dụng cho bất kỳ thuật toán RL nào (DQN, PPO, A3C, v.v.).

4.4 So sánh LIME, SHAP, VIPER

CHƯƠNG 5: Đồ án Reinforcement Learning Super Mario

5.1 Mô tả qua về game

Super Mario Bros là một tựa game kinh điển do Nintendo phát triển, lần đầu ra mắt năm 1985 trên hệ máy NES (Nintendo Entertainment System). Đây là một trong những trò chơi nền tảng (platformer) nổi tiếng nhất, với hàng triệu người chơi trên toàn thế giới.

a. Nội dung và mục tiêu

- **Nhân vật chính:** Mario (hoặc Luigi nếu chơi 2 người), một thợ sửa ống nước, phải giải cứu công chúa Peach khỏi Bowser - kẻ đứng đầu vương quốc Koopa.
- **Mục tiêu chính:** Di chuyển qua các màn chơi (levels), vượt qua chướng ngại vật, đánh bại kẻ thù, và đến được cột cờ cuối mỗi màn để tiến tới màn tiếp theo.

Cấu trúc game:

- Trò chơi được chia thành nhiều **worlds** (thế giới), mỗi thế giới có 4 levels (ví dụ: World 1-1, 1-2, 1-3, 1-4).
- Mỗi level có thời gian giới hạn (thường là 400 giây trong game).
- Level cuối của mỗi world (như 1-4, 2-4, v.v.) thường là một trận đấu với boss (Bowser hoặc phiên bản giả của hắn).

b. Cách chơi

- **Điều khiển:**
 - Di chuyển trái/phải: Mario chạy qua các nền tảng (platforms), nhảy lên để tránh chướng ngại vật hoặc tiêu diệt kẻ thù.
 - Nhảy (Jump): Nhảy lên để phá gạch, thu thập vật phẩm, hoặc nhảy qua kẻ thù.
 - Chạy nhanh (Sprint): Giữ nút chạy để tăng tốc và nhảy xa hơn.
- **Kẻ thù:**
 - Goomba (nấm nhỏ), Koopa Troopa (rùa), và nhiều kẻ thù khác.
 - Mario có thể tiêu diệt kẻ thù bằng cách nhảy lên đầu chúng hoặc dùng vật phẩm (như Fire Flower để bắn lửa).
- **Vật phẩm và power-ups:**
 - **Coin:** Thu thập 100 đồng xu để được thêm 1 mạng.
 - **Mushroom (Nấm):** Biến Mario thành Super Mario (to hơn, có thể phá gạch).
 - **Fire Flower (Hoa lửa):** Cho phép Mario bắn cầu lửa.
 - **Star (Ngôi sao):** Tạm thời bất tử, có thể chạy qua kẻ thù mà không mất máu.
- **Mạng sống:**
 - Mario bắt đầu với 3 mạng (lives).
 - Nếu bị kẻ thù chạm vào (khi ở trạng thái nhỏ), rơi xuống vực, hoặc hết thời gian, Mario mất 1 mạng.
 - Khi hết mạng, trò chơi kết thúc (game over).

c. Môi trường và đồ họa

- **Đồ họa:** Phong cách pixel 2D cổ điển, với các nền tảng, gạch, ống nước, và kẻ thù được thiết kế đơn giản nhưng đầy màu sắc.
- **Môi trường:** Bao gồm các thế giới với chủ đề khác nhau (cỏ, hang động, lâu đài, nước, v.v.).

d. Framework gym-super-mario-bros

- Trong dự án, trò chơi được tích hợp qua thư viện gym-super-mario-bros, một môi trường được thiết kế cho học máy:
 - **Môi trường:** SuperMarioBros-v0 là phiên bản cơ bản của World 1-1.

- **Hành động:** Có thể tùy chỉnh không gian hành động bằng JoypadSpace. Ví dụ: `[["NOOP"], ["right"], ["right", "A"], ["A"]]` giới hạn 4 hành động (đứng yên, đi phải, nhảy+phải, nhảy).
- **Trạng thái:** Là khung hình của game (một mảng numpy biểu diễn hình ảnh RGB, thường có kích thước 240x256x3).
- **Phần thưởng:** Dựa trên tiến độ của Mario (di chuyển xa hơn về bên phải, thu thập đồng xu, tiêu diệt kẻ thù, hoàn thành level, v.v.).

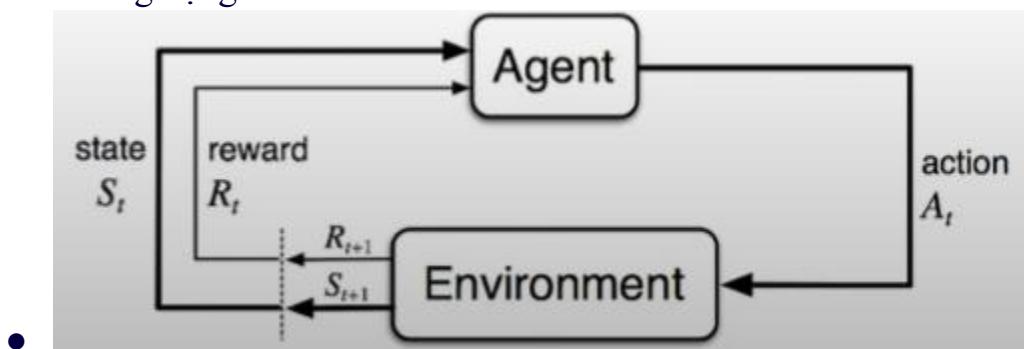
2. Dự án Super-Mario-Bros-RL

*Giới thiệu về Reinforcement Learning

- "Học tăng cường (Reinforcement Learning - RL) là một nhánh quan trọng của học máy, được xem là một trong những tiến bộ vượt bậc trong trí tuệ nhân tạo. Khác với học máy truyền thống dựa trên dữ liệu có nhãn, RL cho phép tác nhân học thông qua tương tác với môi trường, tối ưu hóa quyết định dựa trên phần thưởng. Các ứng dụng bao gồm chatbot, điều khiển robot, và trò chơi học máy."

*Ứng dụng trong Super Mario Bros RL

- "Dự án Super Mario Bros RL áp dụng RL thông qua thuật toán Double Deep Q Network (Double DQN) để huấn luyện AI chơi Super Mario Bros. AI quan sát khung hình từ trò chơi, chọn hành động (như di chuyển hoặc nhảy), và học từ phần thưởng (di chuyển xa hơn, thu thập đồng xu). Mục tiêu là vượt qua chướng ngại vật và hoàn thành level, minh họa khả năng của RL trong môi trường động."



- Tác nhân cung cấp hành động cho môi trường và môi trường trả lại trạng thái và phần thưởng. Một tập (episode) đối với chúng ta hôm nay sẽ chỉ là một lần thử chơi cấp độ. Một tập sẽ kết thúc khi Mario chết, đến cờ đích hoặc hết thời gian. Một chính sách (policy) là một hàm nhận vào một trạng thái và trả về một hành động. Đó là những gì tác nhân của chúng ta sẽ sử dụng để đưa ra quyết định. Một số thuật toán học tăng cường sẽ có chính sách là một phân phối xác suất, tuy nhiên hôm nay trong cách triển khai thuật toán DDQN, chúng ta sẽ sử dụng phương pháp Epsilon greedy.

- Replay buffer

$(s, a, r, s', \text{done})$

s = current state

a = action taken in current state

r = reward received for taking action a in state s

s' = next state

done = whether the episode is over

Experience được biểu diễn dưới dạng một tuple:

(s,a,r,s',done)

Trong đó:

1. **ss (current state):**

- Trạng thái hiện tại của môi trường. Đây là thông tin mà agent nhận được từ môi trường tại thời điểm hiện tại.
- Ví dụ: Trong trò chơi Super Mario Bros, trạng thái có thể là hình ảnh của màn hình trò chơi ở thời điểm hiện tại.

2. **aa (action taken in current state):**

- Hành động mà agent thực hiện trong trạng thái ss.
- Hành động này có thể được chọn dựa trên chính sách epsilon-greedy hoặc giá trị Q cao nhất.
- Ví dụ: Trong Super Mario Bros, hành động có thể là "nhảy", "di chuyển sang phải", hoặc "đứng yên".

3. **rr (reward received for taking action aa in state ss):**

- Phần thưởng mà agent nhận được sau khi thực hiện hành động aa trong trạng thái ss.

- Phần thưởng thường được thiết kế để khuyến khích agent đạt được mục tiêu, ví dụ: +1 điểm khi vượt qua chướng ngại vật hoặc hoàn thành màn chơi.

4. $s's'$ (next state):

- Trạng thái tiếp theo mà môi trường chuyển sang sau khi agent thực hiện hành động a .
- Ví dụ: Sau khi Mario nhảy qua một hố, trạng thái tiếp theo sẽ là hình ảnh mới của trò chơi.

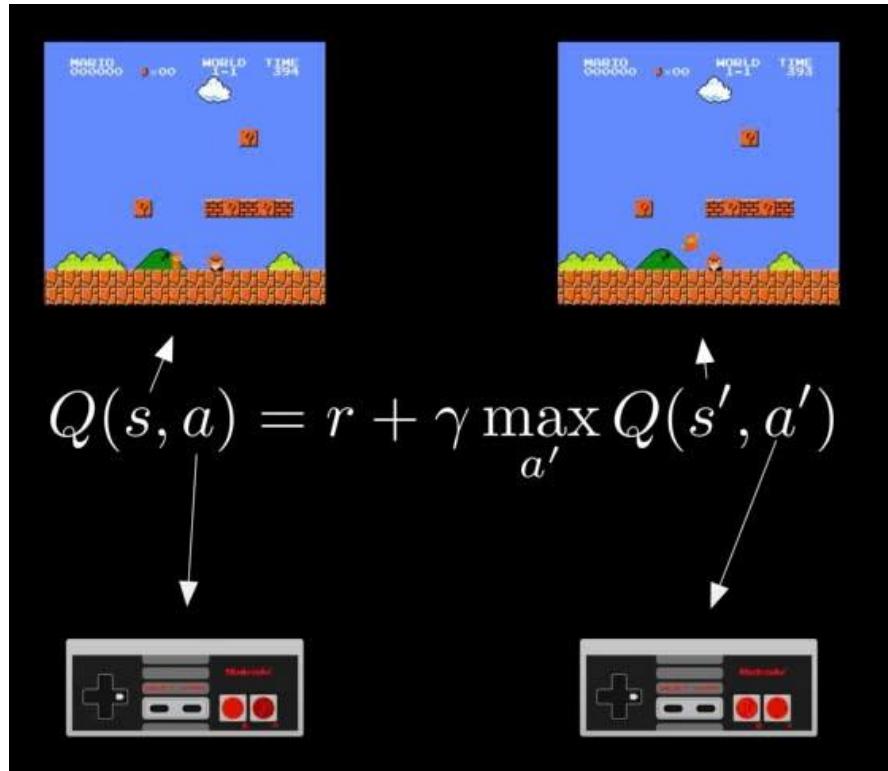
5. done(done (whether the episode is over)):

- Một giá trị Boolean (True/False) cho biết liệu tập hợp các trạng thái và hành động hiện tại đã kết thúc một tập (episode) hay chưa.
- Ví dụ: Trong Super Mario Bros, nếu Mario bị rơi xuống hố hoặc hoàn thành màn chơi, done=True=done=True.

Ý nghĩa của trải nghiệm này

- Trải nghiệm $(s,a,r,s',done)$ là dữ liệu cơ bản mà agent sử dụng để học cách tối ưu hóa hành vi của mình trong môi trường.
- Các trải nghiệm này được lưu trữ vào **Replay Buffer** để sử dụng lại trong quá trình huấn luyện mạng nơ-ron. Replay Buffer giúp cải thiện hiệu quả học bằng cách lấy mẫu ngẫu nhiên từ các trải nghiệm đã lưu trữ.

- Action value - Function Intuition



Một trải nghiệm được biểu diễn dưới dạng tuple: $(s, a, r, s', done)$

- **ss (current state):** Trạng thái hiện tại của môi trường.
- **aa (action):** Hành động mà agent thực hiện trong trạng thái ss.
- **rr (reward):** Phần thưởng nhận được sau khi thực hiện hành động aa.
- **s's' (next state):** Trạng thái tiếp theo sau khi thực hiện hành động.
- **donedone:** Giá trị Boolean cho biết liệu tập (episode) đã kết thúc hay chưa.

1. Agent ở trạng thái hiện tại (ss) và thực hiện một hành động (aa), ví dụ "nhảy" hoặc "di chuyển sang phải".
2. Sau khi thực hiện hành động, agent nhận phần thưởng (rr) và chuyển sang trạng thái tiếp theo (s's').
3. Giá trị Q được cập nhật dựa trên công thức để giúp agent học cách đưa ra quyết định tối ưu hơn trong tương lai.

$$Q(s, a) = r + \gamma \max_{a'} Q(s', a')$$

$$Q(s', a') = r' + \gamma \boxed{\max_{a''} Q(s'', a'')}$$

$$Q(s'', a'') = \boxed{r'' + \gamma \max_{a'''} Q(s''', a'''')}$$

Công thức được áp dụng lặp lại qua các trạng thái liên tiếp:

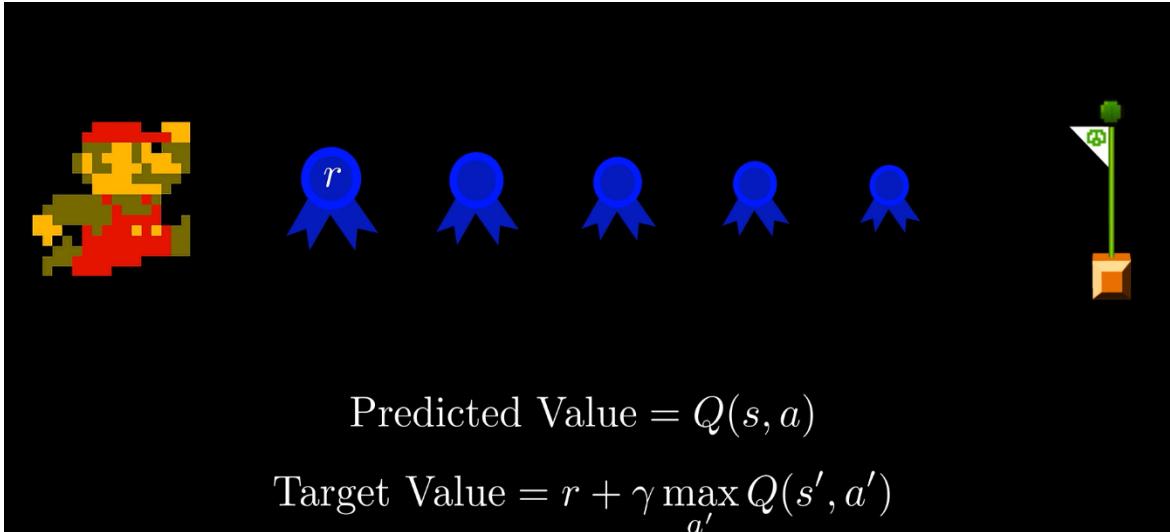
- $Q(s, a) = r + \gamma \max_a Q(s', a')$
- $Q(s', a') = r' + \gamma \max_a Q(s'', a'')$
- $Q(s'', a'') = r'' + \gamma \max_a Q(s''', a''')$

Ý nghĩa: Giá trị Q được lan truyền từ các phần thưởng tương lai về hiện tại. Điều này giúp agent học cách tối ưu hóa hành vi để đạt được phần thưởng cao nhất trong dài hạn. Quá trình cập nhật diễn ra liên tục qua các trạng thái để tối ưu hóa chiến lược của agent.

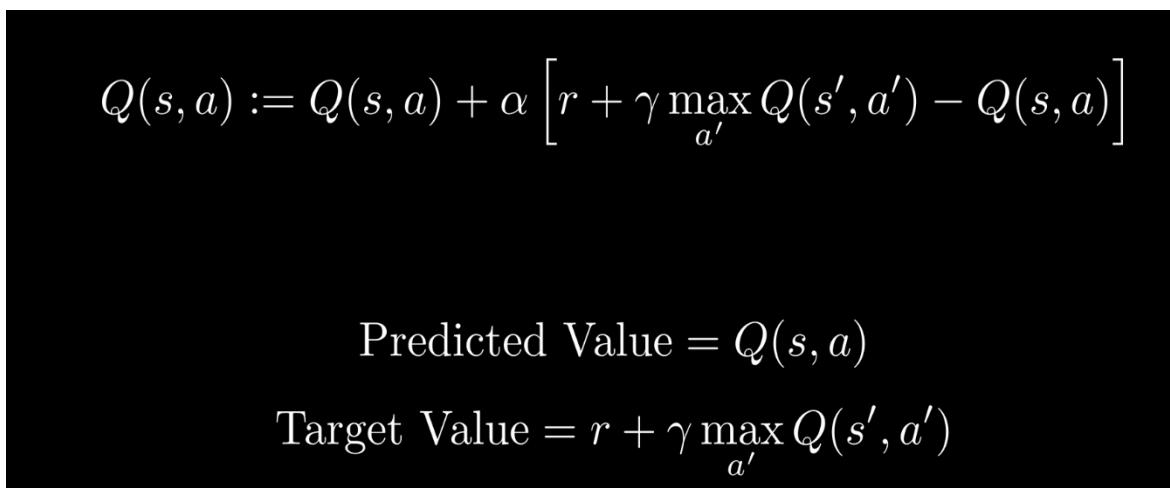
$$Q(s, a) = r + \gamma[r' + \gamma[r'' + \gamma \max_{a'''} Q(s''', a''')]]$$

$$Q(s, a) = r + \gamma r' + \gamma^2 r'' + \gamma^3 r''' + \dots$$

- Giá trị Q của hành động a tại trạng thái s được tính dựa trên phần thưởng hiện tại r và các phần thưởng tương lai $r', r'', \dots, r', r'', \dots$
- Hệ số chiết khấu γ (gamma) giảm dần tầm quan trọng của các phần thưởng tương lai.
- $\max_a Q(s''', a''')$ là giá trị Q tối đa có thể đạt được từ trạng thái tiếp theo s''' với hành động tối ưu a''' .
- Mỗi phần thưởng tương lai được nhân với một lũy thừa của hệ số chiết khấu γ :
 - r : Phần thưởng hiện tại.
 - $\gamma r' \gamma r'$: Phần thưởng ở bước tiếp theo, chiết khấu bởi γ .
 - $\gamma^2 r'' \gamma^2 r''$: Phần thưởng ở bước sau nữa, chiết khấu bởi γ^2 .



Mario, đại diện cho agent, bắt đầu từ trạng thái hiện tại và thực hiện các hành động nhằm tối ưu hóa phần thưởng nhận được. Các biểu tượng huy chương màu xanh biểu thị phần thưởng (r) mà Mario nhận được khi thực hiện các hành động như nhảy qua chướng ngại vật hoặc tiến gần hơn đến cột cờ (mục tiêu cuối cùng). Giá trị Q của một hành động tại trạng thái s được dự đoán bằng $Q(s, a)$ và được cập nhật dựa trên công thức: $\text{Target Value} = r + \gamma \max_{a'} Q(s', a')$, trong đó γ là hệ số chiết khấu, giúp giảm tầm quan trọng của các phần thưởng tương lai. Quá trình này cho phép AI học cách đánh giá mức độ "tốt" của mỗi hành động và dần tối ưu hóa chiến lược chơi để hoàn thành trò chơi một cách hiệu quả.



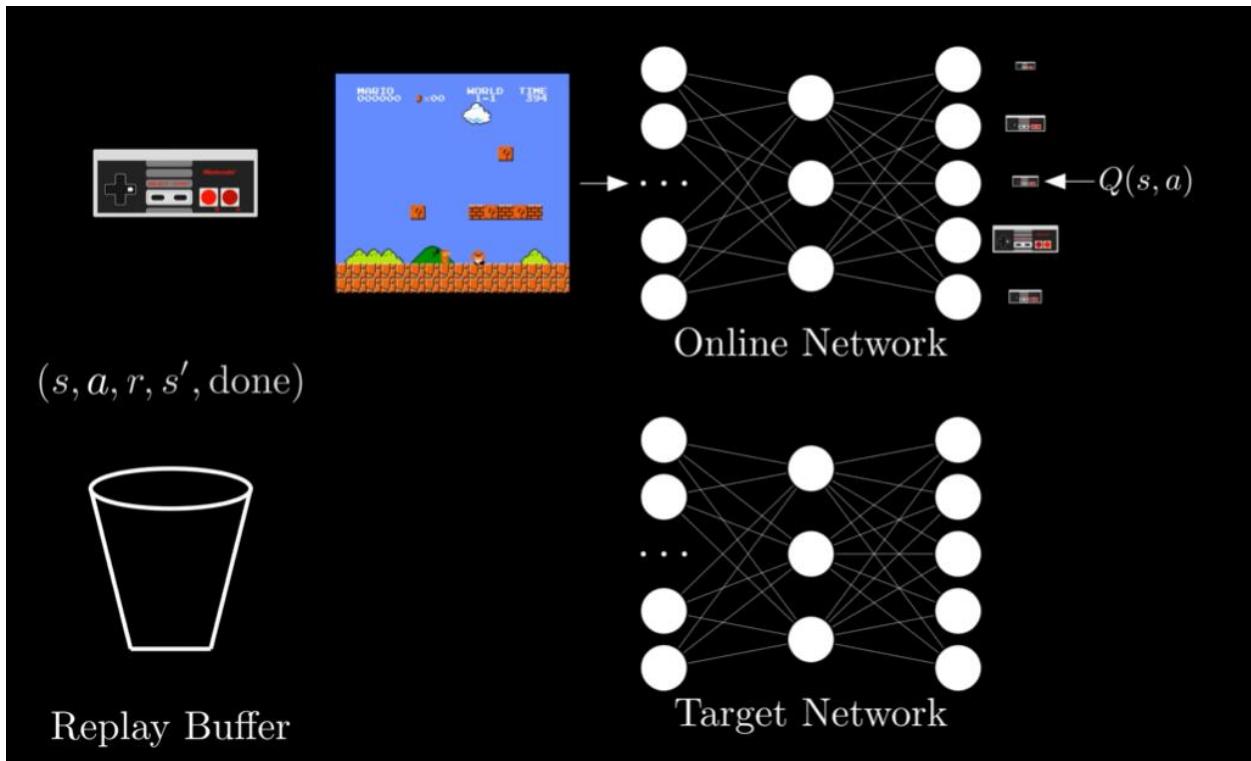
$$y = r + \gamma \max_{a'} Q(s', a')$$

$$\hat{y} = Q(s, a)$$

$$\text{MSE} = \frac{1}{2} (y - \hat{y})^2$$

$$\frac{\partial \text{MSE}}{\partial \hat{y}} = y - \hat{y}$$

- **The DQQN Algorithm**



Agent (Mario) học cách tối ưu hóa hành động của mình bằng cách tương tác với môi trường trò chơi.

1. Bộ điều khiển và trạng thái trò chơi:

- Agent nhận trạng thái hiện tại của trò chơi (hình ảnh màn hình) và sử dụng thông tin này để quyết định hành động tiếp theo, ví dụ: "nhảy", "di chuyển sang phải", hoặc "đứng yên".

2. Trải nghiệm $(s,a,r,s',done)$:

- Mỗi trải nghiệm bao gồm:
 - ss : Trạng thái hiện tại.
 - aa : Hành động thực hiện tại trạng thái ss .
 - rr : Phần thưởng nhận được sau hành động.
 - $s's'$: Trạng thái tiếp theo.
 - $donedone$: Giá trị Boolean cho biết liệu trò chơi đã kết thúc hay chưa.
- Các trải nghiệm này được lưu trữ trong **Replay Buffer** để sử dụng lại trong quá trình huấn luyện.

3. Mạng nơ-ron trực tuyến (Online Network):

- Mạng trực tuyến dự đoán giá trị Q cho mỗi hành động tại trạng thái hiện tại ($Q(s,a)Q(s,a)$).
- Agent chọn hành động dựa trên giá trị Q cao nhất hoặc theo chính sách epsilon-greedy.

4. Mạng nơ-ron mục tiêu (Target Network):

- Mạng mục tiêu cung cấp giá trị Q mục tiêu ($r+\gamma \max_{a'} Q(s',a')r+\gamma \max_{a'} Q(s',a')$), giúp ổn định quá trình huấn luyện.
- Mạng mục tiêu được đồng bộ hóa với mạng trực tuyến sau một số bước học nhất định.

5. Replay Buffer:

- Replay Buffer lưu trữ các trải nghiệm $(s,a,r,s',done)$ mà agent thu thập được trong quá trình chơi.

- Các trải nghiệm này được lấy mẫu ngẫu nhiên để huấn luyện mạng nơ-ron, giảm thiểu sự phụ thuộc vào dữ liệu thời gian và cải thiện hiệu quả học.
- Implementation in Code



```

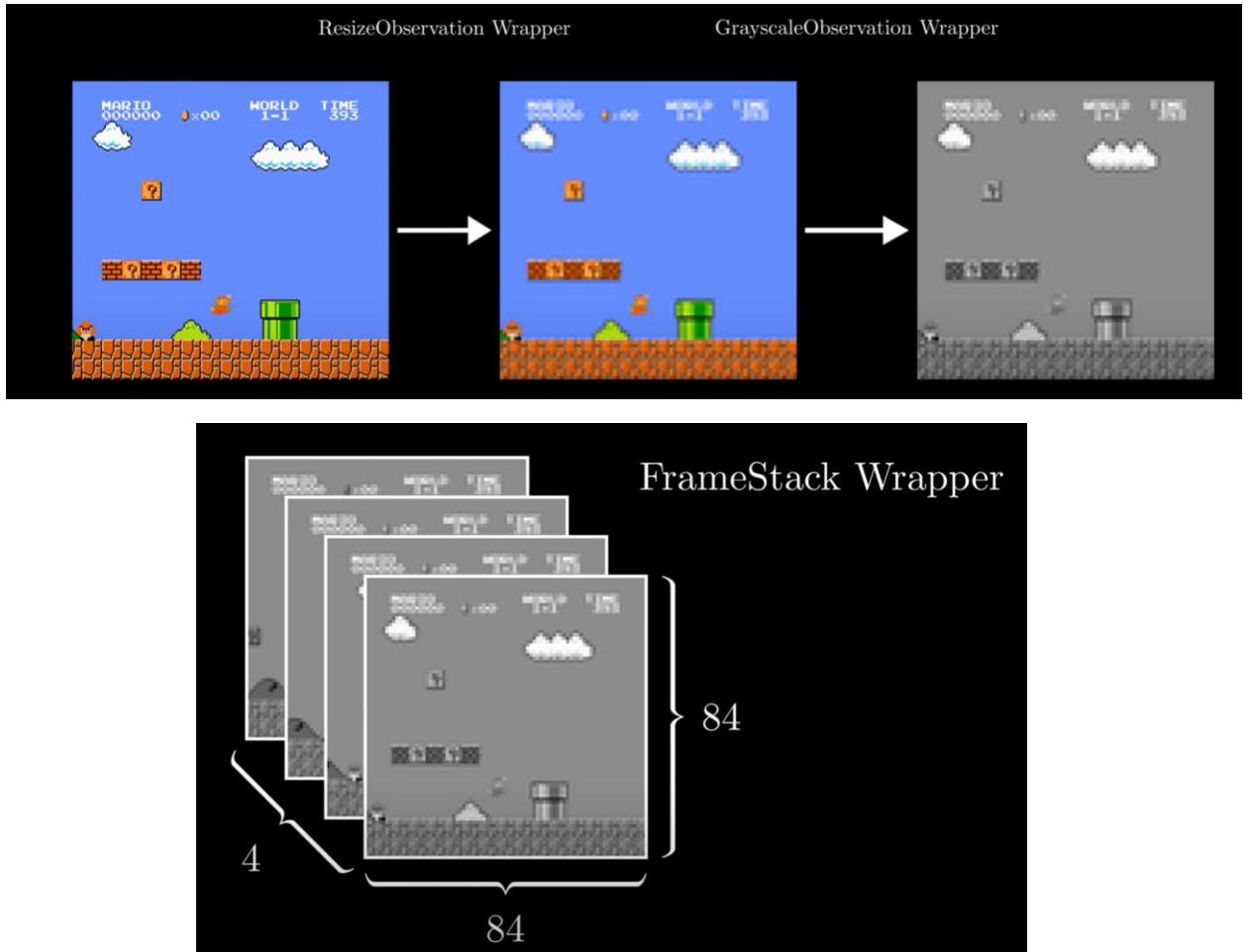
1 import gym_super_mario_bros
2 from gym_super_mario_bros.actions import RIGHT_ONLY
3 from nes_py.wrappers import JoypadSpace
4
5 ENV_NAME = 'SuperMarioBros-1-1-v0'
6 env = gym_super_mario_bros.make(ENV_NAME, render_mode='human', apply_api_compatibility=True)
7 env = JoypadSpace(env, RIGHT_ONLY)
8
9 done = False
10 env.reset()
11
12 while not done:
13     action = env.action_space.sample()
14     _, _, done, _ = env.step(action)
15     env.render()

```

- env = gym_super_mario_bros.make(...): Tạo môi trường Super Mario Bros với tên phiên bản (ENV_NAME) và chế độ hiển thị (render_mode='human').
- env = JoypadSpace(env, RIGHT_ONLY): Giới hạn hành động của Mario chỉ di chuyển sang phải.

while not done: Vòng lặp chạy cho đến khi trò chơi kết thúc.

- action = env.action_space.sample(): Lấy một hành động ngẫu nhiên từ không gian hành động.
- _, _, done, _ = env.step(action): Thực hiện hành động và cập nhật trạng thái của môi trường.
- env.render(): Hiển thị trạng thái hiện tại của trò chơi.



Quá trình xử lý đầu vào (observations) trong học tăng cường khi chơi trò chơi **Super Mario Bros**, nhằm tối ưu hóa dữ liệu đầu vào cho mạng nơ-ron.

Resize và Grayscale Observation Wrapper

1. ResizeObservation Wrapper:

- Trạng thái ban đầu của trò chơi là hình ảnh đầy đủ màu sắc với kích thước lớn.
- Hình ảnh này được giảm kích thước xuống còn 84x84 pixel, giúp giảm tải tính toán và làm cho dữ liệu đầu vào dễ xử lý hơn.

2. GrayscaleObservation Wrapper:

- Sau khi giảm kích thước, hình ảnh được chuyển sang dạng grayscale (anh xám), loại bỏ thông tin màu sắc không cần thiết.
- Chuyển sang grayscale giúp giảm số lượng kênh (channel) từ 3 (RGB) xuống còn 1, giữ lại các đặc điểm quan trọng như hình dạng và vị trí đối tượng.

FrameStack Wrapper

- Để giúp AI hiểu được chuyển động và ngữ cảnh của trò chơi, nhiều khung hình liên tiếp (frames) được xếp chồng lên nhau.
- Trong ví dụ này, 4 khung hình grayscale liên tiếp được xếp chồng thành một tensor có kích thước $(4, 84, 84)$.
- Việc xếp chồng các khung hình giúp AI nhận diện các thay đổi giữa các trạng thái (ví dụ: Mario đang nhảy hoặc di chuyển).

5.2 Triển khai đồ án

5.2.1 Mục tiêu triển khai

Mục tiêu của việc triển khai là thiết lập môi trường, huấn luyện agent, kiểm tra hiệu suất, và sử dụng LIME (Local Interpretable Model-agnostic Explanations) để giải thích hành động của agent. Các bước triển khai bao gồm clone mã nguồn từ GitHub, cài đặt Anaconda, thiết lập môi trường ảo, và chạy các file chính của đồ án.

5.2.2 Yêu cầu hệ thống

Để triển khai đồ án, hệ thống cần đáp ứng các yêu cầu sau:

- Hệ điều hành: Windows 10/11, macOS, hoặc Linux.
- Phần cứng:
 - CPU: Tối thiểu 4 nhân, khuyến nghị 8 nhân.
 - RAM: Tối thiểu 8GB, khuyến nghị 16GB.
 - GPU: Khuyến nghị NVIDIA GPU hỗ trợ CUDA (ví dụ: GTX 1050 trở lên) để tăng tốc huấn luyện.
- Phần mềm:
 - Git (dùng để clone repository).
 - Anaconda (dùng để quản lý môi trường ảo và thư viện).
 - Python 3.10 (phiên bản được sử dụng trong đồ án).

5.2.3 Các bước triển khai

1. Cài đặt Anaconda in Windows: Anaconda được sử dụng để quản lý môi trường ảo và các thư viện Python cần thiết cho đồ án.
- Tải Anaconda:
 - Truy cập trang chính thức của Anaconda:
<https://www.anaconda.com/products/distribution>.
 - Tải phiên bản phù hợp với hệ điều hành:
 - Windows: Chọn phiên bản 64-bit (hoặc 32-bit nếu máy tính là 32-bit).
 - macOS/Linux: Chọn phiên bản tương ứng.
 - File tải về có dạng: Anaconda3-2023.09-0-Windows-x86_64.exe.
 - Cài đặt Anaconda:
 - Chạy file cài đặt.
 - Làm theo hướng dẫn:
 1. Chọn "Install for: Just Me".
 2. Chọn thư mục cài đặt (mặc định: C:\Users\<YourUsername>\Anaconda3).
 3. Đánh dấu cả hai tùy chọn:

"Add Anaconda3 to my PATH environment variable".

"Register Anaconda3 as my default Python 3.x".

- Hoàn tất cài đặt.

2. Kiểm tra cài đặt

Mở CMD và kiểm tra:

```
conda --version
```

Nếu cài đặt thành công, kết quả sẽ hiển thị phiên bản của conda (ví dụ: conda 23.9.0).

3. Clone Git Repository

Sử dụng lệnh sau để clone mã nguồn hoặc tải trực tiếp trong đường link Github:

```
git clone https://github.com/DogguG/Reinforcement-Learning-Mario-Project
```

4. Thiết lập môi trường ảo: Môi trường ảo được tạo để cô lập các thư viện của đồ án, tránh xung đột với các dự án khác.

- Tạo môi trường ảo với Python 3.10

```
conda create -n smbrl python=3.10
```

Kích hoạt môi trường

```
conda activate smbrl
```

Sau khi kích hoạt, dấu nhắc lệnh sẽ hiển thị (smbrl).

- Cài đặt các thư viện cần thiết

Cài đặt PyTorch (đối với máy sử dụng GPU Nvidia), cài đặt PyTorch với CUDA 12.4:

```
pip install torch==2.4.0 torchvision==0.19.0 torchaudio==2.4.0 --index-url  
https://download.pytorch.org/whl/cu124
```

Cài đặt các thư viện khác:

```
pip install gym-super-mario-bros==7.4.0 nes-py==8.2.1 numpy==1.23.5  
matplotlib==3.5.3 scikit-image==0.19.3 lime tensordict==0.1.2 torchrl==0.1.2  
gym==0.26.2
```

Danh sách thư viện bao gồm:

gym-super-mario-bros==7.4.0, nes-py==8.2.1: Môi trường Super Mario Bros.
numpy==1.23.5: Xử lý mảng.
matplotlib==3.5.3: Vẽ hình ảnh LIME.
scikit-image==0.19.3, lime: Cần cho LIME.
tensordict==0.1.2, torchrl==0.1.2: Thư viện phụ trợ cho PyTorch.
gym==0.26.2: Thư viện môi trường.

- Kiểm tra cài đặt

Kiểm tra danh sách thư viện

```
pip list
```

Kiểm tra PyTorch và CUDA

```
python -c "import torch; print(torch.__version__); print(torch.cuda.is_available())"
```

Kết quả mong đợi:

2.4.0+cu124

True

5. Chạy đồ án
 - Huấn luyện Agent

Kích hoạt môi trường

conda activate smbrl

Chạy file Python

python main.py

Kết quả:

1. File main.py sẽ huấn luyện agent qua 1 triệu episode (hoặc số episode đã chỉnh sửa).
2. Mỗi 5000 episode, mô hình được lưu vào thư mục models (ví dụ: model_5000_iter.pt).
3. In reward sau mỗi episode

Episode: 0, Reward: [reward], Epsilon: 1.0
Episode: 1, Reward: [reward], Epsilon: 0.99999
...
• Kiểm tra Agent

Chạy file Python

python D:/Projects/Super-Mario-Bros-RL/generate_clips.py

Kết quả:

1. Mỗi 100 frame, hình ảnh giải thích được hiển thị
2. In tổng rewards sau mỗi lượt chơi

Episode 0, Total Reward: 257
Episode 1, Total Reward: 351

5.2.4 Giải thích từng hàm trong đồ án

1. agent_nn.py

```
import torch
from torch import nn
import numpy as np

class AgentNN(nn.Module):
    def __init__(self, input_shape, n_actions, freeze=False):
        super().__init__()
        # Convolutional layers
        self.conv_layers = nn.Sequential(
            nn.Conv2d(input_shape[0], 32, kernel_size=8, stride=4),
            nn.ReLU(),
            nn.Conv2d(32, 64, kernel_size=4, stride=2),
            nn.ReLU(),
            nn.Conv2d(64, 64, kernel_size=3, stride=1),
            nn.ReLU(),
        )

        conv_out_size = self._get_conv_out(input_shape)

        # Linear layers
        self.network = nn.Sequential(
            self.conv_layers,
            nn.Flatten(),
            nn.Linear(conv_out_size, 512),
            nn.ReLU(),
            nn.Linear(512, n_actions)
        )

    if freeze:
        self._freeze()

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

    def forward(self, x):
        return self.network(x)

    def _get_conv_out(self, shape):
```

```

o = self.conv_layers(torch.zeros(1, *shape))
# np.prod returns the product of array elements over a given axis
return int(np.prod(o.size()))

def _freeze(self):
    for p in self.network.parameters():
        p.requires_grad = False

```

Giai thích :

- Thư viện torch và torch.nn: Đây là một phần của thư viện PyTorch. torch là thư viện cốt lõi, và torch.nn cung cấp các module và lớp để xây dựng mạng nơ-ron.
- numpy với tên gọi np: Một thư viện cho các phép toán số học trong Python, được sử dụng ở đây để tính tích của các chiều.
- Định nghĩa lớp: Agent_nn kế thừa từ nn.Module, là lớp cơ sở cho tất cả các module mạng nơ-ron trong PyTorch.

Khởi tạo (`__init__` method):

a. Tham số gồm :

- `input_shape`: Hình dạng của dữ liệu đầu vào (ví dụ: kích thước của một hình ảnh).
- `n_actions`: Số lượng hành động đầu ra hoặc lớp mà mạng cần dự đoán.
- `freeze`: Một giá trị boolean chỉ định có đóng băng các tham số của mạng hay không (tức là ngăn chúng được cập nhật trong quá trình huấn luyện).

Các lớp tích chập (Convolutional Layers):

- Ba lớp tích chập được định nghĩa bằng `nn.Conv2d`, mỗi lớp sau bởi một hàm kích hoạt ReLU (`nn.ReLU()`).
- Các lớp này được sử dụng để trích xuất đặc trưng từ dữ liệu đầu vào.

b. Tính toán kích thước đầu ra:

- `_get_conv_out` là một phương thức trợ giúp tính toán kích thước đầu ra từ các lớp tích chập, cần thiết để định nghĩa lớp tuyến tính đầu tiên.

c. Các lớp tuyến tính (Linear Layers):

- Một chuỗi các lớp bao gồm một lớp làm phẳng (nn.Flatten()), một lớp kết nối đầy đủ (nn.Linear), một hàm kích hoạt ReLU, và một lớp kết nối đầy đủ khác để tạo ra đầu ra cuối cùng.
- Đóng băng tham số:
- Nếu freeze là True, phương thức `_freeze` được gọi để đặt `requires_grad` thành False cho tất cả các tham số, ngăn chúng được cập nhật trong quá trình huấn luyện.

d. Thiết lập thiết bị:

- Mô hình được chuyển sang GPU nếu có sẵn, nếu không thì sẽ ở trên CPU.
- Phương thức Forward:
- Định nghĩa cách dữ liệu đầu vào x được truyền qua mạng để tạo ra đầu ra
- Phương thức hỗ trợ gồm :
- `_get_conv_out`: Tính toán kích thước đầu ra từ các lớp tích chập bằng cách truyền một tensor giả qua chúng và tính tích của các chiều của nó.
- `_freeze`: Duyệt qua tất cả các tham số trong mạng và đặt `requires_grad` thành False, thực hiện việc đóng băng mô hình.
- Lớp này là một thiết lập điển hình cho một mạng nơ-ron tích chập (CNN) được sử dụng trong các tác vụ mà đầu vào là dữ liệu dạng hình ảnh, và có thể được mở rộng hoặc sửa đổi cho các ứng dụng cụ thể.

2. agent.py

```
import torch
import numpy as np
from agent_nn import AgentNN

from tensordict import TensorDict
from torchrl.data import TensorDictReplayBuffer, LazyMemmapStorage

class Agent:
    def __init__(self,
                 input_dims,
                 num_actions,
                 lr=0.00025,
```

```

        gamma=0.9,
        epsilon=1.0,
        eps_decay=0.99999975,
        eps_min=0.1,
        replay_buffer_capacity=100_000,
        batch_size=32,
        sync_network_rate=10000):

    self.num_actions = num_actions
    self.learn_step_counter = 0

    # Hyperparameters
    self.lr = lr
    self.gamma = gamma
    self.epsilon = epsilon
    self.eps_decay = eps_decay
    self.eps_min = eps_min
    self.batch_size = batch_size
    self.sync_network_rate = sync_network_rate

    # Networks
    self.online_network = AgentNN(input_dims, num_actions)
    self.target_network = AgentNN(input_dims, num_actions, freeze=True)

    # Optimizer and loss
    self.optimizer = torch.optim.Adam(self.online_network.parameters(), lr=self.lr)
    self.loss = torch.nn.MSELoss()
    # self.loss = torch.nn.SmoothL1Loss() # Feel free to try this loss function instead!

    # Replay buffer
    storage = LazyMemmapStorage(replay_buffer_capacity)
    self.replay_buffer = TensorDictReplayBuffer(storage=storage)

def choose_action(self, observation):
    if np.random.random() < self.epsilon:
        return np.random.randint(self.num_actions)
    # Passing in a list of numpy arrays is slower than creating a tensor from a numpy array
    # Hence the `np.array(observation)` instead of `observation`
    # observation is a LIST of numpy arrays because of the LazyFrame wrapper

```

```

# Unsqueeze adds a dimension to the tensor, which represents the batch dimension
observation = torch.tensor(np.array(observation), dtype=torch.float32) \
    .unsqueeze(0) \
    .to(self.online_network.device)

# Grabbing the index of the action that's associated with the highest Q-value
return self.online_network(observation).argmax().item()

def decay_epsilon(self):
    self.epsilon = max(self.epsilon * self.eps_decay, self.eps_min)

def store_in_memory(self, state, action, reward, next_state, done):
    self.replay_buffer.add(TensorDict({
        "state": torch.tensor(np.array(state), dtype=torch.float32),
        "action": torch.tensor(action),
        "reward": torch.tensor(reward),
        "next_state": torch.tensor(np.array(next_state), dtype=torch.float32),
        "done": torch.tensor(done)
    }, batch_size=[]))

def sync_networks(self):
    if self.learn_step_counter % self.sync_network_rate == 0 and self.learn_step_counter > 0:
        self.target_network.load_state_dict(self.online_network.state_dict())

def save_model(self, path):
    torch.save(self.online_network.state_dict(), path)

def load_model(self, path):
    self.online_network.load_state_dict(torch.load(path))
    self.target_network.load_state_dict(torch.load(path))

def learn(self):
    if len(self.replay_buffer) < self.batch_size:
        return

    self.sync_networks()

    self.optimizer.zero_grad()

    samples = self.replay_buffer.sample(self.batch_size).to(self.online_network.device)

```

```

keys = ("state", "action", "reward", "next_state", "done")

states, actions, rewards, next_states, dones = [samples[key] for key in keys]

predicted_q_values = self.online_network(states) # Shape is (batch_size, n_actions)
predicted_q_values = predicted_q_values[np.arange(self.batch_size), actions.squeeze()]

# Max returns two tensors, the first one is the maximum value, the second one is the index of the
maximum value
target_q_values = self.target_network(next_states).max(dim=1)[0]
# The rewards of any future states don't matter if the current state is a terminal state
# If done is true, then 1 - done is 0, so the part after the plus sign (representing the future rewards) is 0
target_q_values = rewards + self.gamma * target_q_values * (1 - dones.float())

loss = self.loss(predicted_q_values, target_q_values)
loss.backward()
self.optimizer.step()

self.learn_step_counter += 1
self.decay_epsilon()

```

Giải thích:

1. Thư viện sử dụng:

- **torch và numpy**: Được sử dụng để xử lý tensor và các phép toán số học.
- **AgentNN**: Lớp mạng nơ-ron được định nghĩa trước đó, sử dụng để tạo các mạng nơ-ron cho tác nhân.
- **TensorDict, TensorDictReplayBuffer, LazyMemmapStorage**: Các công cụ từ **torchrl.data** để quản lý bộ nhớ đệm và lưu trữ dữ liệu.

2. Định nghĩa lớp Agent:

- **Khởi tạo (`__init__` method)**:

- **Tham số**:

- **input_dims, num_actions**: Kích thước đầu vào và số lượng hành động.

- **lr, gamma, epsilon, eps_decay, eps_min:** Các siêu tham số cho quá trình học.
- **replay_buffer_capacity, batch_size, sync_network_rate:** Các tham số cho bộ nhớ đệm và đồng bộ hóa mạng.
- **Mạng nơ-ron:**
 - **online_network:** Mạng nơ-ron chính để dự đoán giá trị Q.
 - **target_network:** Mạng nơ-ron mục tiêu để ổn định quá trình học, được đồng bộ hóa định kỳ với **online_network**.
- **Bộ tối ưu hóa và hàm mất mát:**
 - Sử dụng Adam optimizer và MSELoss để tối ưu hóa và tính toán mất mát.
- **Bộ nhớ đệm:**
 - Sử dụng **TensorDictReplayBuffer** để lưu trữ và quản lý các trải nghiệm.

3. Phương thức choose_action:

- Chọn hành động dựa trên epsilon-greedy: với xác suất **epsilon**, chọn hành động ngẫu nhiên; ngược lại, chọn hành động có giá trị Q cao nhất từ **online_network**.

4. Phương thức decay_epsilon:

- Giảm dần giá trị **epsilon** theo thời gian để giảm tần suất chọn hành động ngẫu nhiên.

5. Phương thức store_in_memory:

- Lưu trữ trạng thái, hành động, phần thưởng, trạng thái tiếp theo và trạng thái kết thúc vào bộ nhớ đệm.

6. Phương thức sync_networks:

- Đồng bộ hóa **target_network** với **online_network** sau mỗi **sync_network_rate** bước học.

7. Phương thức save_model và load_model:

- Lưu và tải trạng thái của **online_network** và **target_network**.

8. Phương thức learn:

- Thực hiện một bước học từ bộ nhớ đệm:
 - Lấy mẫu từ bộ nhớ đệm.

- Tính toán giá trị Q dự đoán và giá trị Q mục tiêu.
- Tính toán mát mát và cập nhật **online_network** bằng cách sử dụng gradient descent.
- Tăng bộ đếm bước học và giảm **epsilon**.

3. generate_clips.py

```
import gym_super_mario_bros

from gym_super_mario_bros.actions import RIGHT_ONLY

from nes_py.wrappers import JoypadSpace

from agent import Agent

from gym import Wrapper

from gym.wrappers import GrayScaleObservation, ResizeObservation, FrameStack

import os

from PIL import Image

import torch

import numpy as np

from lime import lime_image

import matplotlib.pyplot as plt

from skimage.segmentation import mark_boundaries


# Modified SkipFrame wrapper to log frames and actions

class SkipFrame(Wrapper):

    def __init__(self, env, skip):

        super().__init__(env)

        self._skip = skip

        self.counter = 0
```

```

self.frames_log = []
self.actions_log = []

def step(self, action):
    total_reward = 0.0
    done = False
    for _ in range(self._skip):
        next_state, reward, done, trunc, info = self.env.step(action)
        self.frames_log.append(next_state.copy())
        self.actions_log.append(action)
        total_reward += reward
        if done:
            break
    return next_state, total_reward, done, trunc, info

def reset(self, **kwargs):
    state, info = self.env.reset(**kwargs)
    self.frames_log = [state.copy()]
    self.actions_log = [0]
    return state, info

def apply_wrappers(env):
    env = SkipFrame(env, skip=4)
    env = ResizeObservation(env, shape=84)
    env = GrayScaleObservation(env)
    env = FrameStack(env, num_stack=4, lz4_compress=True)
    return env

```

```

# Hàm để chuyển trạng thái thành hình ảnh RGB (cho LIME)

def state_to_rgb(state):

    # state có shape (4, 84, 84) (4 khung hình grayscale)

    # Lấy khung hình cuối cùng (index 3) và chuyển thành RGB

    frame = state[3] # Khung hình cuối cùng

    frame = np.repeat(frame[:, :, np.newaxis], 3, axis=2) # Chuyển thành RGB

    return frame.astype(np.uint8)

# Hàm dự đoán cho LIME

def predict_fn(images, agent):

    # images: List các hình ảnh RGB (84, 84, 3)

    # Chuyển thành định dạng phù hợp với online_network (4, 84, 84)

    processed_states = []

    for img in images:

        # Chuyển RGB thành grayscale

        img_gray = np.mean(img, axis=2).astype(np.float32)

        # Tạo stack 4 khung hình (dùng cùng khung hình cho đơn giản)

        img_stack = np.stack([img_gray] * 4, axis=0) # Shape: (4, 84, 84)

        processed_states.append(img_stack)

    processed_states = np.array(processed_states) # Shape: (batch_size, 4, 84, 84)

    states_tensor = torch.tensor(processed_states, dtype=torch.float32).to(agent.online_network.device)

    with torch.no_grad():

        q_values = agent.online_network(states_tensor) # Shape: (batch_size, n_actions)

        return q_values.cpu().numpy()

```

```
# Hàm giải thích bằng LIME

def explain_action(state, action, agent, episode, frame_idx):

    # Chuyển trạng thái thành hình ảnh RGB
    state_rgb = state_to_rgb(state)

    # Tạo explainer
    explainer = lime_image.LimeImageExplainer()

    # Giải thích
    explanation = explainer.explain_instance(
        state_rgb,
        lambda x: predict_fn(x, agent),
        top_labels=1,
        num_samples=1000
    )

    # Trực quan hóa
    temp, mask = explanation.get_image_and_mask(
        explanation.top_labels[0],
        positive_only=True,
        num_features=5
    )

    plt.figure(figsize=(10, 5))
    plt.subplot(1, 2, 1)
    plt.imshow(state_rgb)
    plt.title(f"Original State (Episode {episode}, Frame {frame_idx})")
```

```

plt.subplot(1, 2, 2)

plt.imshow(mark_boundaries(state_rgb, mask))

plt.title(f'LIME Explanation: Action {action}')

plt.show()

ENV_NAME = 'SuperMarioBros-1-1-v0'

NUM_OF_EPISODES = 1_000

# controllers = [Image.open(f'controllers/{i}.png') for i in range(5)]


env = gym_super_mario_bros.make(ENV_NAME, render_mode='rgb_array', apply_api_compatibility=True)

env = JoypadSpace(env, RIGHT_ONLY)

env = apply_wrappers(env)

agent = Agent(input_dims=env.observation_space.shape, num_actions=env.action_space.n)

# agent.load_model("models/folder_name/ckpt_name")

for i in range(NUM_OF_EPISODES):

    done = False

    state, _ = env.reset()

    rewards = 0

    frame_idx = 0

    while not done:

        action = agent.choose_action(state)

        frame = env.render()

        new_state, reward, done, truncated, info = env.step(action)

        rewards += reward

```

```

# Giải thích hành động bằng LIME (chỉ làm cho một vài frame để tránh chật)

if frame_idx % 100 == 0: # Giải thích mỗi 100 frame

    explain_action(state, action, agent, i, frame_idx)

state = new_state

frame_idx += 1


if done:

    print(f"Episode: {i}, Reward: {rewards}")

    if info["flag_get"]:

        os.makedirs(os.path.join("games", f"game_{i}"), exist_ok=True)

        frame_skip_env = env.env.env.env # Unwrapping the environment to get the SkipFrame wrapper

        frames_log = frame_skip_env.frames_log

        actions_log = frame_skip_env.actions_log

        for j, (frame, action) in enumerate(zip(frames_log, actions_log)):

            scaling_factor = 10

            new_dims = (frame.shape[1] * scaling_factor, frame.shape[0] * scaling_factor)

            frame = Image.fromarray(frame).resize(new_dims, Image.NEAREST)

            frame.save(os.path.join("games", f"game_{i}", f"frame_{j}.png"))

            # controllers[action].save(os.path.join("games", f"game_{i}", f"controller_{j}.png"))

# if i % 5000 == 0 and i > 0:

#     agent.save_model(os.path.join("models", f"model_{i}_iter.pt"))

env.close()

```

Giải thích: Đoạn mã này thiết lập một môi trường học tăng cường cho trò chơi Super Mario Bros, sử dụng một tác nhân học sâu để điều khiển Mario. Nó cũng sử dụng LIME (Local Interpretable Model-agnostic Explanations) để giải thích các hành động của tác nhân.

1. Thư viện sử dụng:

- **gym_super_mario_bros, nes_py.wrappers:** Để tạo và quản lý môi trường Super Mario Bros.
- **agent:** Lớp tác nhân đã được định nghĩa trước đó.
- **gym, gym.wrappers:** Để áp dụng các bộ bao bọc (wrappers) cho môi trường.
- **PIL, torch, numpy, lime, matplotlib, skimage:** Các thư viện hỗ trợ xử lý hình ảnh, tensor, và trực quan hóa.

2. Lớp SkipFrame:

Một bộ bao bọc tùy chỉnh để bỏ qua một số khung hình nhất định, giúp tăng tốc độ xử lý và ghi lại các khung hình và hành động.

3. Hàm apply_wrappers:

Áp dụng các bộ wrappers (bao bọc) cho môi trường, bao gồm **SkipFrame**, **ResizeObservation**, **GrayScaleObservation**, và **FrameStack**.

4. Hàm state_to_rgb:

Chuyển đổi trạng thái từ định dạng grayscale (4, 84, 84) sang hình ảnh RGB để sử dụng với LIME.

5. Hàm predict_fn:

Dự đoán giá trị Q cho các hình ảnh đầu vào, được sử dụng bởi LIME để giải thích.

6. Hàm explain_action:

Sử dụng LIME để giải thích hành động của tác nhân tại một trạng thái cụ thể, và trực quan hóa kết quả.

7. Thiết lập môi trường và tác nhân:

Tạo môi trường Super Mario Bros với các bộ bao bọc cần thiết.

Khởi tạo tác nhân với các tham số phù hợp.

8. Vòng lặp huấn luyện:

Chạy qua một số tập (episodes) nhất định.

Tại mỗi bước, tác nhân chọn một hành động dựa trên trạng thái hiện tại.

Mỗi trường thực hiện hành động và trả về trạng thái mới, phần thưởng, và thông tin khác.

Mỗi 100 khung hình, LIME được sử dụng để giải thích hành động của tác nhân.

Nếu Mario hoàn thành cấp độ, các khung hình và hành động được lưu lại.

9. Lưu và tải mô hình:

Mô hình có thể được lưu lại sau một số tập nhất định và tải lại khi cần thiết.

4. lib.py

```
import torch
import gym_super_mario_bros
import lime
import skimage
import matplotlib
import numpy

print("NumPy version:", numpy.__version__)
print("PyTorch version:", torch.__version__)
print("CUDA available:", torch.cuda.is_available())
print("Gym-Super-Mario-Bros version:", gym_super_mario_bros.__version__)
print("LIME installed:", lime.__version__ if hasattr(lime, '__version__') else "Yes")
print("Scikit-Image installed:", skimage.__version__)
print("Matplotlib installed:", matplotlib.__version__)
```

Giải thích : Đoạn mã này kiểm tra và in ra phiên bản của một số thư viện Python quan trọng được sử dụng trong các ứng dụng học máy và xử lý hình ảnh.

1. Thư viện sử dụng:

- torch: Thư viện PyTorch, được sử dụng cho học sâu và tính toán tensor.
- gym_super_mario_bros: Thư viện để tạo môi trường trò chơi Super Mario Bros cho học tăng cường.
- lime: Thư viện LIME, được sử dụng để giải thích các mô hình học máy.
- skimage: Thư viện Scikit-Image, được sử dụng cho xử lý hình ảnh.
- matplotlib: Thư viện Matplotlib, được sử dụng để trực quan hóa dữ liệu.
- numpy: Thư viện NumPy, được sử dụng cho các phép toán số học.

2. In ra phiên bản của các thư viện:

- numpy.__version__: In ra phiên bản của NumPy.
- torch.__version__: In ra phiên bản của PyTorch.
- torch.cuda.is_available(): Kiểm tra xem CUDA có khả dụng không, điều này cho biết liệu có thể sử dụng GPU để tăng tốc tính toán với PyTorch.
- gym_super_mario_bros.__version__: In ra phiên bản của thư viện Gym-Super-Mario-Bros.
- lime.__version__: In ra phiên bản của LIME nếu có thuộc tính __version__, nếu không thì in "Yes" để xác nhận rằng LIME đã được cài đặt.
- skimage.__version__: In ra phiên bản của Scikit-Image.
- matplotlib.__version__: In ra phiên bản của Matplotlib.

5. main.py

```
import torch

import gym_super_mario_bros
from gym_super_mario_bros.actions import RIGHT_ONLY

from agent import Agent

from nes_py.wrappers import JoypadSpace
```

```
from wrappers import apply_wrappers

import os

from utils import *

model_path = os.path.join("models", get_current_date_time_string())
os.makedirs(model_path, exist_ok=True)

if torch.cuda.is_available():
    print("Using CUDA device:", torch.cuda.get_device_name(0))
else:
    print("CUDA is not available")

ENV_NAME = 'SuperMarioBros-1-1-v0'
SHOULD_TRAIN = True
DISPLAY = True
CKPT_SAVE_INTERVAL = 5000
NUM_OF_EPISODES = 50_000

env = gym_super_mario_bros.make(ENV_NAME, render_mode='human' if DISPLAY else 'rgb',
apply_api_compatibility=True)
env = JoypadSpace(env, RIGHT_ONLY)

env = apply_wrappers(env)

agent = Agent(input_dims=env.observation_space.shape, num_actions=env.action_space.n)

if not SHOULD_TRAIN:
    folder_name = ""
    ckpt_name = ""
    agent.load_model(os.path.join("models", folder_name, ckpt_name))
    agent.epsilon = 0.2
    agent.eps_min = 0.0
    agent.eps_decay = 0.0

env.reset()
next_state, reward, done, trunc, info = env.step(action=0)
```

```

for i in range(NUM_OF_EPISODES):
    print("Episode:", i)
    done = False
    state, _ = env.reset()
    total_reward = 0
    while not done:
        a = agent.choose_action(state)
        new_state, reward, done, truncated, info = env.step(a)
        total_reward += reward

        if SHOULD_TRAIN:
            agent.store_in_memory(state, a, reward, new_state, done)
            agent.learn()

    state = new_state

    print("Total reward:", total_reward, "Epsilon:", agent.epsilon, "Size of replay buffer:",
len(agent.replay_buffer), "Learn step counter:", agent.learn_step_counter)

    if SHOULD_TRAIN and (i + 1) % CKPT_SAVE_INTERVAL == 0:
        agent.save_model(os.path.join(model_path, "model_" + str(i + 1) + "_iter.pt"))

    print("Total reward:", total_reward)

env.close()

```

Giải thích : Đoạn mã này thiết lập và chạy một tác nhân học tăng cường để chơi trò chơi Super Mario Bros, sử dụng PyTorch và môi trường **gym_super_mario_bros**. Dưới đây là giải thích chi tiết về từng phần của mã:

1. Thư viện sử dụng:

- **torch**: Để kiểm tra và sử dụng GPU nếu có.
- **gym_super_mario_bros**: Để tạo môi trường trò chơi Super Mario Bros.
- **agent**: Lớp tác nhân đã được định nghĩa trước đó.
- **nes_py.wrappers**: Để sử dụng bộ điều khiển Joypad.
- **wrappers**: Bộ bao bọc tùy chỉnh để xử lý môi trường.

- **os**: Để làm việc với hệ thống tệp.
- **utils**: Một module tùy chỉnh có thể chứa các hàm tiện ích như `get_current_date_time_string`.

2. Thiết lập đường dẫn mô hình:

- Tạo một thư mục để lưu trữ các mô hình đã huấn luyện, sử dụng ngày và giờ hiện tại để đặt tên.

3. Kiểm tra CUDA:

- Kiểm tra xem CUDA có khả dụng không và in ra tên thiết bị GPU nếu có.

4. Thiết lập môi trường:

- Tạo môi trường Super Mario Bros với chế độ điều khiển **RIGHT_ONLY**.
- Áp dụng các bộ bao bọc tùy chỉnh cho môi trường.

5. Khởi tạo tác nhân:

- Tạo một đối tượng **Agent** với kích thước đầu vào và số lượng hành động từ môi trường.

6. Tải mô hình nếu không huấn luyện:

- Nếu không huấn luyện (**SHOULD_TRAIN** là **False**), tải mô hình đã lưu và điều chỉnh các tham số epsilon để tác nhân không khám phá thêm.

7. Vòng lặp huấn luyện:

- Chạy qua một số tập (episodes) nhất định.
- Tại mỗi bước, tác nhân chọn một hành động dựa trên trạng thái hiện tại.
- Môi trường thực hiện hành động và trả về trạng thái mới, phần thưởng, và thông tin khác.
- Nếu đang huấn luyện, lưu trữ trải nghiệm vào bộ nhớ đệm và thực hiện một bước học.
- In ra thông tin về phần thưởng, epsilon, kích thước bộ nhớ đệm, và số bước học.

8. Lưu mô hình:

- Nếu đang huấn luyện, lưu mô hình sau mỗi **CKPT_SAVE_INTERVAL** tập.

9. Đóng môi trường:

- Đóng môi trường sau khi hoàn thành huấn luyện hoặc chơi.

6. simple_example.py

```
import gym_super_mario_bros

from gym_super_mario_bros.actions import RIGHT_ONLY

from nes_py.wrappers import JoypadSpace

ENV_NAME = 'SuperMarioBros-1-1-v0'

env = gym_super_mario_bros.make(ENV_NAME, render_mode='human', apply_api_compatibility=True)
env = JoypadSpace(env, RIGHT_ONLY)

done = False

env.reset()
counter = 0

while not done:

    # Only go right
    # action = RIGHT_ONLY.index(['right'])

    # Choose random action
    action = env.action_space.sample()

    _, _, done, _, _ = env.step(action)

    env.render()
```

Giải thích: Đoạn mã này tạo và chạy một môi trường trò chơi Super Mario Bros, trong đó Mario được điều khiển bằng cách chọn ngẫu nhiên các hành động từ không gian hành động.

1. Thư viện nhập khẩu:

- **gym_super_mario_bros**: Thư viện để tạo môi trường trò chơi Super Mario Bros.
- **nes_py.wrappers**: Để sử dụng bộ điều khiển Joypad.

2. Thiết lập môi trường:

- **ENV_NAME**: Tên của môi trường trò chơi, ở đây là "SuperMarioBros-1-1-v0", đại diện cho màn chơi đầu tiên của Super Mario Bros.
- **env = gym_super_mario_bros.make(...)**: Tạo môi trường trò chơi với chế độ hiển thị là 'human', cho phép bạn xem trò chơi khi nó đang chạy.
- **env = JoypadSpace(env, RIGHT_ONLY)**: Áp dụng bộ điều khiển Joypad với chế độ **RIGHT_ONLY**, giới hạn các hành động có thể thực hiện chỉ bao gồm các hành động di chuyển sang phải.

3. Vòng lặp trò chơi:

- **done = False**: Biến cờ để kiểm tra xem trò chơi đã kết thúc hay chưa.
- **env.reset()**: Đặt lại môi trường về trạng thái ban đầu.
- **while not done**: Vòng lặp chạy cho đến khi trò chơi kết thúc.
 - **action = env.action_space.sample()**: Chọn một hành động ngẫu nhiên từ không gian hành động của môi trường.
 - **_, _, done, _, _ = env.step(action)**: Thực hiện hành động đã chọn, cập nhật trạng thái của trò chơi và kiểm tra xem trò chơi đã kết thúc chưa.
 - **env.render()**: Hiển thị trò chơi trên màn hình.

7. simplified_main.py

```
import gym_super_mario_bros

from gym_super_mario_bros.actions import RIGHT_ONLY

from nes_py.wrappers import JoypadSpace

from wrappers import apply_wrappers
```

```
from agent import Agent

ENV_NAME = 'SuperMarioBros-1-1-v0'

SHOULD_TRAIN = True

DISPLAY = True

NUM_OF_EPISODES = 50_000

env = gym_super_mario_bros.make(ENV_NAME, render_mode='human' if DISPLAY else 'rgb_array',
apply_api_compatibility=True)

env = JoypadSpace(env, RIGHT_ONLY)

env = apply_wrappers(env)

agent = Agent(input_dims=env.observation_space.shape, num_actions=env.action_space.n)

for i in range(NUM_OF_EPISODES):

    done = False

    state, _ = env.reset()

    while not done:

        a = agent.choose_action(state)

        new_state, reward, done, truncated, info = env.step(a)

        agent.store_in_memory(state, a, reward, new_state, done)

        agent.learn()

    state = new_state
```

```
env.close()
```

Giải thích: Đoạn mã này thiết lập và chạy một tác nhân học tăng cường để chơi trò chơi Super Mario Bros, sử dụng một mô hình học sâu để học cách điều khiển Mario.

1. Thư viện nhập khẩu:

- **gym_super_mario_bros**: Thư viện để tạo môi trường trò chơi Super Mario Bros.
- **nes_py.wrappers**: Để sử dụng bộ điều khiển Joypad.
- **wrappers**: Một module tùy chỉnh có thể chứa các bộ bao bọc để xử lý môi trường.
- **agent**: Lớp tác nhân đã được định nghĩa trước đó.

2. Thiết lập môi trường:

- **ENV_NAME**: Tên của môi trường trò chơi, ở đây là "SuperMarioBros-1-1-v0".
- **SHOULD_TRAIN**: Biến cờ để xác định xem có nên huấn luyện tác nhân hay không.
- **DISPLAY**: Biến cờ để xác định xem có hiển thị trò chơi khi chạy hay không.
- **NUM_OF_EPISODES**: Số lượng tập (episodes) mà tác nhân sẽ chơi.

3. Khởi tạo môi trường:

- Tạo môi trường trò chơi với chế độ hiển thị là 'human' nếu **DISPLAY** là **True**, cho phép bạn xem trò chơi khi nó đang chạy.
- Áp dụng bộ điều khiển Joypad với chế độ **RIGHT_ONLY**, giới hạn các hành động có thể thực hiện chỉ bao gồm các hành động di chuyển sang phải.
- Áp dụng các bộ bao bọc tùy chỉnh cho môi trường thông qua hàm **apply_wrappers**.

4. Khởi tạo tác nhân:

- Tạo một đối tượng **Agent** với kích thước đầu vào và số lượng hành động từ môi trường.

5. Vòng lặp huấn luyện:

- Chạy qua một số tập (episodes) nhất định.

- Tại mỗi bước, tác nhân chọn một hành động dựa trên trạng thái hiện tại.
- Môi trường thực hiện hành động và trả về trạng thái mới, phần thưởng, và thông tin khác.
- Lưu trữ trải nghiệm vào bộ nhớ đệm của tác nhân và thực hiện một bước học.
- Cập nhật trạng thái hiện tại thành trạng thái mới.

6. Đóng môi trường:

- Đóng môi trường sau khi hoàn thành huấn luyện hoặc chơi.

8. utils.py

```
import time

import datetime


def get_current_date_time_string():

    return datetime.datetime.now().strftime("%Y-%m-%d-%H_%M_%S")


class Timer():

    def __init__(self):
        self.times = []


    def start(self):
        self.t = time.time()

    def print(self, msg=""):

        print(f"Time taken: {msg}", time.time() - self.t)

    def get(self):

        return time.time() - self.t
```

```
def store(self):  
  
    self.times.append(time.time() - self.t)  
  
  
def average(self):  
  
    return sum(self.times) / len(self.times)
```

Giải thích: Đoạn mã này định nghĩa một hàm tiện ích và một lớp để làm việc với thời gian trong Python.

1. Hàm `get_current_date_time_string`:

- Mục đích: Trả về chuỗi biểu diễn ngày và giờ hiện tại theo định dạng "YYYY-MM-DD-HH_MM_SS".
- Sử dụng `datetime.datetime.now()` để lấy thời gian hiện tại và `strftime` để định dạng thời gian thành chuỗi.

2. Lớp Timer:

- Mục đích: Cung cấp các phương thức để đo thời gian thực thi của các đoạn mã.
- **Thuộc tính:**
 - `self.times`: Danh sách để lưu trữ các khoảng thời gian đã đo được.
- **Phương thức:**
 - `start()`: Bắt đầu đo thời gian bằng cách lưu thời gian hiện tại vào `self.t`.
 - `print(msg="")`: In ra thời gian đã trôi qua kể từ khi `start()` được gọi, kèm theo một thông điệp tùy chọn.
 - `get()`: Trả về thời gian đã trôi qua kể từ khi `start()` được gọi.
 - `store()`: Lưu thời gian đã trôi qua vào danh sách `self.times`.
 - `average()`: Tính và trả về thời gian trung bình của tất cả các khoảng thời gian đã lưu trong `self.times`.

Lớp **Timer** rất hữu ích để đo lường hiệu suất của các đoạn mã, đặc biệt là trong các ứng dụng cần tối ưu hóa thời gian thực thi.

Hàm **get_current_date_time_string** có thể được sử dụng để tạo tên tệp hoặc thư mục dựa trên thời gian hiện tại, giúp dễ dàng quản lý và sắp xếp dữ liệu theo thời gian.

9. wrappers.py

```
import numpy as np

from gym import Wrapper

from gym.wrappers import GrayScaleObservation, ResizeObservation, FrameStack

class SkipFrame(Wrapper):

    def __init__(self, env, skip):
        super().__init__(env)
        self.skip = skip

    def step(self, action):
        total_reward = 0.0
        done = False
        for _ in range(self.skip):
            next_state, reward, done, trunc, info = self.env.step(action)
            total_reward += reward
            if done:
                break
        return next_state, total_reward, done, trunc, info

def apply_wrappers(env):
    env = SkipFrame(env, skip=4) # Num of frames to apply one action to
```

```

env = ResizeObservation(env, shape=84) # Resize frame from 240x256 to 84x84

env = GrayScaleObservation(env)

env = FrameStack(env, num_stack=4, lz4_compress=True) # May need to change lz4_compress to False if
issues arise

return env

```

Giải thích : Đoạn mã này định nghĩa một lớp và một hàm để áp dụng các bộ bao bọc (wrappers) cho môi trường trong thư viện Gym, nhằm xử lý và tối ưu hóa dữ liệu đầu vào cho các tác vụ học tăng cường.

1. Thư viện sử dụng:

- **numpy**: Thư viện cho các phép toán số học, mặc dù không được sử dụng trực tiếp trong đoạn mã này.
- **gym**: Thư viện để tạo và quản lý các môi trường mô phỏng.
- **gym.wrappers**: Các bộ bao bọc có sẵn trong Gym để xử lý dữ liệu đầu vào.

2. Lớp SkipFrame:

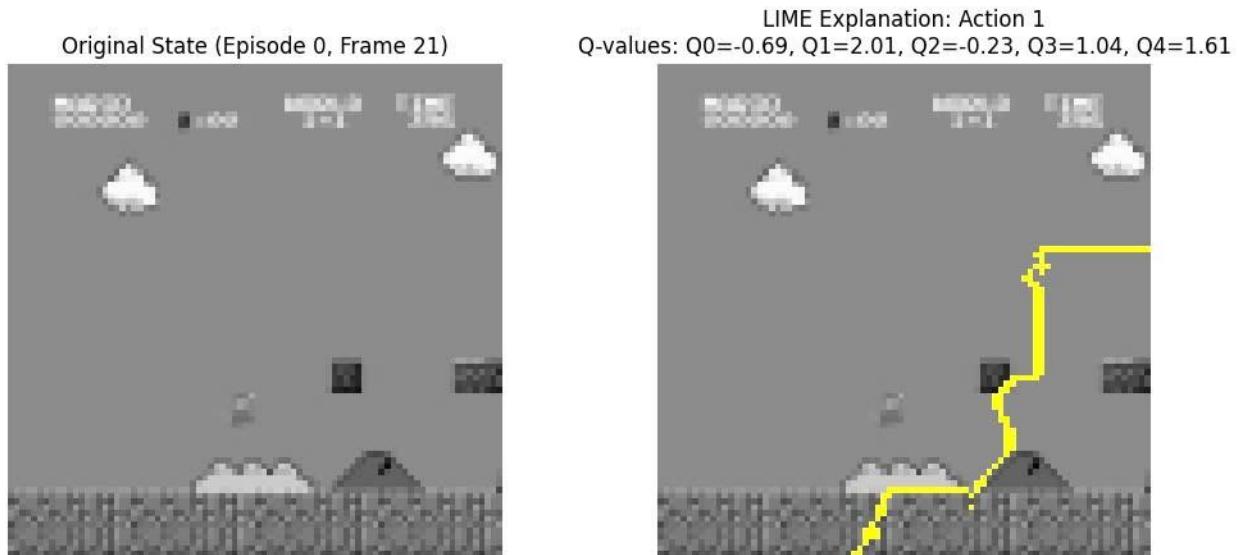
- Mục đích: Giảm tần suất cập nhật môi trường bằng cách bỏ qua một số khung hình nhất định, giúp tăng tốc độ xử lý.
- **Khởi tạo (__init__ method):**
 - **env**: Môi trường Gym cần được bao bọc.
 - **skip**: Số lượng khung hình sẽ bị bỏ qua giữa các lần cập nhật.
- **Phương thức step:**
 - Thực hiện hành động **action** trong môi trường nhiều lần (theo số lượng **skip**), cộng dồn phần thưởng và kiểm tra xem trò chơi đã kết thúc chưa.
 - Trả về trạng thái tiếp theo, tổng phần thưởng, và các thông tin khác.

3. Hàm apply_wrappers:

- Mục đích: Áp dụng một chuỗi các bộ bao bọc cho môi trường để chuẩn bị dữ liệu đầu vào cho mô hình học máy.
- **Các bộ bao bọc được áp dụng:**
 - **SkipFrame**: Bỏ qua 4 khung hình giữa các lần cập nhật.

- **ResizeObservation:** Thay đổi kích thước khung hình từ 240x256 thành 84x84, giúp giảm kích thước dữ liệu và tăng tốc độ xử lý.
- **GrayScaleObservation:** Chuyển đổi khung hình thành ảnh xám, giảm số lượng kênh màu từ 3 (RGB) xuống 1, giúp giảm độ phức tạp của dữ liệu.
- **FrameStack:** Xếp chồng 4 khung hình liên tiếp lại với nhau, tạo ra một chuỗi khung hình để mô hình có thể nắm bắt được động lực học của môi trường.

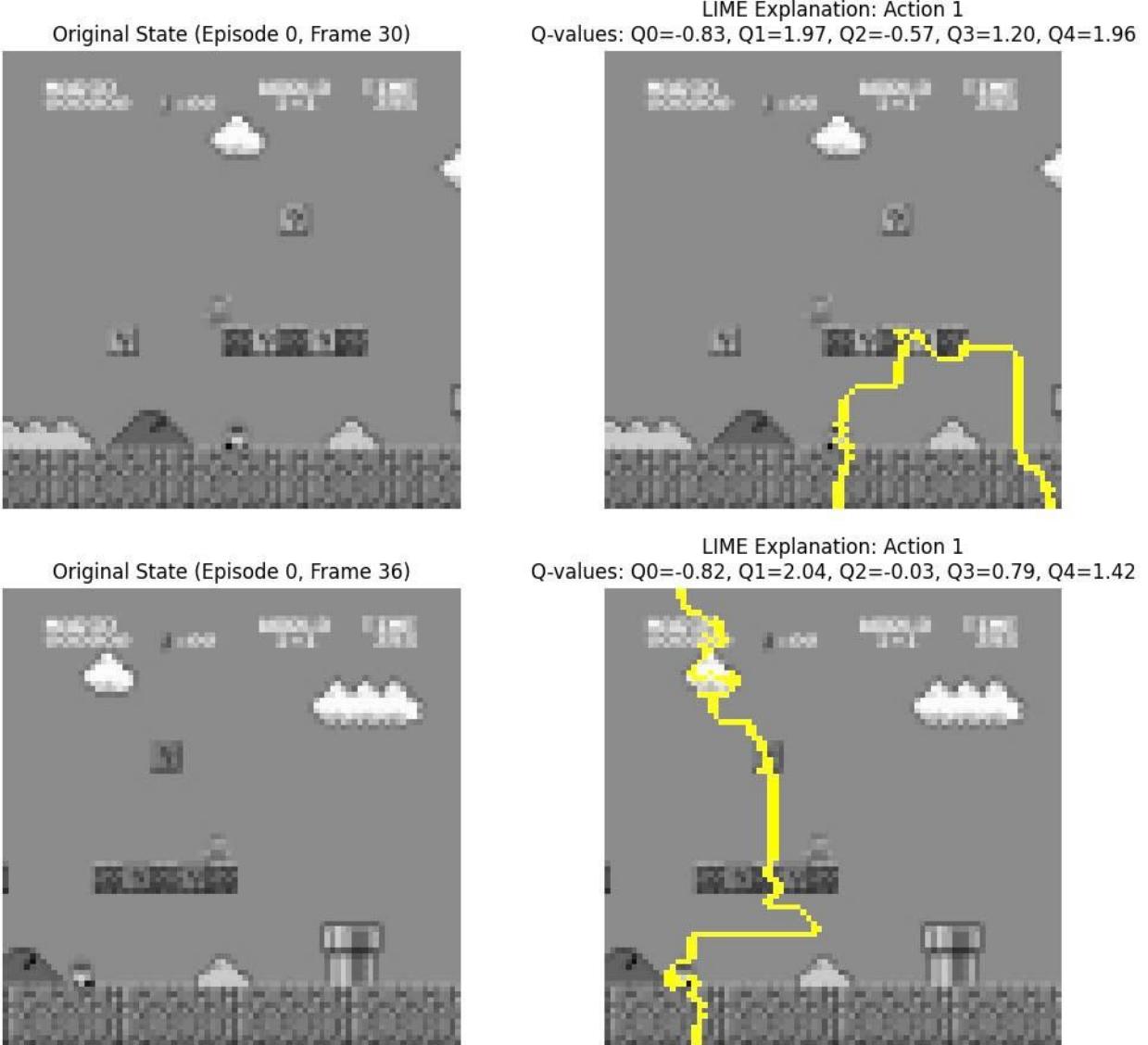
Kết quả:



Bên trái (Original State): Hiển thị trạng thái gốc của trò chơi, gồm một đám mây, ngọn núi, và một khối gạch treo. Mario không xuất hiện, có thể do camera di chuyển.

Bên phải (LIME Explanation): LIME làm nổi bật (vùng màu vàng) khu vực quanh khối gạch và ngọn núi, cho thấy đây là các yếu tố ảnh hưởng đến quyết định của agent khi chọn Action 1 (nhảy).

Q-values: $Q0=-0.69$, $Q1=2.01$, $Q2=-0.23$, $Q3=-1.04$, $Q4=1.61$. $Q1$ (Action 1) có giá trị cao nhất, phù hợp với hành động nhảy được chọn.



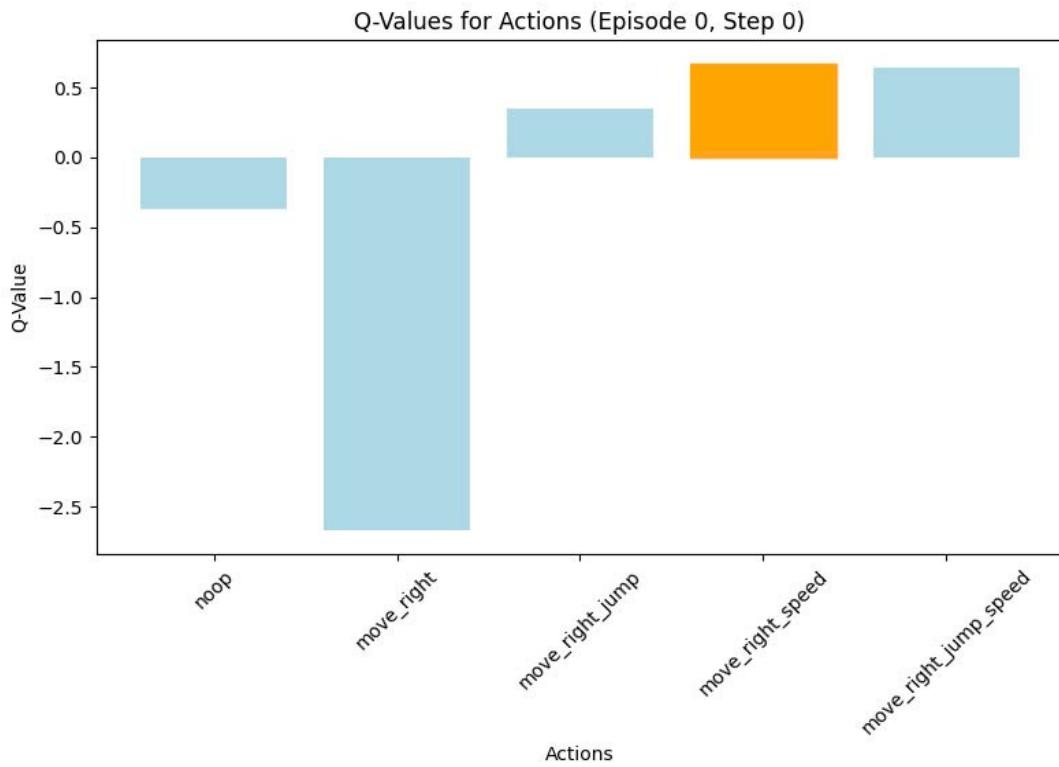
Frame 30:

- **Original State:** Hiển thị trạng thái gốc với đám mây, ngọn núi, và khói gạch treo. Mario không xuất hiện, có thể do camera di chuyển.
- **LIME Explanation:** LIME làm nổi bật (vùng màu vàng) khu vực quanh khói gạch, cho thấy đây là yếu tố chính ảnh hưởng đến quyết định nhảy (Action 1).
- **Q-values:** Q0=-0.83, Q1=1.97, Q2=-0.57, Q3=1.20, Q4=1.96. Q1 (Action 1) cao nhất, phù hợp với hành động nhảy.

Frame 36:

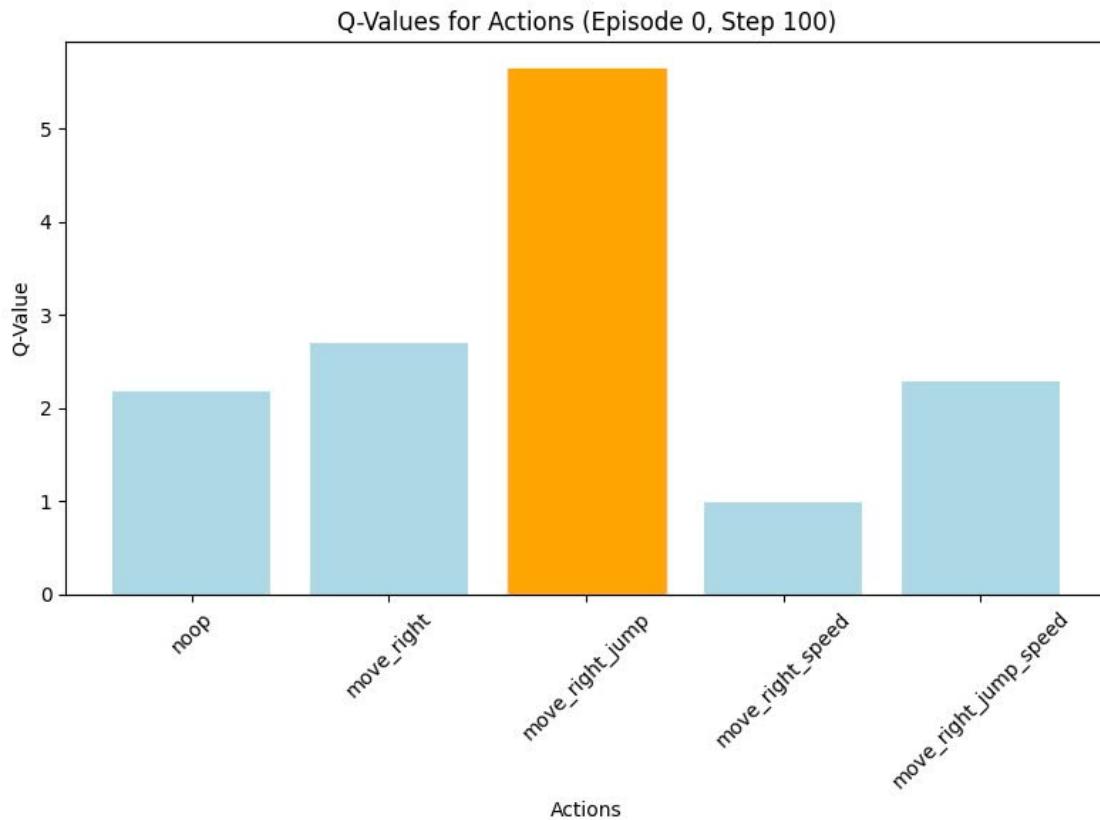
- **Original State:** Hiển thị đám mây, ngọn núi, và một ống nước. Mario vẫn không xuất hiện.
- **LIME Explanation:** LIME làm nổi bật khu vực quanh ống nước, cho thấy agent tập trung vào ống nước khi quyết định nhảy (Action 1).
- **Q-values:** $Q_0 = -0.82$, $Q_1 = 2.04$, $Q_2 = -0.03$, $Q_3 = -0.79$, $Q_4 = 1.42$. Q_1 (Action 1) cao nhất, xác nhận hành động nhảy.

Agent nhảy (Action 1) để tránh chướng ngại vật (khối gạch ở frame 30, ống nước ở frame 36), dựa trên các vùng được LIME làm nổi bật. Q-values cho thấy agent đưa ra quyết định hợp lý, nhưng việc Mario không xuất hiện trong khung hình hạn chế khả năng giải thích chi tiết. LIME chứng minh agent không hành động ngẫu nhiên mà dựa trên các đặc điểm môi trường cụ thể.



Ảnh 2.1

- Q-Values for Actions (Episode 0, Step 0)
 - Đây là trạng thái **ban đầu** khi mô hình bắt đầu học (bước 0, tập 0).
- **Trục X - Actions:** Gồm 5 hành động khả thi:
 1. noop – Không làm gì.
 2. move_right – Di chuyển sang phải.
 3. move_right_jump – Vừa di chuyển phải vừa nhảy.
 4. move_right_speed – Di chuyển phải nhanh.
 5. move_right_jump_speed – Vừa nhảy vừa di chuyển phải nhanh.



Ảnh 2.2

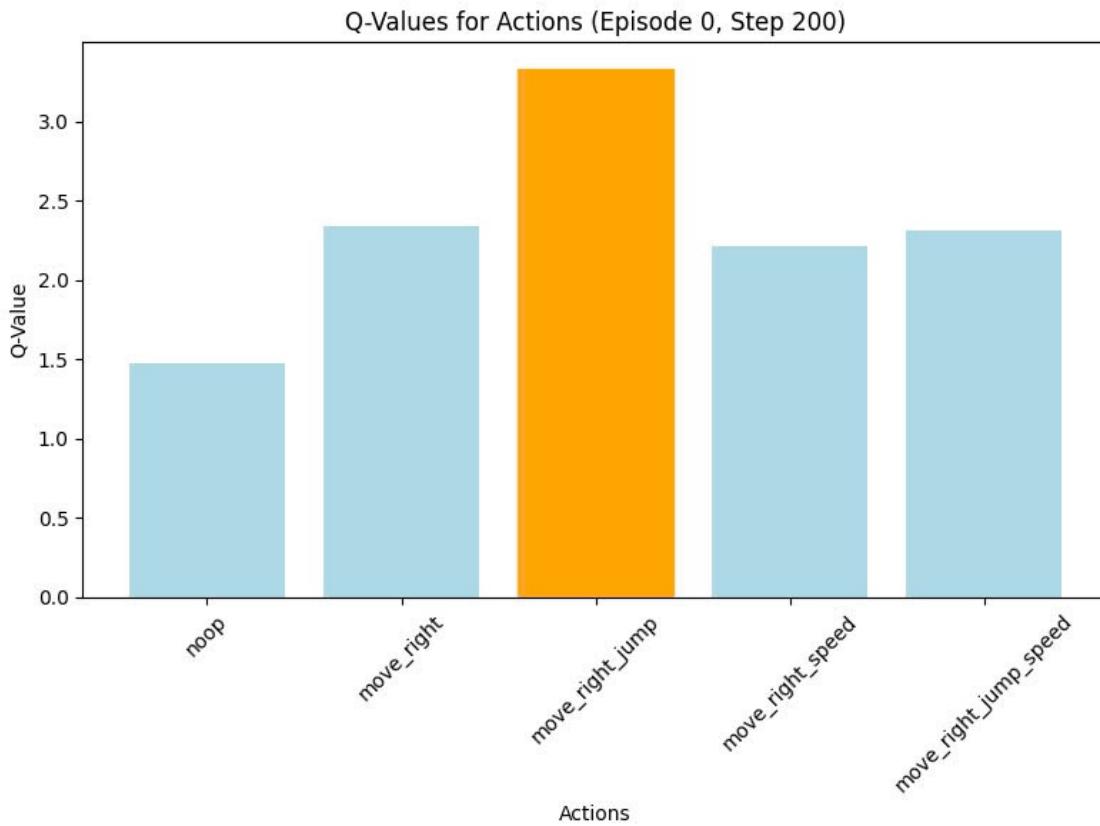
- **Q-Values for Actions (Episode 0, Step 100)**
 - Đây là thời điểm sau khi mô hình đã được huấn luyện qua **100 bước đầu tiên**.
 - Vẫn là **Episode đầu tiên (số 0)**.
- Các hành động và Q-value tương ứng:

Hành động	Q-value (trước lượng)	Màu sắc	Ý nghĩa
noop	~2.2	Xanh nhạt	Không làm gì có mức đánh giá tạm ổn.
move_right	~2.8	Xanh nhạt	Di chuyển sang phải là hành động tiềm năng.

move_right_jump	~5.6	Cam	Đây là hành động tốt nhất hiện tại , mô hình đánh giá rất cao.
move_right_speed	~1.0	Xanh nhạt	Kém hiệu quả so với các hành động khác.
move_right_jump_ speed	~2.3	Xanh nhạt	Tốt hơn speed, kém hơn jump.

Giải thích ý nghĩa mô hình học được gì:

- **Hành động tối ưu:** move_right_jump được tô màu cam, có Q-value cao nhất (~5.6) → mô hình đánh giá **đây là hành động mang lại phần thưởng cao nhất** trong bước 100.
- Đây là **sự cải thiện rõ rệt so với bước 0 và bước 400** mà bạn đã gửi ở các ảnh trước:
 - Bước 0: Q-values nhỏ, có cả giá trị âm.
 - Bước 100: Q-values đã tăng đáng kể, và có sự **phân biệt rõ ràng giữa các hành động**.
- Điều này cho thấy mô hình **đang học dần và bắt đầu đưa ra lựa chọn hợp lý hơn**.



Ảnh 2.3

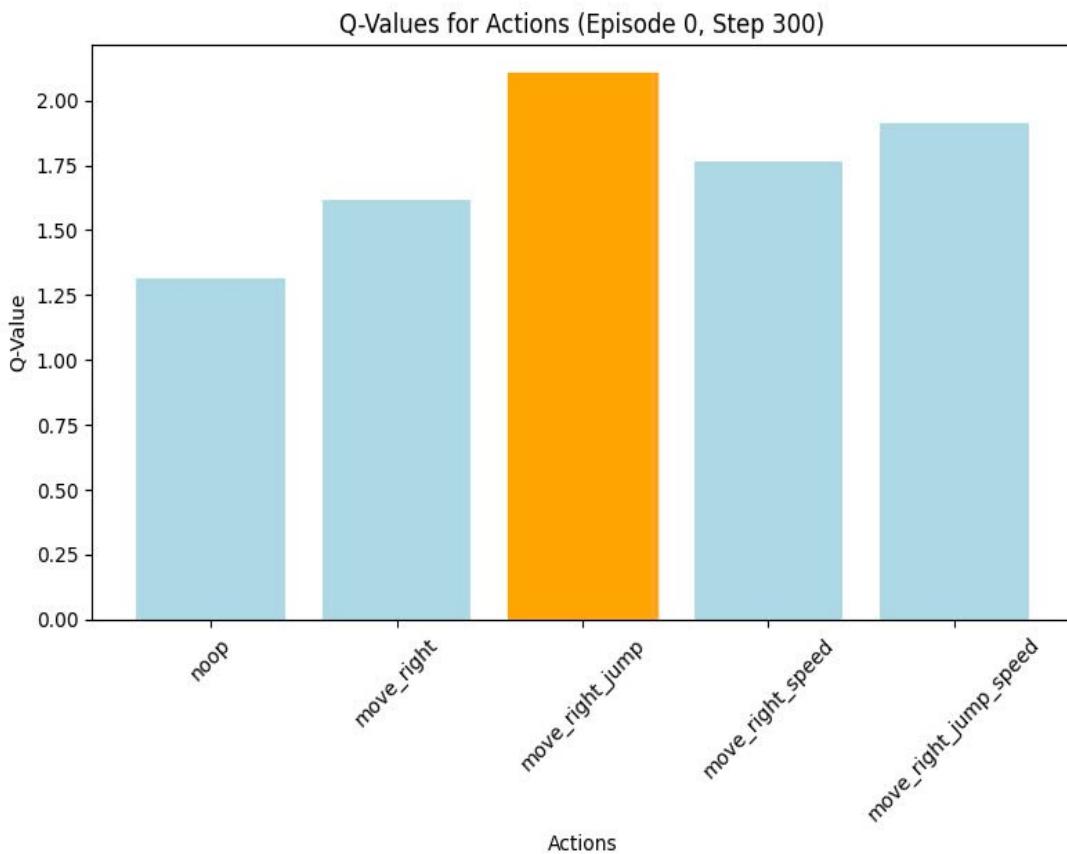
- **Q-Values for Actions (Episode 0, Step 200)**
 - Đây là thời điểm ở giữa quá trình huấn luyện của Episode đầu tiên (0) tại bước thứ 200.
- **Các hành động và Q-value tương ứng:**

Hành động	Q-value (ước lượng)	Màu sắc	Ý nghĩa
noop	~1.5	Xanh nhạt	Không làm gì, ít có giá trị.

move_right	~2.3	Xanh nhạt	Di chuyển sang phải vẫn là hành động tốt.
move_right_jump	~3.3	Cam	Hành động tốt nhất hiện tại, có Q-value cao nhất.
move_right_speed	~2.2	Xanh nhạt	Di chuyển nhanh, có giá trị nhưng không phải tối ưu.
move_right_jump_speed	~2.3	Xanh nhạt	Kết hợp nhiều hành động, gần bằng với move_right.

Giải thích ý nghĩa mô hình học được gì ở bước 200:

- **Hành động tốt nhất:** move_right_jump tiếp tục được mô hình đánh giá cao nhất, điều này thể hiện rằng:
 - Trong môi trường, hành động **nhảy và đi sang phải** thường mang lại phần thưởng cao nhất.
 - Mô hình đang dần học được điều này thông qua cập nhật Q-values.
- Các hành động như move_right, move_right_speed, và move_right_jump_speed cũng được đánh giá tích cực → có thể do các tình huống mà hành động này dẫn đến trạng thái "tốt".
- **Không có giá trị âm** → mô hình đã vượt qua giai đoạn "học thử sai" ban đầu và đang dần tập trung vào những hành động hiệu quả.



Ảnh 2.4

- **Q-Values for Actions (Episode 0, Step 300)**

- Tại bước 300 trong Episode đầu tiên.
- Đây là một snapshot thể hiện mức độ kỳ vọng phần thưởng (Q-value) của mỗi hành động tại thời điểm hiện tại.

Hành động	Q-value (ước lượng)	Màu sắc	Ý nghĩa
noop	~1.3	Xanh nhạt	Không làm gì – ít giá trị nhất.
move_right	~1.6	Xanh nhạt	Di chuyển sang phải – tạm ổn.

move_right_jump	~2.1	Cam	Hành động tốt nhất hiện tại (Q-value cao nhất).
move_right_speed	~1.8	Xanh nhạt	Hành động có tốc độ – giá trị tốt, tiệm cận hành động tốt nhất.
move_right_jump_speed	~1.9	Xanh nhạt	Nhảy + chạy nhanh – hiệu quả, gần đạt mức tối ưu.

Phân tích mô hình học được gì tại bước này:

- **Hành động tối ưu hiện tại:** move_right_jump tiếp tục được đánh giá là tốt nhất (màu cam), nhưng Q-value **không vượt trội** so với các hành động khác như:
 - move_right_jump_speed
 - move_right_speed

→ Điều này cho thấy mô hình bắt đầu **cân nhắc kỹ hơn** giữa các hành động có vẻ tốt ngang nhau.

Ý nghĩa Q-values tiệm cận nhau:

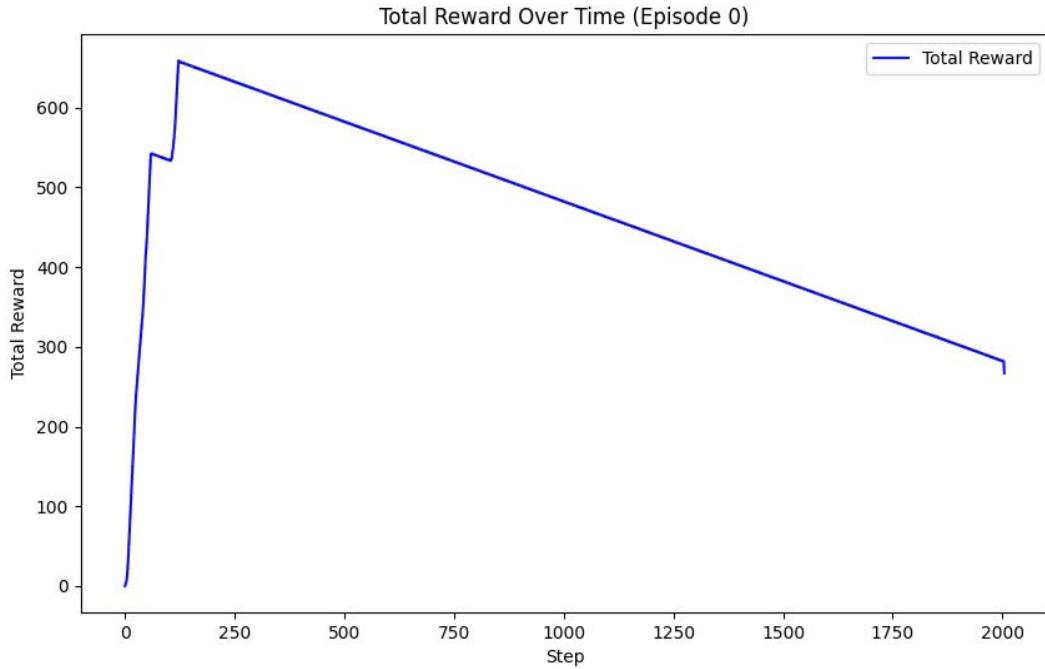
- Q-values của các hành động bắt đầu xích lại gần nhau, chứng tỏ:
 - Mô hình đã tích lũy được kha khá kinh nghiệm.
 - Đang **khám phá (exploration)** và **khai thác (exploitation)** cùng lúc.
 - Cân nhắc rằng có thể nhiều hành động đều có giá trị tương đương, tùy từng tình huống cụ thể trong môi trường.

So sánh với các bước trước:

Bước	Hành động tốt nhất	Q-value cao nhất	Ghi chú chính
0	move_right_speed	~0.65	Khởi đầu học, chưa rõ hành động nào tốt.
100	move_right_jump	~5.6	Mô hình bắt đầu nhận ra hành động hiệu quả.
200	move_right_jump	~3.3	Vẫn giữ vững vị trí tối ưu, nhưng Q-value cân bằng hơn.
300	move_right_jump	~2.1	Tiếp tục tối ưu, nhưng nhiều hành động khác cũng được đánh giá cao.

Tổng kết:

- move_right_jump tiếp tục là hành động được ưu tiên, nhưng sự chênh lệch giữa các hành động đã giảm.
- Mô hình đang **tinh chỉnh** hiểu biết của mình về môi trường, không chỉ tối ưu một hành động duy nhất mà còn đang học **sự linh hoạt** khi chọn lựa hành động dựa vào trạng thái cụ thể.



Ảnh 3.1

Ý nghĩa của biểu đồ:

- Trục X (Step): Số bước mà agent đã thực hiện (từ 0 đến khoảng 2000).
- Trục Y (Total Reward): Tổng phần thưởng tích lũy được đến thời điểm đó.
- Đường màu xanh: Biểu diễn sự thay đổi phần thưởng theo thời gian.

Giải thích theo từng giai đoạn:

⚡ Giai đoạn 1: Tăng nhanh (Step 0 → ~150)

- Total Reward tăng rất nhanh → agent đang làm những hành động đúng, nhận được nhiều phần thưởng tích cực.

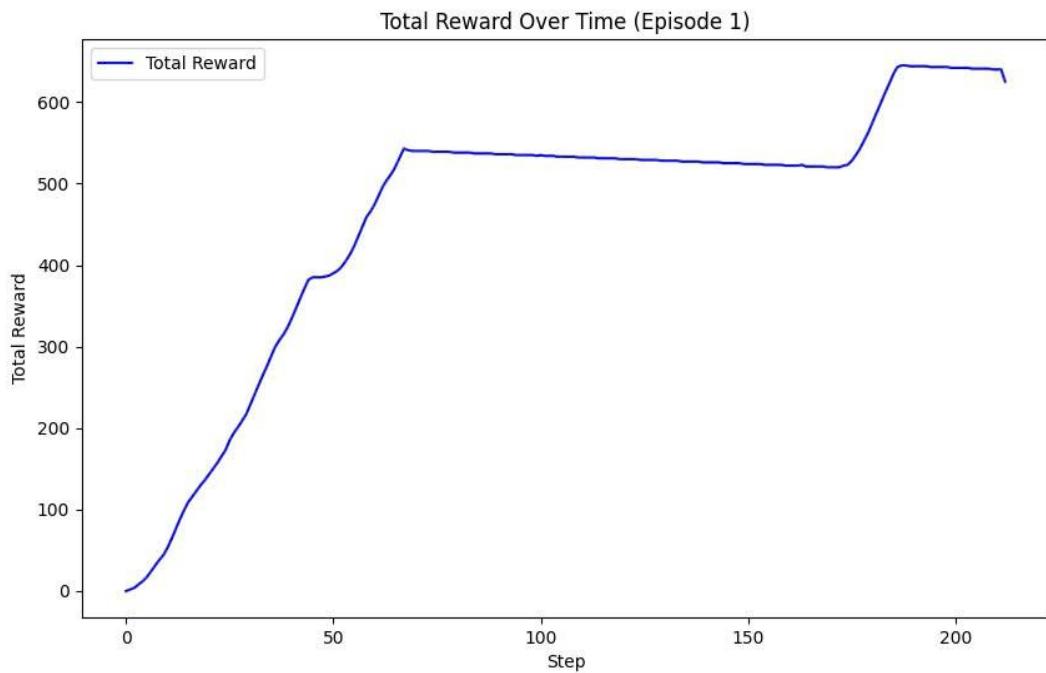
- Đây có thể là thời điểm agent đi đúng hướng, vượt qua chướng ngại vật hoặc đạt mục tiêu ngắn hạn.

Giai đoạn 2: Đạt đỉnh cao (~Step 160–200)

- Tổng reward đạt đỉnh khoảng trên 650.
- Có một đoạn hơi chững lại → có thể do một vài hành động không hiệu quả, hoặc agent bị mắc kẹt ngắn hạn.
- Đây là điểm cao nhất mà agent đạt được trong tập này.

Giai đoạn 3: Giảm dần đều (Step 200 → 2000)

- Tổng reward giảm tuyến tính → mỗi hành động tiếp theo nhận reward âm (negative reward).
- Có thể agent đang thực hiện các hành động kém hiệu quả hoặc bị kẹt (ví dụ như nhảy tại chỗ, đi ngược lại, không đạt mục tiêu).
- Nếu phần thưởng âm này xuất hiện đều → khả năng là môi trường đang phạt agent vì không tiến bộ (ví dụ: bị đứng yên quá lâu, hoặc đi sai đường).



Ảnh 3.2

Phân tích chi tiết:

⚡ Giai đoạn 1: Tăng mạnh (Step 0 → ~65)

- Agent nhận được phần thưởng liên tục, biểu đồ tăng nhanh và mượt.
- Điều này cho thấy agent đang hành động tốt, di chuyển đúng hướng, tránh chướng ngại vật hoặc đạt các mục tiêu phụ.
- Giai đoạn này kết thúc với reward khoảng 540+.

⌚ Giai đoạn 2: Chững lại nhẹ (~Step 65 → 175)

- Reward gần như không thay đổi hoặc giảm rất nhẹ → hành động của agent không nhận thêm reward hoặc bị trừ ít reward.

- Có thể agent đang đứng yên, hoặc di chuyển không mang lại lợi ích, ví dụ như chạy tại chỗ, nhảy sai thời điểm.
- Tuy không giảm nhiều như Episode 0, nhưng cũng không có tiến triển.

Giai đoạn 3: Tăng đột biến (Step 175 → ~190)

- Một đoạn tăng rất nhanh → agent thực hiện được hành động quan trọng, ví dụ: vượt chướng ngại lớn, tiêu diệt kẻ địch, đạt checkpoint,...
- Tổng reward đạt đỉnh ~650 tại đây.

Giai đoạn 4: Tụt nhẹ cuối đoạn (~Step 190 → End)

- Có một đoạn tụt nhẹ phần thưởng → có thể agent gặp lỗi nhỏ hoặc hành động kém hiệu quả (nhưng không nghiêm trọng).
- Tổng thể vẫn duy trì ở mức cao.

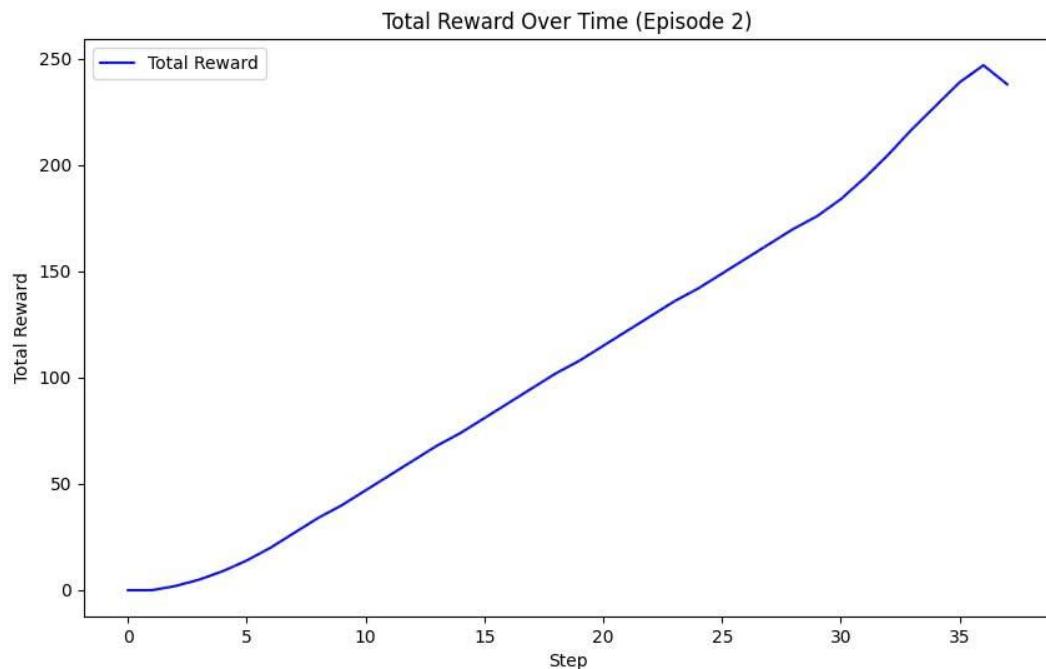
So sánh nhanh với Episode 0:

Đặc điểm	Episode 0	Episode 1
Tăng ban đầu	Nhanh và cao	Nhanh và cao
Giữa episode	Giảm dần đều (reward âm nhiều)	Chững lại nhẹ, vẫn giữ ổn
Cuối episode	Tụt mạnh về 300	Giữ mức cao (khoảng 640)

Tổng thể

Chưa ổn định

Cải thiện rõ
rệt, tốt hơn



Ảnh 3.3

Phân tích chi tiết:

● Giai đoạn 1: Bắt đầu chậm (Step 0 → ~5)

- Tổng reward tăng nhẹ → agent mới bắt đầu tương tác môi trường.
- Có thể đang dò đường, thử hành động → chưa tối ưu.

● Giai đoạn 2: Tăng đều (Step 5 → 30)

- Reward tăng tuyến tính và đều đặn → agent đang hành động liên tục tốt.
- Giai đoạn này cho thấy học được một chiến lược ổn định, di chuyển đúng, tránh được kẻ địch và bẫy.

▲ Giai đoạn 3: Tăng nhanh (Step 30 → 36)

- Tổng reward tăng nhanh hơn → có thể agent vừa đạt checkpoint, tiêu diệt kẻ địch hoặc ăn vật phẩm có giá trị cao.

▼ Giai đoạn 4: Tụt nhẹ cuối (Step 36 → 37)

- Reward giảm nhẹ ở bước cuối → hành động cuối cùng gây mất điểm, ví dụ như:

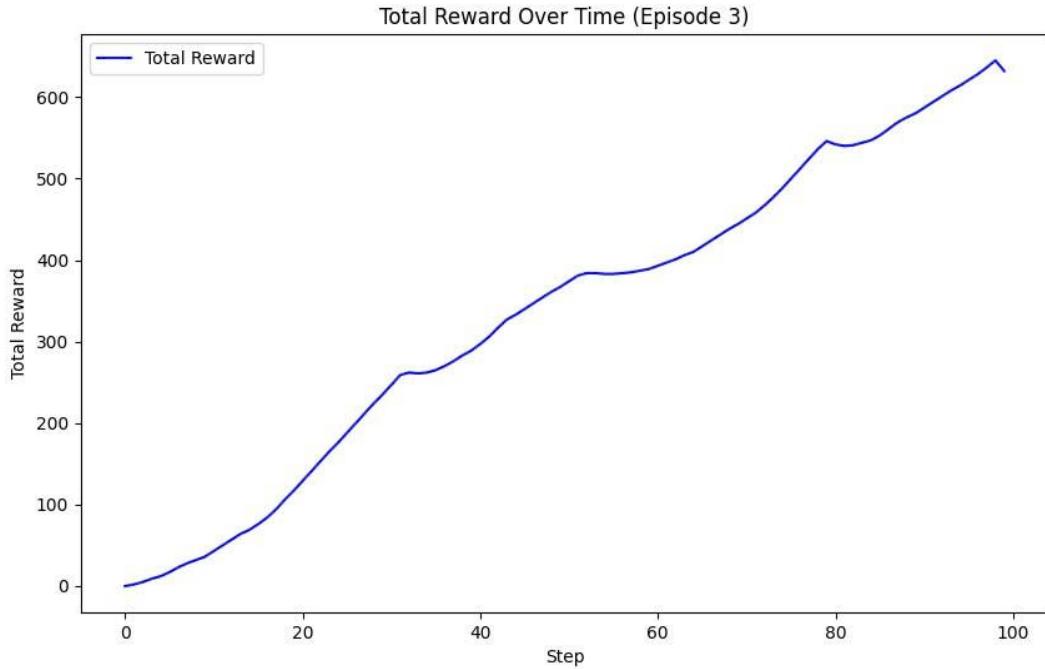
- Bị va chạm,
- Rơi xuống hố,
- Hết thời gian,...

So sánh với các episode trước:

Tập	Tổng Reward Cao Nhất	Chiều dài Episode	Mức độ ổn định
0	~650 rồi giảm về ~290	Rất dài (~2000 bước)	Không ổn định
1	~650 và giữ ổn định	Trung bình (~220 bước)	Khá tốt
2	~245 và kết thúc sớm	Rất ngắn (~37 bước)	Rất ổn định

Episode 2 có reward thấp hơn nhưng:

- Ngắn hơn nhiều → hành động hiệu quả ngay từ đầu.
- Tăng đều, rõ ràng và sạch sẽ → policy đang tiến bộ.



Ảnh 3.4

Phân tích chi tiết theo từng giai đoạn:



Giai đoạn 1: Tăng đều (Step 0 → 30)

- Reward tăng dần và đều → agent đang di chuyển tốt, tránh va chạm, thực hiện hành động hiệu quả.
- Đây là dấu hiệu của một chính sách đang ổn định và có hiệu suất.



Giai đoạn 2: Giảm tốc độ tăng (Step 30 → 60)

- Reward vẫn tăng nhưng có chậm lại, đặc biệt quanh step 55–60 có đoạn gần như đi ngang (plateau).
 - Có thể agent rơi vào trạng thái lặp lại hành động kém hiệu quả.
 - Hoặc gặp chướng ngại khó vượt, dẫn đến không có reward thêm trong vài bước.

Giai đoạn 3: Bùng nổ (Step 60 → 80)

- Reward tăng nhanh → agent có thể:
 - Đã vượt qua được đoạn khó,
 - Gặp nhiều vật phẩm/điểm số,
 - Thực hiện chuỗi hành động tốt (combo jump, tiêu diệt kẻ thù, vượt vật cản).

Giai đoạn 4: Tăng ổn định và kết thúc (Step 80 → 98)

- Reward tiếp tục tăng, không tụt → tập kết thúc khi agent đạt ~640 reward.

So sánh với các episode trước:

Tập	Max Reward	Steps	Nhận xét ngắn gọn
0	~650 → giảm về ~290	~2000	Dài nhưng không hiệu quả
1	~650 ổn định	~220	Chiến lược tốt
2	~245	~37	Ngắn, hiệu quả rõ ràng
3	~640 (rất tốt)	~98	Hiệu quả cao, ổn định

Nhận xét tổng quan Episode 3:

- Đây là một **episode hiệu quả nhất** tính đến hiện tại:
→ reward gần bằng max của episode 0 và 1, **nhiều với ít bước hơn rất** (chỉ ~98 bước).
- Biểu đồ cho thấy:
 - Agent **đang học rất tốt**,
 - Chiến lược đã gần hội tụ (tăng đều, ít dao động),
 - Không có phần thưởng âm hay mất điểm ở cuối như một số tập trước.

TÀI LIỆU THAM KHẢO

- [1]. **Source code** - <https://github.com/DogguG/Reinforcement-Learning> - 3/2025
- [2]. **Chung Pham Van** - [Deep Learning] Tìm hiểu về mạng tích chập (CNN) [6].
Monte Carlo Tree Search <https://www.geeksforgeeks.org/ml-monte-carlo-tree-search-mcts/> - 23/5/2023
- [3]. **Reinforcement Learning** <https://www.geeksforgeeks.org/what-is-reinforcement-learning/> - 24/2/2025
- [4]. **Markov decision process** <https://www.geeksforgeeks.org/markov-decision-process/> - 5/7/2024
- [5]. **Train AI to Beat Super Mario Bros! || Reinforcement Learning Completely from Scratch** - [Train AI to Beat Super Mario Bros! || Reinforcement Learning Completely from Scratch](#) - 2/10/2023
- [6]. **SHAP, LIME** <https://codelearn.io/sharing/giai-ma-thuat-toan-lime-explainable-ai> - 23/3/2025
- [7]. **SHAP, LIME** <https://www.markovml.com/blog/lime-vs-shap> - 23/3/2025