BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC XÂY DỰNG HÀ NỘI

---------- ----------

ĐỒ ÁN TỐT NGHIỆP ĐẠI HỌC

ỨNG DỤNG THUẬT TOÁN PROXIMAL POLICY OPTIMIZATION TRONG MÔ PHỎNG ROBOT CÂN BẰNG HAI CHÂN TRÊN MUJOCO

Ngành: Khoa học máy tính

Mã số: 608824

NGUYỄN HOÀNG NGUYÊN

HÀ NỘI - 2025

BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC XÂY DỰNG HÀ NỘI

ĐỒ ÁN TỐT NGHIỆP ĐẠI HỌC

ỨNG DỤNG THUẬT TOÁN PROXIMAL POLICY OPTIMIZATION TRONG MÔ PHỎNG ROBOT CÂN BẰNG HAI CHÂN TRÊN MUJOCO

Giáo viên hướng dẫn: TS. Roãn Thị Ngân

Sinh viên thực hiện: NGUYỄN HOÀNG NGUYÊN

Mã số sinh viên: 147964

Khoa/ngành: Công nghệ thông tin/Khoa học máy tính

Lớp/Khoá: 64CS2/64

Bậc/Loại hình đào tạo: Đại học-B7/Chính quy - CDIO

HÀ NỘI - THÁNG 1 - NĂM 2025

LỜI CAM ĐOAN

Tôi tên là NGUYỄN HOÀNG NGUYÊN, mã số sinh viên 147964, sinh viên lớp 64CS2, khoá 64, giáo viên hướng dẫn là TS. ROÃN THỊ NGÂN. Tôi xin cam đoan toàn bộ nội dung được trình bày trong đồ án "**Ứng dụng thuật toán Proximal Policy Optimization trong mô phỏng robot cân bằng hai chân trên Mujoco**" là kết quả quá trình tìm hiểu và nghiên cứu của tôi. Các dữ liệu được nêu trong đồ án là hoàn toàn trung thực, phản ánh đúng kết quả thí nghiệm, đo đạc thực tế. Mọi thông tin trích dẫn đều tuân thủ các quy định về sở hữu trí tuệ; các tài liệu tham khảo được liệt kê rõ ràng. Tôi xin chịu hoàn toàn trách nhiệm với những nội dung được viết trong đồ án này.

...., ngày ...., tháng ...., năm .....

Người cam đoan

NGUYỄN HOÀNG NGUYÊN

MỤC LỤC

DANH MỤC HÌNH ẢNH

DANH MỤC BẢNG BIỂU

DANH MỤC CÔNG THỨC

MỞ ĐẦU

1. Lý do chọn để tài

Trong bối cảnh toàn cầu hóa và công nghiệp hóa hiện nay, ngành robotics học đang nổi lên như một lĩnh vực quan trọng với tiềm năng phát triển vượt bậc. Theo thống kê từ Statista, thị trường robotics AI tại Việt Nam dự kiến sẽ đạt mức 42,69 triệu USD vào năm 2024, với tốc độ tăng trưởng hàng năm (CAGR) là 25,55% trong giai đoạn 2024-2030. Đến năm 2030, khối lượng thị trường dự kiến sẽ chạm ngưỡng 167,20 triệu USD. Trên phạm vi toàn cầu, con số này còn ấn tượng hơn với quy mô thị trường đạt 17,03 tỷ USD vào năm 2024 và tăng trưởng đến 64,35 tỷ USD vào năm 2030. Mặc dù tỷ lệ của thị trường robot AI so với tổng quy mô toàn cầu còn khiêm tốn, chỉ chiếm 0,017% trong số khoảng 100 nghìn tỷ USD, sự phát triển nhanh chóng này cho thấy tiềm năng khổng lồ của công nghệ robot kết hợp trí tuệ nhân tạo trong tương lai.

Robot cân bằng hai chân không chỉ thể hiện ưu điểm vượt trội khi có khả năng hoạt động trong những môi trường không bằng phẳng, mà còn có tính linh hoạt cao, tương thích với nhiều tình huống khác nhau, mà các loại robot truyền thống gặp khó khăn. Những robot này có thể di chuyển trên địa hình phức tạp, leo cầu thang, hoạt động trong không gian chật hẹp, và thậm chí thực hiện các nhiệm vụ cần sự chính xác cao như mang vác đồ vật hay hỗ trợ người khuyết tật trong các hoạt động thường nhật. Không chỉ giới hạn trong lĩnh vực dân sự, robot cân bằng hai chân còn được kỳ vọng sẽ đóng vai trò quan trọng trong các lĩnh vực cấp thiết như cứu hộ, chăm sóc sức khỏe và ứng phó khẩn cấp. Chúng giúp giảm thiểu rủi ro cho con người trong những tình huống nguy hiểm và tăng cường hiệu quả công việc, đặc biệt trong những môi trường như khu vực thiên tai hay nhà máy công nghiệp.

Thế nhưng, sự phát triển và kiểm thử các hệ thống điều khiển cho loại robot này vẫn đang gặp nhiều thách thức lớn. Do tính phức tạp của hệ động lực học và các yếu tố môi trường, chi phí phát triển và thử nghiệm robot thực tế trở

nên rất cao. Theo báo cáo từ McKinsey, các công ty có thể tiết kiệm tới 30-40% chi phí nếu áp dụng các mô phỏng ảo trong quá trình thiết kế và thử nghiệm sản phẩm. Điều này không chỉ giúp giảm thiểu rủi ro mà còn tối ưu hóa hiệu suất và tính khả thi trong quá trình phát triển. Việc nghiên cứu và phát triển mô phỏng robot cân bằng hai chân trên MuJoCo không chỉ mang lại lợi ích về mặt chi phí, mà còn góp phần cải thiện khả năng điều khiển robot trong các điều kiện thực tế khắc nghiệt, từ đó đáp ứng được nhu cầu ngày càng tăng của thị trường robot toàn cầu.

Mặc dù vậy, sự phát triển nhanh chóng của công nghệ robot cũng đặt ra nhiều thách thức về pháp lý, an toàn và trách nhiệm. Đặc biệt, việc sử dụng robot trong môi trường thực tế đòi hỏi phải tuân thủ các quy định nghiêm ngặt để đảm bảo an toàn và hiệu quả hoạt động. Việc sử dụng môi trường mô phỏng như MuJoCo để kiểm thử và phát triển robot cân bằng hai chân trước khi triển khai trong thực tế giúp giảm thiểu rủi ro và đảm bảo tuân thủ các tiêu chuẩn pháp lý cần thiết.

Từ những lý do trên, tác giả lựa chọn đề tài "Ứng dụng thuật toán Proximal Policy Optimization trong mô phỏng robot cân bằng hai chân trên MuJoCo" làm hướng nghiên cứu chính cho đồ án tốt nghiệp của mình. Với mục tiêu nghiên cứu cách mà robot có thể cân bằng 2 chân và di chuyển trên địa hình đơn giản và làm tiền đề để tác giả có thể phát triển kiến thức chuyên mô trong lĩnh vực robotics học. Đề tài không chỉ mang lại giá trị thực tiễn mà còn đóng góp vào sự phát triển của các thuật toán học tăng cường trong robot học. Đây là cơ hội để tác giả đóng góp vào một lĩnh vực tiềm năng, đồng thời nâng cao hiểu biết và khả năng ứng dụng công nghệ hiện đại trong thực tế.

2. Mục tiêu nghiên cứu

Đồ án hướng đến mục tiêu nghiên cứu và phát triển một mô hình điều khiển robot cân bằng hai chân trong môi trường mô phỏng MuJoCo. Mục tiêu quan trọng nhất là sử dụng thuật toán Proximal Policy Optimization (PPO) để giúp robot có thể cân bằng và di chuyển của robot trên các địa hình đơn giản,

từ đó giúp tác giả hiểu rõ hơn về cách mà robot học việc di chuyển giống như con người. Việc kiểm chứng mô hình này thông qua mô phỏng sẽ giúp tác giả có được một nền tảng vững chắc cho việc triển khai robot trong các ứng dụng thực tiễn trong cứu hộ, y tế và công nghiệp sau này của tác giả.

3. Nhiệm vụ nghiên cứu

Để đạt được mục tiêu đã đề ra, nghiên cứu cần thực hiện các nhiệm vụ sau:

- Xây dựng mô hình động học của robot cân bằng hai chân trong môi trường mô phỏng MuJoCo.
- Triển khai thuật toán học tăng cường Proximal Policy Optimization (PPO) để điều khiển robot giữ thăng bằng và di chuyển

4. Đối tượng nghiên cứu

Đối tượng nghiên cứu chính của đồ án là hệ thống robot cân bằng hai chân, tập trung vào việc phát triển thuật toán điều khiển dựa trên Proximal Policy Optimization (PPO) trong môi trường mô phỏng MuJoCo. Nghiên cứu cũng bao gồm việc phân tích các yếu tố ảnh hưởng đến khả năng điều khiển và ổn định của robot trong quá trình di chuyển trên các địa hình phức tạp.

5. Phạm vi nghiên cứu

Phạm vi nghiên cứu tập trung vào môi trường mô phỏng MuJoCo để phát triển và kiểm chứng các thuật toán điều khiển robot cân bằng hai chân. Mặc dù các kết quả mô phỏng có thể được áp dụng cho các ứng dụng thực tế, nghiên cứu chủ yếu giới hạn ở việc mô phỏng trên máy tính, không mở rộng sang việc triển khai trên các mô hình robot thực tế. Phạm vi về địa hình sẽ bao gồm bề mặt bằng phẳng để kiểm tra khả năng giữ thăng bằng của robot.

6. Phương pháp nghiên cứu

Nghiên cứu sử dụng phương pháp mô phỏng số với môi trường MuJoCo làm công cụ chính để phát triển và kiểm thử các thuật toán điều khiển. Thuật toán Proximal Policy Optimization (PPO) sẽ được triển khai trong môi trường

mô phỏng này để điều khiển robot. Quá trình nghiên cứu sẽ kết hợp giữa lý thuyết về học tăng cường và thực nghiệm thông qua việc thử nghiệm các thuật toán khác nhau và phân tích kết quả.

7. Kết cấu đồ án

Đồ án sẽ được tổ chức thành 5 chương, với các nội dung chính như sau:

CHƯƠNG 1: GIỚI THIỆU VẤN ĐỀ

Chương này trình bày tổng quan về robot cân bằng hai chân, vai trò của việc mô phỏng, thuật toán học tăng cường PPO trong điều khiển robot, và mối liên hệ giữa chúng, đồng thời định hướng nghiên cứu cùng ứng dụng thực tiễn.

CHƯƠNG 2: CƠ SỞ LÝ THUYẾT

Giới thiệu về các khái niệm cơ bản và các yếu tố quan trọng trong học tăng cường như mô hình Markov, chính sách, hàm giá trị, v.v. Sau đó đi sâu vào thuật toán PPOClip và các mạng LSTM trong mô hình.

CHƯƠNG 3: MÔ HÌNH ĐỀ XUẤT

Trình bày chi tiết phần thiết kế mô hình cho việc huấn luyện robot qua mô phỏng MuJoCo. Phân tích luồng dữ liệu được triển khai trong mô hình, vị trí các công thức toán học được áp dụng. Giải thích các hàm python quan trọng trong dự án.

CHƯƠNG 4: KẾT QUẢ THỰC NGHIỆM

Đưa ra kết quả từng quá trình xây dựng mô hình từ ban đầu cho đến khi hoàn thành. Hiển thị thông tin quan trọng trong quá trình huấn luyện robot bằng đồ thị. Cuối cùng là phân tích kết quả và nhận xét.

CHƯƠNG 5: KẾT LUẬN

Đưa ra kết kết luận và một số những đề xuất hướng phát triển.

CHƯƠNG 1: GIỚI THIỆU VẤN ĐỀ

1.1. Tổng quan về robot cân bằng hai chân

Robot cân bằng hai chân là một nhánh quan trọng trong lĩnh vực robotics, lấy cảm hứng từ khả năng di chuyển linh hoạt của con người. Đây là loại robot được thiết kế với hai chân hoạt động độc lập, có thể giữ thăng bằng và di chuyển trên nhiều loại địa hình khác nhau. Mục tiêu của robot này là tái hiện cách con người bước đi, từ đó đáp ứng được các yêu cầu trong các môi trường mà robot bánh xe hoặc robot bốn chân gặp nhiều hạn chế.

Sự phát triển của robot cân bằng hai chân bắt đầu từ những năm 1970, với các mô hình đơn giản tập trung vào khả năng giữ thăng bằng. Đến nay, các công ty lớn như Honda (với robot ASIMO) và Boston Dynamics (với Atlas) đã đạt được nhiều thành tựu vượt trội, tạo ra các robot không chỉ có khả năng cân bằng mà còn thực hiện được các thao tác phức tạp như leo cầu thang, nhảy, hoặc mang vác đồ vật.

Ưu điểm lớn nhất của robot cân bằng hai chân là tính linh hoạt. Chúng có thể hoạt động hiệu quả trong các môi trường chật hẹp, không bằng phẳng, hoặc thậm chí địa hình phức tạp. Điều này mở ra tiềm năng ứng dụng trong nhiều lĩnh vực như công nghiệp, y tế, cứu hộ, và đời sống thường nhật.

Tuy nhiên, phát triển robot cân bằng hai chân là một thách thức lớn. Hệ thống động lực học của chúng rất phức tạp, yêu cầu các thuật toán điều khiển tiên tiến để duy trì sự ổn định và chuyển động mượt mà. Những khó khăn này làm tăng chi phí và thời gian nghiên cứu, nhưng nhờ các tiến bộ trong công nghệ mô phỏng như MuJoCo, các nhà nghiên cứu có thể giảm thiểu rủi ro và tối ưu hóa hiệu quả trong phát triển.

1.2. Công nghệ mô phỏng trong phát triển robot

Trong lĩnh vực phát triển robot, công nghệ mô phỏng đóng vai trò rất quan trọng, giúp các nhà nghiên cứu và phát triển kiểm tra, tối ưu hóa hệ thống điều khiển mà không cần phụ thuộc vào phần cứng thực tế. Sử dụng môi trường mô phỏng không chỉ tiết kiệm chi phí mà còn giảm thiểu rủi ro hỏng hóc thiết bị trong giai đoạn thử nghiệm. Hiện nay, một số công nghệ mô phỏng phổ biến bao gồm MuJoCo, Gazebo, Isaac Sim, Webots, và PyBullet, mỗi công cụ lại có những ưu và nhược điểm riêng, phù hợp với từng loại dự án cụ thể.

MuJoCo (Multi-Joint dynamics with Contact) nổi bật với khả năng xử lý chính xác các hệ thống robot phức tạp, đặc biệt là các hệ thống nhiều khớp nối và tương tác vật lý phức tạp. Điểm mạnh của MuJoCo nằm ở hiệu suất tính toán cao, hỗ trợ mô phỏng trong thời gian thực, điều này rất quan trọng khi áp dụng các thuật toán học tăng cường như Proximal Policy Optimization (PPO). Khả năng xử lý chính xác động lực học, bao gồm ma sát, va chạm và tiếp xúc, khiến MuJoCo trở thành lựa chọn hàng đầu cho các nghiên cứu liên quan đến robot hai chân.

Các công cụ khác như Isaac Sim của NVIDIA và Gazebo cũng có những thế mạnh riêng. Isaac Sim nổi bật trong việc tích hợp AI và học máy nhờ khả năng dựng hình đồ họa cao cấp và mô phỏng vật lý chi tiết. Trong khi đó, Gazebo lại phổ biến nhờ hỗ trợ tích hợp với hệ thống ROS, rất phù hợp cho các ứng dụng thực tế trên robot vật lý. Tuy nhiên, các công cụ này thường đòi hỏi cấu hình phần cứng mạnh mẽ hoặc gặp hạn chế về tốc độ xử lý trong các mô phỏng phức tạp.

Từ những phân tích trên, MuJoCo được lựa chọn cho đồ án này vì tính chính xác cao, hiệu suất vượt trội và khả năng phù hợp với nghiên cứu về robot hai chân. Đây là nền tảng lý tưởng để kiểm tra và phát triển các thuật toán điều khiển tiên tiến, đảm bảo kết quả mô phỏng có thể áp dụng thực tế.

1.3. Thuật toán học tăng cường trong điều khiển robot

Thuật toán học tăng cường (Reinforcement Learning - RL) đã và đang trở thành một phương pháp quan trọng trong điều khiển robot, đặc biệt là các hệ thống phức tạp như robot hai chân. RL tập trung vào việc dạy cho robot cách thực hiện các hành động tối ưu thông qua quá trình thử nghiệm và học hỏi từ môi trường. Điểm khác biệt của RL so với các phương pháp điều khiển truyền thống là nó không yêu cầu một mô hình động lực học chính xác, mà thay vào đó dựa vào việc tương tác trực tiếp với môi trường để cải thiện hiệu suất.

Trong RL, môi trường được mô hình hóa dưới dạng hệ thống bao gồm trạng thái, hành động và phần thưởng. Robot nhận trạng thái hiện tại từ môi trường, thực hiện một hành động, và nhận được phần thưởng dựa trên chất lượng của hành động đó. Mục tiêu của thuật toán RL là tối đa hóa tổng phần thưởng nhận được theo thời gian, dẫn đến hành vi tối ưu.

Một số thuật toán RL phổ biến trong điều khiển robot bao gồm Q-Learning, Deep Q-Network (DQN), và các thuật toán dựa trên policy gradient như Proximal Policy Optimization (PPO) hoặc Trust Region Policy Optimization (TRPO). Trong đó, PPO là một trong những thuật toán được ưa chuộng nhất do sự cân bằng giữa hiệu suất và tính ổn định trong huấn luyện. PPO sử dụng cơ chế giới hạn cập nhật (clipping) để tránh việc thay đổi chính sách quá lớn, giúp cải thiện hiệu quả học tập và giảm nguy cơ mất ổn định.

Ứng dụng RL trong điều khiển robot hai chân mang lại nhiều lợi thế. Thay vì phải lập trình các quy tắc điều khiển cụ thể, RL cho phép robot tự động khám phá cách di chuyển và giữ thăng bằng một cách tối ưu.

Trong đồ án này, thuật toán PPO được lựa chọn để huấn luyện robot hai chân trong môi trường mô phỏng MuJoCo. Với khả năng giải quyết tốt các bài toán học tăng cường phức tạp, PPO không chỉ giúp robot học cách giữ thăng bằng mà còn khám phá các chiến lược di chuyển hiệu quả, tạo nền tảng cho việc triển khai trong các ứng dụng thực tế sau này.

1.4. Mối liên hệ giữa thuật toán PPO và mô hình động học robot

Thuật toán Proximal Policy Optimization (PPO) và mô hình động học của robot có mối quan hệ chặt chẽ, đặc biệt khi được ứng dụng trong việc điều khiển robot cân bằng hai chân. Sự kết hợp giữa thuật toán học tăng cường và động học của robot không chỉ giúp tối ưu hóa khả năng kiểm soát chuyển động mà còn đảm bảo tính ổn định và hiệu quả trong các môi trường mô phỏng cũng như thực tế.

1.4.1. Mô hình động học robot

Động học thuận (Forward Kinematics): Từ góc khớp nối và chiều dài các đoạn chân, tính toán vị trí và tư thế của robot trong không gian. Điều này giúp xác định robot đang ở đâu trong môi trường.

Động học nghịch (Inverse Kinematics): Từ vị trí mong muốn của các phần cơ thể (như bàn chân hoặc hông), xác định góc khớp nối cần thiết để đạt được vị trí đó.

Ngoài ra, động lực học của robot (robot dynamics) còn bao gồm các yếu tố như ma sát, va chạm và lực phản hồi, là những yếu tố quan trọng ảnh hưởng đến sự ổn định của robot hai chân.

1.4.2. Vai trò của PPO trong điều khiển robot

PPO là thuật toán học tăng cường dựa trên phương pháp policy gradient, cho phép robot học cách đưa ra các hành động tối ưu để đạt được mục tiêu như cân bằng hoặc di chuyển. PPO cải tiến từ các thuật toán trước đó như Trust Region Policy Optimization (TRPO), với ưu điểm là đơn giản hơn trong việc triển khai nhưng vẫn duy trì hiệu quả cao và tính ổn định.

Trong bối cảnh điều khiển robot hai chân, PPO sử dụng mô hình động học của robot như một phần của trạng thái đầu vào để huấn luyện chính sách. Cụ thể:

- **Trạng thái đầu vào:** Bao gồm các thông tin liên quan đến động học của robot, như góc khớp nối, vị trí bàn chân, vận tốc góc, và các lực tác động. Những thông tin này giúp PPO hiểu được trạng thái hiện tại của

robot và đưa ra hành động phù hợp.

- **Hành động đầu ra:** Là các tín hiệu điều khiển, chẳng hạn như mô-men xoắn hoặc vị trí mục tiêu cho các khớp nối. Những hành động này sẽ được áp dụng vào mô hình động học của robot để tạo ra chuyển động.
- **Phần thưởng (Reward):** Được thiết kế dựa trên hiệu quả của hành động, ví dụ như giữ robot ở trạng thái thăng bằng, duy trì tư thế ổn định, hoặc di chuyển theo quỹ đạo mong muốn.

1.4.3. Mối liên hệ giữa PPO và mô hình động học

PPO là thuật toán học tăng cường dựa trên phương pháp policy gradient, cho phép robot học cách đưa ra các hành động tối ưu để đạt được mục tiêu như cân bằng hoặc di chuyển. PPO cải tiến từ các thuật toán trước đó như Trust Region Policy Optimization (TRPO), với ưu điểm là đơn giản hơn trong việc triển khai nhưng vẫn duy trì hiệu quả cao và tính ổn định.

Định hướng học tập: Mô hình động học cung cấp bối cảnh cho thuật toán PPO, giúp PPO học cách xử lý các yếu tố vật lý phức tạp như va chạm, ma sát và lực tiếp xúc.

Tối ưu hóa điều khiển: PPO sử dụng thông tin từ mô hình động học để tối ưu hóa chiến lược điều khiển. Ví dụ, khi robot sắp mất thăng bằng, PPO có thể học cách điều chỉnh lực ở các khớp nối để khôi phục trạng thái ổn định.

Tăng cường tính chính xác: Thông qua mô hình động học, PPO đảm bảo rằng các hành động điều khiển được đưa ra phù hợp với các giới hạn vật lý và cơ học của robot. Điều này giúp tránh các hành động không thực tế, chẳng hạn như vượt quá giới hạn mô-men xoắn của khớp.

1.4.4. Kết hợp trong môi trường mô phỏng

Môi trường mô phỏng MuJoCo đóng vai trò quan trọng trong việc tích hợp giữa PPO và mô hình động học. MuJoCo mô phỏng chính xác các yếu tố động lực học, cung cấp dữ liệu đầu vào chính xác cho PPO. Đồng thời, các hành động do PPO tạo ra được áp dụng ngay lập tức vào mô hình động học, cho phép robot học thông qua phản hồi từ môi trường.

1.4.5. Kết luận

Thuật toán PPO và mô hình động học robot hỗ trợ lẫn nhau để đạt được hiệu quả tối đa trong điều khiển robot hai chân. Mô hình động học cung cấp ngữ cảnh vật lý và phản hồi cần thiết để PPO huấn luyện các chính sách tối ưu, trong khi PPO giúp khai thác tối đa tiềm năng của mô hình động học bằng cách học cách điều khiển hiệu quả và linh hoạt. Việc kết hợp chặt chẽ giữa hai yếu tố này không chỉ giúp robot cân bằng tốt hơn mà còn mở ra nhiều ứng dụng thực tiễn, từ di chuyển trên địa hình phức tạp đến thực hiện các nhiệm vụ yêu cầu độ chính xác cao.

1.5. Định hướng nghiên cứu và ứng dụng thực tiễn

1.5.1. Định hướng nghiên cứu

Robot cân bằng hai chân, với khả năng tái tạo động học giống con người, đang trở thành một trong những lĩnh vực thu hút sự quan tâm của các nhà nghiên cứu. Định hướng nghiên cứu tập trung vào các khía cạnh sau:

Phát triển thuật toán điều khiển tiên tiến: Một trong những mục tiêu quan trọng là cải thiện khả năng điều khiển của robot thông qua việc phát triển các thuật toán học tăng cường như Proximal Policy Optimization (PPO). Các nghiên cứu hướng tới việc kết hợp PPO với các kỹ thuật khác như Học Đa Nhiệm (Multi-Task Learning) hoặc Chuyển Giao Học Tập (Transfer Learning) để giúp robot không chỉ giữ thăng bằng mà còn thực hiện các nhiệm vụ phức tạp, như leo cầu thang, tránh chướng ngại vật hoặc di chuyển trên địa hình không bằng phẳng.

Mô phỏng chi tiết hơn: Mặc dù các công cụ như MuJoCo cung cấp khả năng mô phỏng chính xác, nhưng vẫn cần nghiên cứu để tăng cường độ chân thực của các mô phỏng. Điều này bao gồm mô hình hóa tốt hơn các yếu tố môi trường, chẳng hạn như ma sát, tác động thời tiết và các tương tác vật lý phức tạp khác.

Tích hợp cảm biến và phản hồi: Định hướng tiếp theo là cải thiện khả năng nhận biết và phản ứng của robot bằng cách tích hợp nhiều cảm biến như

IMU. Các nghiên cứu cũng tập trung vào việc khai thác tối đa dữ liệu cảm biến thông qua các thuật toán học sâu để nâng cao khả năng ra quyết định.

Chuyển đổi từ mô phỏng sang thực tế: Một thách thức lớn là chuyển đổi các mô hình và thuật toán được phát triển trong môi trường mô phỏng sang các hệ thống robot thực tế. Nghiên cứu hướng đến việc giảm thiểu sự khác biệt giữa mô phỏng và thực tế, được gọi là Sim-to-Real Transfer, để đảm bảo rằng các giải pháp đã thử nghiệm trên mô phỏng hoạt động tốt trong các tình huống thực tế.

1.5.2. Ứng dụng thực tiễn

Robot cân bằng hai chân không chỉ mang lại giá trị trong lĩnh vực nghiên cứu mà còn mở ra nhiều ứng dụng tiềm năng trong thực tế:

Ứng dụng trong cứu hộ và an ninh: Robot hai chân có thể hoạt động trong các khu vực nguy hiểm hoặc khó tiếp cận, như vùng xảy ra thiên tai, hỏa hoạn hoặc khu vực có phóng xạ. Khả năng giữ thăng bằng và di chuyển linh hoạt cho phép chúng thực hiện các nhiệm vụ như vận chuyển vật tư, tìm kiếm và cứu hộ người bị nạn.

Hỗ trợ y tế và chăm sóc sức khỏe: Robot hai chân có thể hỗ trợ người khuyết tật trong các hoạt động thường nhật, chẳng hạn như đi lại, lấy đồ vật hoặc thực hiện các công việc nhà. Trong các bệnh viện, chúng có thể được sử dụng để vận chuyển thuốc, mẫu xét nghiệm hoặc các thiết bị y tế.

Ứng dụng công nghiệp: Trong các nhà máy hoặc công trường xây dựng, robot hai chân có thể thực hiện các nhiệm vụ như kiểm tra thiết bị, vận chuyển vật liệu qua các địa hình khó khăn hoặc hỗ trợ con người trong các công việc yêu cầu sức mạnh và độ chính xác cao.

Giải trí và giáo dục: Các robot như Atlas của Boston Dynamics đã trở thành minh chứng về sự kết hợp giữa công nghệ và giải trí. Robot hai chân có thể biểu diễn, tham gia vào các chương trình giáo dục, hoặc trở thành công cụ giảng dạy về công nghệ và lập trình.

Nghiên cứu không gian: Trong các nhiệm vụ khám phá không gian, robot

hai chân có thể được triển khai để di chuyển trên bề mặt các hành tinh với địa hình gồ ghề. Khả năng tự cân bằng và thích nghi với điều kiện môi trường khắc nghiệt giúp chúng trở thành lựa chọn lý tưởng trong các sứ mệnh tương lai.

1.5.3. Tại sao tập trung vào robot cân bằng hai chân?

Robot hai chân có ưu thế lớn trong các môi trường mà robot truyền thống, như robot bánh xe hoặc bốn chân, gặp khó khăn. Khả năng di chuyển giống con người giúp chúng vượt qua các rào cản tự nhiên như bậc thang, địa hình dốc hoặc không gian hẹp. Thêm vào đó, chúng có thể mang theo các thiết bị, công cụ hoặc vật phẩm giống như con người, mở rộng khả năng hoạt động trong các nhiệm vụ yêu cầu sự linh hoạt.

1.5.4. Kết luận

Việc nghiên cứu và ứng dụng robot cân bằng hai chân không chỉ thúc đẩy sự tiến bộ trong công nghệ robot mà còn giải quyết nhiều vấn đề thực tiễn trong xã hội. Sự kết hợp giữa thuật toán tiên tiến như PPO và môi trường mô phỏng chính xác như MuJoCo đã mở ra một hướng đi hiệu quả để phát triển các robot thông minh và linh hoạt hơn. Những nghiên cứu này không chỉ góp phần nâng cao hiểu biết về robot học mà còn đặt nền tảng cho các ứng dụng trong tương lai, từ đó đáp ứng nhu cầu ngày càng cao của con người và xã hội.

CHƯƠNG 2: CƠ SỞ LÝ THUYẾT

Trong quá trình tìm hiểu và nghiên cứu về lĩnh vực Học Tăng Cường (Reinforcement Learning - RL), tác giả nhận thấy rằng việc trình bày lý thuyết một cách hệ thống và mạch lạc không chỉ giúp người đọc dễ dàng nắm bắt tiến trình phát triển của các phương pháp, mà còn hiểu rõ bản chất sâu sắc của RL. Với mong muốn mang lại một cái nhìn rõ ràng, toàn diện và logic, tác giả đã xây dựng Chương 2: Cơ sở lý thuyết như một bức tranh tái hiện hành trình phát triển của RL, từ những nền tảng cơ bản đến các phương pháp hiện đại, tiêu biểu là Tối Ưu Hóa Chính Sách Gần Nhất (Proximal Policy Optimization - PPO).

Chương mở đầu với phần 2.1 Giới thiệu RL, cung cấp cái nhìn tổng quan về Học Tăng Cường, tập trung làm rõ mối quan hệ giữa trạng thái, hành động và phần thưởng. Tiếp đó, tác giả trình bày 2.2 Quy trình Quyết định Markov (Markov Decision Process - MDP), là nền tảng để mô hình hóa các bài toán quyết định tuần tự và đặt nền móng lý thuyết cho các thuật toán RL.

Phần 2.3 Phương trình Bellman (1950–1954) và lập trình động được giới thiệu như những phương pháp cốt lõi để giải các bài toán tối ưu hóa trong môi trường có cấu trúc MDP.

Cuối cùng, chương tập trung vào 2.4 Phương pháp Tối Ưu Hóa Chính Sách Gần Nhất (PPO) (2017), một trong những phương pháp RL hiện đại nổi bật. PPO được đánh giá cao nhờ cân bằng giữa hiệu suất và tính ổn định trong quá trình học, trở thành công cụ hiệu quả trong các bài toán thực tiễn.

Thực ra hết chương 2.3 đến 2.4 còn khá nhiều nội dung hay và thú vị về RL, nhưng do lý thuyết đó khá dài và khó nên tác giả không nghiên cứu kĩ mà chỉ tìm hiểu thêm để biết lý do mà các thuật toán trong RL phát triển. Tiếp nối sau phương trình Bellman, Vấn đề Tên Cướp Nhiều Vũ Trang - Multi-Armed Bandit (MAB) được như một trong những khái niệm sớm nhất, làm rõ nguyên lý khám phá và khai thác (exploration vs exploitation) – yếu tố trọng yếu trong

các bài toán RL. Sau đó, các phương pháp lập kế hoạch (Planning methods) ra đời, trong đó một số kỹ thuật đáng chú ý như Lập trình động - Dynamic Programming (DP) tiếng việt hay gọi là quy hoạch động. Và Policy Iteration và Value Iteration là 2 giải thuật giải dựa trên lý thuyết lập trình động để giải phương trình Bellman. Nối tiếp đó là phương pháp Monte Carlo (MC) xuất hiện như một giải pháp tiên phong, cho phép học từ trải nghiệm thực tế mà không cần mô hình hóa môi trường, khắc phục hạn chế của các phương pháp lập kế hoạch (Planning methods) trước đó. Tuy nhiên,lại một lần nữa hạn chế lớn của Monte Carlo là sự chậm trễ trong cập nhật, khi cần hoàn thành toàn bộ tập dữ liệu mới có thể tính toán giá trị. Chính vấn đề này đã mở đường cho sự ra đời của các phương pháp hiệu quả hơn. Đó là phương pháp TD Learning, giúp cập nhật giá trị ngay khi có dữ liệu mới, mang lại hiệu quả cao hơn và phù hợp với các môi trường thực tế. Từ đây, hai hướng tiếp cận chính trong RL được hình thành rõ nét: Value-Based Methods và Policy Gradient Methods. Và, Policy Gradient Methods đã đánh dấu một bước đột phá với khả năng tối ưu trực tiếp chính sách mà không cần lưu trữ giá trị trạng thái, trở thành nền tảng cho thuật toán Tối Ưu Hóa Chính Sách Gần Nhất (PPO).

Việc trình bày các nội dung lý thuyết một cách có hệ thống và logic không chỉ giúp người đọc hình dung rõ ràng mối quan hệ giữa các phương pháp mà còn tạo cơ sở vững chắc để tiếp cận các chương sau, nơi lý thuyết được áp dụng để giải quyết các bài toán thực tiễn. Qua đó, Chương 2 không chỉ là nền tảng lý thuyết mà còn góp phần định hướng tư duy cho toàn bộ luận văn.

2.1. Giới thiệu về học tăng cường (Reinforcement Learning)

2.1.1. Nguồn gốc và sự phát triển

Reinforcement Learning (RL) bắt nguồn từ các khái niệm cơ bản trong tâm lý học và lý thuyết điều khiển. Ý tưởng về học qua thưởng phạt, cơ sở của RL, được hình thành từ nghiên cứu của Edward Thorndike vào cuối thế kỷ 19 với "Luật hiệu ứng" (Law of Effect). Thorndike phát hiện rằng các hành vi dẫn

đến kết quả tích cực có xu hướng được lặp lại, trong khi các hành vi không hiệu quả thì bị loại bỏ.

Trong suốt thế kỷ 20, các nguyên lý này đã được tiếp tục nghiên cứu và mở rộng trong lĩnh vực tâm lý học hành vi. B.F. Skinner, một nhà tâm lý học nổi tiếng, đã phát triển lý thuyết điều kiện hóa (operant conditioning), trong đó ông chứng minh rằng hành vi có thể được hình thành và điều chỉnh thông qua việc sử dụng thưởng phạt. Những nguyên lý này đã đóng vai trò nền tảng trong sự phát triển ban đầu của RL.

Vào giữa thế kỷ 20, RL bắt đầu gắn kết chặt chẽ với lý thuyết điều khiển và toán học. Richard Bellman đã giới thiệu phương trình Bellman, một công cụ toán học quan trọng cho các vấn đề tối ưu hóa động lực. Phương trình Bellman đã đặt nền móng cho nhiều phương pháp và thuật toán trong RL, cho phép giải quyết các bài toán ra quyết định tuần tự. Cùng với đó, John von Neumann và Oskar Morgenstern phát triển lý thuyết trò chơi, cung cấp thêm cơ sở lý thuyết cho việc nghiên cứu hành vi của agent trong các tình huống cạnh tranh.

Các thập kỷ sau đó chứng kiến sự ra đời của các thuật toán học tăng cường đầu tiên, như Q-learning và SARSA, đặt nền tảng cho RL hiện đại. Những thuật toán này cho phép agent học hỏi từ tương tác với môi trường thông qua các chuỗi hành động và phản hồi từ môi trường.

Sự kết hợp giữa RL và học sâu (Deep Learning) vào đầu thập kỷ 2010 đã tạo ra bước đột phá quan trọng, đưa RL vào kỷ nguyên mới. DeepMind, một công ty con của Google, đã phát triển Deep Q-Network (DQN), một thuật toán sử dụng mạng nơ-ron sâu để học chính sách hành động trong các không gian trạng thái phức tạp. Thành công của DQN trong việc đánh bại con người trong các trò chơi video phức tạp như Atari đã chứng minh tiềm năng to lớn của sự kết hợp này.

Ngày nay, RL tiếp tục phát triển mạnh mẽ, được áp dụng trong nhiều lĩnh vực khác nhau từ robot học, điều khiển tự động, tài chính, đến y tế, mở ra nhiều cơ hội mới cho trí tuệ nhân tạo và học máy.

2.1.2. Các bước ngoặt quan trọng

Reinforcement Learning đã trải qua nhiều bước ngoặt quan trọng, giúp nó phát triển từ một ý tưởng lý thuyết đến một công cụ mạnh mẽ trong trí tuệ nhân tạo. Một trong những cột mốc quan trọng đầu tiên là sự ra đời của thuật toán Q-learning vào năm 1989 bởi Christopher Watkins. Q-learning đánh dấu một bước tiến lớn khi cho phép agent học cách tối ưu hóa hành vi mà không cần một mô hình hoàn chỉnh của môi trường, mở rộng khả năng ứng dụng của RL.

Bước ngoặt tiếp theo là sự kết hợp giữa RL và học sâu, đặc biệt là qua sự phát triển của Deep Q-Network (DQN) vào năm 2013. DQN, được phát triển bởi nhóm nghiên cứu tại DeepMind, là một thành tựu đột phá khi lần đầu tiên sử dụng mạng nơ-ron sâu để học chính sách hành động trong các không gian trạng thái lớn và phức tạp. Thành công của DQN trong việc đánh bại các trò chơi video cổ điển đã làm bùng nổ sự quan tâm đến RL và học sâu, đặt nền móng cho hàng loạt các nghiên cứu và ứng dụng tiếp theo.

Một bước ngoặt khác là sự ra đời của các thuật toán như Trust Region Policy Optimization (TRPO) và Proximal Policy Optimization (PPO). Những thuật toán này giải quyết vấn đề về ổn định và hiệu quả trong quá trình học của agent, làm cho RL trở nên mạnh mẽ hơn trong các ứng dụng thực tế. TRPO và PPO đã giúp RL trở nên linh hoạt và ổn định hơn, cho phép nó được áp dụng trong nhiều lĩnh vực khác nhau, từ điều khiển robot đến tài chính.

2.1.3. Ứng dụng của RL

Reinforcement Learning (RL) đã được áp dụng trong nhiều lĩnh vực khác nhau nhờ khả năng xử lý các tình huống phức tạp và không chắc chắn. Trong chơi game, RL đã nổi tiếng với thành công của AlphaGo, một agent đã đánh bại nhà vô địch cờ vây thế giới. Trong robot học, RL giúp các robot học cách điều khiển và thực hiện các nhiệm vụ phức tạp như tự động lái xe hay thao tác vật lý trong môi trường không xác định. Ngoài ra, RL còn được áp dụng trong tài chính để tối ưu hóa chiến lược đầu tư, và trong y tế để phát triển các phác đồ điều trị cá nhân hóa.

2.1.4. Khái niệm RL

Reinforcement Learning (RL) là một phương pháp học máy trong đó một agent học cách hành động trong một môi trường để tối đa hóa phần thưởng tích lũy theo thời gian. Trong RL, agent không có trước thông tin về môi trường mà thay vào đó, nó học hỏi bằng cách thực hiện các hành động và quan sát kết quả, điều này giúp điều chỉnh chính sách hành động để đạt được mục tiêu tối ưu. RL khác với các phương pháp học có giám sát vì nó không yêu cầu dữ liệu gán nhãn, mà học từ tương tác trực tiếp với môi trường.

2.1.5. Các thành phần quan trọng trong RL

Để định nghĩa các thành phần quan trọng trong một bài toán Học Tăng cường (Reinforcement Learning - RL), tác giả sẽ bắt đầu từ một ví dụ thực tế đơn giản như sau (có thể xem trước *video trên youtube* để hình dung rõ ràng hơn ví dụ): Trên mặt bàn có nhiều tấm bìa hình tròn với các màu sắc khác nhau, và một chú gà cần được huấn luyện để mổ trúng tấm bìa màu hồng. Quá trình huấn luyện diễn ra như sau:

Ban đầu, chỉ một tấm bìa màu hồng được đặt trên mặt bàn, và trên đó có một chấm trắng tượng trưng cho một hạt gạo. Mỗi khi chú gà mổ trúng chấm trắng này, người ta sẽ thưởng cho chú một hạt gạo từ khay. Việc này được lặp đi lặp lại cho đến khi chú gà hình thành phản xạ: hễ nhìn thấy tấm bìa màu hồng, chú sẽ mổ trúng và nhận được phần thưởng.

Sau khi hình thành phản xạ này, người ta đặt thêm các tấm bìa màu khác lên bàn. Lúc này, nếu chú gà mổ đúng tấm bìa màu hồng, chú sẽ được thưởng gạo; ngược lại, nếu mổ sai, chú sẽ không được thưởng. Qua một thời gian, chú gà đã có khả năng phân biệt tấm bìa màu hồng giữa nhiều tấm bìa màu khác nhau và mổ chính xác.

Từ ví dụ trên, có thể nhận thấy việc huấn luyện một robot hay một đối tượng nào đó để hoàn thành nhiệm vụ trong bài toán RL tương tự như việc huấn luyện chú gà. Các thành phần quan trọng trong một bài toán RL có thể được định nghĩa như sau:

1. Agent (Tác nhân): Là đối tượng mà ta muốn huấn luyện. Trong ví dụ, agent chính là chú gà.

2. Environment (Môi trường): Là tất cả các yếu tố mà agent có thể tương tác, ngoại trừ chính agent. Trong ví dụ, môi trường bao gồm mặt bàn, các tấm bìa, và phần thưởng (hạt gạo).

3. Action (Hành động): Là tập hợp các thao tác mà agent có thể thực hiện trong môi trường. Đối với chú gà, các hành động có thể bao gồm: bước đi theo hướng trên bàn, cúi xuống mổ thóc, quay đầu, v.v.

4. State (Trạng thái): Là thông tin mô tả vị trí và tình trạng của agent tại một thời điểm cụ thể. Ví dụ, tại thời điểm 0:11 chú gà đứng tại chỗ và đang cúi xuống, trạng thái của chú gà có thể bao gồm vị trí chân, đầu, ngón chân và hướng nhìn của nó.

5. Reward (Phần thưởng): Là giá trị mà agent nhận được sau khi thực hiện một hành động trong một trạng thái cụ thể. Ví dụ, tại thời điểm 0:12, nếu chú gà đang đứng trước tấm bìa màu hồng (state) và thực hiện hành động mổ trúng (action), chú sẽ nhận được phần thưởng là một hạt gạo (reward).

6. Policy (Chính sách): Là chiến lược hoặc quy tắc mà agent (tác nhân) sử dụng để quyết định hành động tiếp theo dựa trên trạng thái hiện tại. Trong bài toán RL, policy có thể được biểu diễn dưới dạng một hàm xác suất hoặc một mạng nơ-ron, giúp tác nhân chọn hành động tối ưu để tối đa hóa phần thưởng dài hạn. Trong ví dụ về chú gà, policy của chú gà được hình thành dần qua quá trình huấn luyện như sau:

Ban đầu, policy rất đơn giản: mổ vào tấm bìa duy nhất trên bàn (tấm bìa màu hồng) để nhận được phần thưởng. Sau khi chú gà đã quen với việc liên kết tấm bìa màu hồng với phần thưởng, policy của nó bắt đầu thay đổi. Khi có thêm các tấm bìa màu khác xuất hiện trên bàn, chú gà phải học cách chọn đúng tấm bìa màu hồng để tối đa hóa phần thưởng (thay vì mổ ngẫu nhiên). Quá trình này giúp policy của chú gà được cải thiện theo thời gian, chuyển từ hành động ngẫu nhiên sang hành động có định hướng và tối ưu hơn. Policy của chú gà trong ví

dụ trên có thể hiểu là một hàm xác định xác suất mổ vào từng tấm bìa dựa trên quan sát:

Nếu quan sát là một tấm bìa màu hồng, xác suất mổ vào tấm bìa đó tăng lên cao (gần 100%). Nếu quan sát là tấm bìa màu khác, xác suất mổ vào đó sẽ giảm xuống thấp (gần 0%). Trong các bài toán RL, policy có thể là:

Deterministic Policy (Chính sách xác định): Ví dụ, mỗi khi nhìn thấy tấm bìa màu hồng, chú gà sẽ luôn mổ. Stochastic Policy (Chính sách ngẫu nhiên): Ví dụ, chú gà chọn hành động mổ với một xác suất cao hơn khi nhìn thấy tấm bìa màu hồng so với các tấm bìa khác. Như vậy, policy không chỉ đơn thuần là cách ra quyết định mà còn là trung tâm của quá trình học tập, giúp tác nhân cải thiện hành vi của mình qua thời gian dựa trên trải nghiệm và phản hồi từ môi trường.

Ví dụ này minh họa cách các thành phần trong RL được liên kết với nhau, tạo thành một khung huấn luyện hoàn chỉnh để **tối ưu hóa** hành vi của agent (policy) nhằm đạt được mục tiêu đã đề ra.

2.2. Quy trình Quyết định Markov (1907–1938)

2.2.1. Giới thiệu

Như chúng ta đều biết, toán học không chỉ là công cụ lý thuyết mà còn là chìa khóa giải quyết các bài toán thực tế một cách hệ thống và chính xác. Để làm được điều này, vấn đề thực tế cần được chuyển hóa qua nhiều bước: từ ngôn ngữ tự nhiên sang dạng văn bản chính quy, rồi từ văn bản đó thành các ký hiệu và khái niệm toán học và từ đây chúng ta có thể dùng toán học để giải quyết vấn đề đó. Quá trình biến văn bản thành khái niệm toán học trên được gọi là mô hình hóa toán học (Mathematical Modeling), một bước đi nền tảng và không thể thiếu trong toán học ứng dụng.

Trong lĩnh vực trí tuệ nhân tạo (AI) và học máy (Machine Learning), một trong những phương pháp mô hình hóa mạnh mẽ và hiệu quả nhất là Quy trình Quyết định Markov (Markov Decision Process - MDP). MDP cung cấp một

khung lý thuyết vững chắc để biểu diễn và giải quyết các bài toán ra quyết định trong môi trường ngẫu nhiên và có tính thời gian, mở ra những ứng dụng quan trọng trong robot học, tự động hóa, và nhiều lĩnh vực khác của AI."

MDP là một mô hình toán học được sử dụng để mô tả cách một tác nhân (agent) đưa ra quyết định trong một môi trường động và bất định nhằm tối ưu hóa một mục tiêu cụ thể. MDP được coi là nền tảng của các bài toán ra quyết định tuần tự (sequential decision-making), nơi mà kết quả của một hành động không chỉ phụ thuộc vào hành động đó mà còn vào trạng thái hiện tại và hành động trong tương lai.

Vấn đề cốt lõi mà MDP hướng tới là tối ưu hóa một chuỗi quyết định, sao cho phần thưởng nhận được (reward) trong dài hạn là cao nhất. Điều này đặc biệt quan trọng trong các bài toán mà tác nhân cần cân nhắc giữa "lợi ích ngắn hạn" và "lợi ích dài hạn" để đạt được hiệu suất tốt nhất.

Ví dụ, trong việc điều khiển robot di chuyển từ điểm A đến điểm B, robot cần quyết định:

- Nên chọn con đường ngắn nhưng nhiều chướng ngại vật, hay con đường dài nhưng ít rủi ro hơn?
- Làm thế nào để vừa giảm thiểu năng lượng tiêu hao vừa tối đa hóa tốc độ?

MDP cung cấp một khuôn khổ lý thuyết để mô hình hóa và giải quyết các bài toán ra quyết định như trên. Ta có thể xem MDP như các biến của hàm số f(x), trong đó x đại diện cho các thành phần của bài toán ra quyết định. Các phương pháp được trình bày trong các phần tiếp theo của chương này, như Phương trình Bellman, Lập trình động, v.v., sẽ đóng vai trò là cơ sở lý luận và thuật toán nhằm giải quyết bài toán, tức là tìm giá trị x sao cho hàm f(x) đạt được giá trị tối ưu.

2.2.2. Định nghĩa

Bài toán cốt lõi của MDP là tìm ra một nguyên tắc ra quyết định tối ưu – thường được gọi là **chính sách** (π). Chính sách $\pi(s)$ xác định hành động mà

người ra quyết định sẽ chọn khi ở trạng thái s. Một MDP là một mô hình quyết định trong đó một tác nhân (agent) chọn các hành động trong các trạng thái khác nhau với mục tiêu tối đa hóa phần thưởng tích lũy trong tương lai. Chìa khóa trong MDP là Tính chất Markov: *"Tương lai phụ thuộc vào hiện tại và không phụ thuộc vào quá khứ"*. Cụ thể hơn, tương lai độc lập với quá khứ khi biết hiện tại. và một quá trình quyết định Markov có thể được biểu diễn như một bộ 5 phần tử:

$$(\mathbf{S}, \mathbf{A}, \mathbf{P}(\cdot,\cdot), \mathbf{R}(\cdot,\cdot), \gamma), \tag{2.2.1}$$

Trong đó:

- $\mathbf{S}$: Là tập hợp các trạng thái của hệ thống, có thể là hữu hạn hoặc vô hạn. Trong các thuật toán và ứng dụng thực tế, thường giả định rằng $\mathbf{S}$ là hữu hạn.
- $\mathbf{A}$: Là tập hợp các hành động có thể thực hiện được. Với mỗi trạng thái $\mathbf{s} \in \mathbf{S}$, có một tập con $\mathbf{A_s} \subseteq \mathbf{A}$ các hành động khả thi trong trạng thái đó. Các hành động này có thể tác động đến môi trường và dẫn đến các trạng thái mới.
- $\mathbf{P}(\mathbf{s}'|\mathbf{s}, \mathbf{a})$: Là xác suất chuyển từ trạng thái $\mathbf{s}$ sang trạng thái $\mathbf{s}'$ khi thực hiện hành động $\mathbf{a}$ tại trạng thái $\mathbf{s}$ vào thời điểm $\mathbf{t}$. Đây là xác suất chuyển trạng thái từ $\mathbf{s}$ sang $\mathbf{s}'$ sau khi thực hiện hành động $\mathbf{a}$ tại trạng thái $\mathbf{s}$ vào thời điểm $\mathbf{t}$. Đặc điểm của MDP là xác suất chuyển trạng thái chỉ phụ thuộc vào trạng thái hiện tại và hành động thực hiện, không phụ thuộc vào quá trình lịch sử.
- $\mathbf{R}(\mathbf{s}'|\mathbf{s}, \mathbf{a})$: Là phần thưởng nhận được sau khi chuyển từ trạng thái $\mathbf{s}$ sang trạng thái $\mathbf{s}'$ khi thực hiện hành động $\mathbf{a}$. Đây là phần thưởng trực tiếp mà tác nhân nhận được ngay lập tức sau mỗi lần chuyển trạng thái. Ví dụ, trong một bài toán quản lý năng lượng tòa nhà, $\mathbf{R}(\mathbf{s}'|\mathbf{s}, \mathbf{a})$ có thể là chi phí năng lượng hoặc mức tiết kiệm được sau mỗi hành động.
- γ: Là hệ số chiết khấu, một giá trị trong khoảng $[0, 1]$, thể hiện mức độ

quan trọng của phần thưởng trong tương lai so với phần thưởng hiện tại. Khi γ gần 1, tác nhân coi trọng phần thưởng trong tương lai. Ngược lại, khi γ gần 0, tác nhân chỉ quan tâm đến phần thưởng hiện tại. Hệ số chiết khấu này thường được tính theo công thức:

$$\gamma = \frac{\mathbf{1}}{\mathbf{1}+\mathbf{r}}, \tag{2.2.2}$$

với $\mathbf{r}$ là tỷ lệ chiết khấu (*discount rate*).

Ví dụ cho các ký hiệu toán học trong MDP:

Giả sử chúng ta đang giải quyết bài toán quản lý năng lượng trong một tòa nhà thông minh với ba trạng thái:

$$\mathbf{S} = \{\text{Nhiệt độ cao}, \text{Nhiệt độ tối ưu}, \text{Nhiệt độ thấp}\}.$$

Các hành động có thể thực hiện là:

$$\mathbf{A} = \{\text{Bật điều hòa}, \text{Tắt điều hòa}\}.$$

Xác xuất chuyển trạng thái có thể là:

$$\mathbf{P} = \{\mathbf{P}(\mathbf{s}'|\mathbf{s}, \mathbf{a})\}.$$

trong đó $\mathbf{P}(\mathbf{s}'|\mathbf{s}, \mathbf{a})$ là:

- $\mathbf{P}_{\text{Bật}}(\text{Nhiệt độ cao}, \text{Nhiệt độ tối ưu}) = \mathbf{0.8}$

Xác suất điều hòa bật sẽ làm nhiệt độ giảm xuống mức tối ưu.

- $\mathbf{P}_{\text{Tắt}}(\text{Nhiệt độ cao}, \text{Nhiệt độ cao}) = \mathbf{0.9}$

Xác suất khi tắt điều hòa, nhiệt độ vẫn giữ nguyên ở mức cao.

Các phần thưởng có thể là:

$$\mathbf{R} = \{R_a(s, s')\}.$$

- $\mathbf{R}_{\text{Bật}}(\text{Nhiệt độ cao}, \text{Nhiệt độ tối ưu}) = +\mathbf{10}$

Phần thưởng dương vì nhiệt độ đã trở về mức tối ưu.

- $\mathbf{R}_{\text{Tắt}}(\text{Nhiệt độ cao}, \text{Nhiệt độ cao}) = -\mathbf{5}$

Phần thưởng âm vì nhiệt độ vẫn ở mức cao.

Hệ số chiết khấu (γ)

Giả sử $\gamma = \mathbf{0.9}$, điều này có nghĩa là tác nhân coi trọng phần thưởng trong tương lai gần nhưng vẫn ưu tiên phần thưởng hiện tại.

2.2.3. Bài toán tối ưu hoá MDP

Sau khi đưa ra được các thành phần có trong MDP thành dạng toán học, thì điều quan trong tiếp theo đó là: Làm sao để ta có thể tối ưu một bài toán MDP?

Dựa vào ví dụ về chú gà, hãy hình dung quá trình học tập của chú khi mổ vào tấm bìa màu hồng để nhận thưởng là những hạt gạo. Chú gà đã học cách mổ đúng vào tấm bìa màu hồng thông qua việc tối đa hóa phần thưởng nhận được. Nhưng làm thế nào chú gà đạt được điều đó?

Câu trả lời là: chú gà hướng đến mục tiêu nhận được nhiều phần thưởng nhất có thể, cụ thể là các hạt gạo. Tuy nhiên, cần phân biệt giữa "phần thưởng tối đa" và "tổng phần thưởng tích lũy" (**G** - Return value). Tổng phần thưởng tích lũy là tổng số hạt gạo mà chú gà nhận được trong suốt một quá trình, bắt đầu từ trạng thái đầu tiên $\mathbf{s_t}$ đến trạng thái cuối cùng $\mathbf{s_T}$. Chính việc tối ưu hóa tổng phần thưởng tích lũy này giúp chú gà học được cách mổ chính xác.

Để đạt được mục tiêu này, chú gà phát triển một phản xạ, mà trong MDP (Mô hình Quyết định Markov) được gọi là "chính sách" (policy). Chính sách này là chiến lược giúp chú gà đưa ra các hành động tại mỗi trạng thái sao cho tổng phần thưởng tích lũy kỳ vọng đạt giá trị lớn nhất.

Vì vậy, bài toán tối ưu trong MDP được phát biểu là: **"Tìm một chính sách ($\pi(\mathbf{s}|\mathbf{a})$) tối ưu sao cho giá trị kỳ vọng của tổng phần thưởng tích lũy (G) tại mỗi trạng thái $\mathbf{s_t}$ đạt cực đại."** Chính sách này được gọi là **"chính sách tối ưu"** (Optimal Policy).

2.2.4. Chính sách Tối ưu (Optimal Policy)

Như đã trình bày ở trên, "Tìm một chính sách ($\pi(\mathbf{s}|\mathbf{a})$) tối ưu sao cho giá trị kỳ vọng của tổng phần thưởng tích lũy (**G**) tại mỗi trạng thái $\mathbf{s_t}$ đạt cực đại." là cách tối ưu bài toán MDP. Và bước này sẽ phân ra làm 3 phần, phần đầu: Xác định "**chính sách ($\pi(\mathbf{s}|\mathbf{a})$)**", phần sau: Xác định giá trị kỳ vọng của "**tổng phần thưởng tích lũy (G)** tại mỗi trạng thái", phần cuối: Tìm kiếm các giá trị cần thết để **"tối ưu"** chính sách.

a, Xác định chính sách - Policy

Chính sách cũng giống như kinh nghiệm của con người vậy. Kinh nghiệm sẽ được phân ra thành nhiều "loại" khác nhau mỗi loại lại được sử dụng trên "tình huống" phù hợp bên trong một môi trường nào đó. Và chính sách cũng thế, sẽ có các "loại" chính sách khác nhau, được sử dụng trên "điều kiện" phù hợp từ một "môi trường" nào đó. Nói cách khác, agent sử dụng model như công cụ giúp agent thấu hiểu môi trường, policy như công cụ để agent giao tiếp với môi trường. Agent là chủ thể tương tác với môi trường, môi trường phản hồi lại sự tương tác đó, và agent tạo ra model để hiểu sự phản hồi ấy và đáp lại sự phản hồi từ môi trường bằng policy. Để dễ hiểu hơn, hãy hình dung trong môi trường dưới nước, con người sử dụng tàu ngầm để thám hiểm. Tàu ngầm chính là model, các nút điều khiển là policy và môi trường là nước. Tàu ngầm không thể di chuyển trên cạn bởi vì chỉ có ở dưới nước, tàu ngầm mới giúp con người hiểu về lòng đại dương hay chỉ có ở dưới nước tàu ngầm mới được sinh ra hoặc mới có tác dụng. Vì vậy, để xác định chính sách, trước hết những khái niệm cơ bản về 2 kiểu mô hình môi trường và 2 kiểu chính sách sẽ được nêu ra dưới đây:

A. Model-based RL and Model-free RL

1. Khi một tác nhân hoạt động trong một môi trường (environment).

2. Môi trường phản ứng lại tác nhân hoặc cung cấp cho tác nhân thông tin từ môi trường.
3. Tác nhân tiếp nhận thông tin từ môi trường, thực hiện hành động **a** và chuyển từ trạng thái **s** sang trạng thái mới $\mathbf{s}'$, tác nhân sẽ nhận được phần thưởng **r**.

Khi tác nhân được môi trường phản ứng lại hoặc nhận được các thông tin do môi trường cung cấp, tác nhân sẽ dựa vào đó, đưa ra hành động để chuyển từ trạng thái s sang s'. Cách mà môi trường phản ứng lại agent trên được gọi là Model.

Và có 2 loại model:

- **Model-based RL**: Dựa vào model này, tác nhân sẽ xác định "rõ ràng" xác xuất chuyển từ trạng thái s sang s', nói cách khác, khi chọn hành động a để thực hiển, tác nhân đã **biết chắc chắn không gian trạng thái s'** sau đó. Vì vậy với model-based, model sẽ chứa hàm xác suất chuyển trạng thái (transition probability function):

$$\mathbf{P}(\mathbf{s}'|\mathbf{s},\mathbf{a}) = \mathbb{P}[\mathbf{S_{t+1}} = \mathbf{s}'|\mathbf{S_t} = \mathbf{s}, \mathbf{A_t} = \mathbf{a}] \tag{2.2.3}$$

- **Model-fee RL:** Dựa vào model này, tác nhân chỉ nhận được sự tương tác với môi trường mà **không biết trạng thái s'** là gì, mà phải tự thu thập khi thực hiện xong hành động a tại trạng thái s. Vì vậy với model-free, model sẽ không có xác suất chuyển trạng thái mà chỉ dựa vào trải nghiệm để thu thập dữ liệu của trạng thái s'.

B. Deterministic policy and Stochastic policy

Deterministic policy là chính sách cố định, Stochastic policy là chính sách ngẫu nhiên.

- **Deterministic policy:** Ở mỗi trạng thái, chính sách xác định **duy nhất một hành động**. Ví dụ: Nếu robot ở S1, nó luôn chọn A1. Có nghĩa là

nếu cho thử lại 100 lần ở trạng thái S1 với **cùng một policy** thì tác nhân luôn chọn hành động A1 đó. Từ đó ta có công thức policy của chính sách cố định là:

$$\pi(\mathbf{s}) = \mathbf{a} \tag{2.2.4}$$

- **Stochastic policy**: Ở mỗi trạng thái, chính sách xác định **xác suất** để thực hiện các hành động. Ví dụ: Nếu robot ở S1, nó có 70% khả năng chọn A1 và 30% chọn A2. Có nghĩa là nếu cho thử lại 100 lần ở trạng thái S1 với **cùng một policy** thì tác nhân sẽ có 70 lần chọn A1 và 30 lần chọn A2. Từ đó ta có công thức policy của chính sách ngẫu nhiên là:

$$\pi(\mathbf{a}|\mathbf{s}) = \mathbb{P}[\mathbf{A_t} = \mathbf{a}|\mathbf{S_t} = \mathbf{s}] \tag{2.2.5}$$

Từ 2 khái nhiệm A., B. trên, một bài toán tối ưu chính sách sẽ trải qua các bước cơ bản sau:

1. Thu thập trạng thái s
2. Dựa vào Policy đưa ra hành động. Tác nhân có thể sử dụng 1 trong 2 policy sau Deterministic policy hoặc Stochastic policy.
3. Từ model, tác nhân sẽ biết trạng thái tiếp theo s' khi thực hiện hành động a. Tác nhân có thể sử dụng 1 trong 2 model để xác định trạng thái s, Model-based RL hoặc Model-free RL.
4. Tác nhân nhận được phần thưởng từ trạng thái s':

$$\mathbf{R}(\mathbf{s}'|\mathbf{s},\mathbf{a}) = \mathbb{E}[\mathbf{R_{t+1}}|\mathbf{S_t} = \mathbf{s}, \mathbf{A_t} = \mathbf{a}] \tag{2.2.6}$$

Qua đó, các trường hợp có thể xảy ra trong 1 bài toán MDP

Loại chính sách	Model-based RL	Model-free RL
Determini -stic Policy	**Ví dụ**: Robot hút bụi biết bản đồ môi trường và chỉ chọn hành động duy nhất (ví dụ: đi đến điểm sạch nhất tiếp theo). **Chính sách**: Dựa trên mô hình bản đồ (được xây dựng trước) và công thức heuristic để chọn đường đi ngắn nhất. **Model**: Sử dụng mô hình động lực học của môi trường (các quy tắc di chuyển đã biết).	**Ví dụ**: Robot học cách bật công tắc đèn bằng cách thử nghiệm với DQN. **Chính sách**: Chọn hành động duy nhất có giá trị (Q-value) cao nhất tại mỗi trạng thái. **Model**: Không cần xây dựng mô hình môi trường, chỉ tối ưu hóa hàm giá trị (Q-value).
Stochastic Policy	**Ví dụ**: Robot giao hàng qua đường có thể có các yếu tố ngẫu nhiên như đèn giao thông (ngẫu nhiên thời gian đỏ hoặc xanh). **Chính sách**: Xác suất lựa chọn hành động dựa trên mô hình dự đoán rủi ro của môi trường (ví dụ: Chọn đường đi ít tắc nghẽn nhất dựa trên dự báo giao thông). **Model**: Mô hình rủi ro và môi trường được xây dựng trước dựa trên dữ liệu giao thông.	**Ví dụ**: Robot học chơi cờ caro bằng PPO. **Chính sách**: Lựa chọn hành động theo xác suất (ví dụ: đặt quân cờ tại các vị trí có xác suất thắng cao). **Model**: Không cần xây dựng mô hình môi trường, chỉ sử dụng gradient để tối ưu hóa xác suất chọn hành động.

Bảng 2.1: Bảng kết hợp giữa Policy và RL Model

b, Xác định phần thưởng tích luỹ - G

$$\begin{aligned} \mathbf{G_t} &= \sum_{i=0}^{\infty} \gamma^{t+i} \mathbf{R}(\mathbf{S}_{t+i+1} = \mathbf{s}' | \mathbf{S}_{t+i} = \mathbf{s}) \\ \text{or:} \quad G_t &= \sum \gamma^t R_{t+1} \\ &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \end{aligned} \tag{2.2.7}$$

c, Tối ưu

Để tối ưu được một bài toán, đưa ra một hàm tổng quát cho giá trị cần tối ưu luôn là một bước quan trọng. Và ngay trong phần sau của phần này, phương trình Bellman được đưa ra nhằm xác định chính xác hàm cần tối ưu cho bài toán MDP.

Kết luận

Tóm lại, một bài toán MDP được xác định như sau:

Thành lập chuỗi Markov (Markov chain) bao gồm 1 bộ 5 phần tử:

$$(S, A, P(\cdot,\cdot), R(\cdot,\cdot), \gamma),$$

- **S**: Tập hợp các trạng thái của môi trường.
- **A**: Tập hợp các hành động mà agent có thể thực hiện.
- **R**: Hàm phần thưởng xác định phần thưởng nhận được sau khi thực hiện một hành động trong một trạng thái nhất định.
- **P**: Xác suất chuyển trạng thái, mô tả động lực học của môi trường, tức là xác suất chuyển từ trạng thái **s** hiện tại sang trạng thái $\mathbf{s}'$ sau khi thực hiện hành động **a**.

Sau đó là tìm một policy để tối ưu G:

- $\pi(\mathbf{s})$: Chính sách xác định hành động cần thực hiện dựa trên trạng thái hiện tại **s**.
- **G**: Tổng phần thưởng kỳ vọng trong dài hạn.

Từ đó, mục tiêu của các phương pháp tối ưu là thông qua MDP tìm một

hàm chính sách $\pi(\mathbf{s})$ có thể tối đa hóa kỳ vọng tổng phần thưởng tích luỹ **G**.

2.3. Phương trình Bellman (1950-1954)

Tiếp nối phần trên, phương trình Bellman đưa ra các phương trình tổng quát làm cơ sở để các thuật toán tối ưu như Lập trình động, Q-learning, PPO,... tìm ra đáp án cho phương trình.

2.3.1. Giới thiệu

Phương trình Bellman và lập trình động là những thành tựu xuất sắc của nhà toán học vĩ đại Richard Ernest Bellman (26/8/1920 – 19/3/1984). Ông không chỉ đặt nền móng cho lĩnh vực quy hoạch động mà còn để lại dấu ấn sâu rộng trong nhiều lĩnh vực khoa học và kỹ thuật, từ tối ưu hóa hệ thống điều khiển, xử lý tín hiệu, cho đến trí tuệ nhân tạo và học tăng cường. Phương trình Bellman, còn được gọi là phương trình quy hoạch động, đã trở thành công cụ không thể thiếu trong việc phân tích, đánh giá và tối ưu hóa các hệ thống ra quyết định phức tạp.

Phương trình Bellman là nền tảng lý thuyết cốt lõi trong việc giải quyết các bài toán Quy trình Quyết định Markov (MDP), mang đến công thức cơ bản để tính toán giá trị tối ưu cho mỗi trạng thái hoặc hành động trong hệ thống. Phương trình này được biểu diễn qua hai dạng chính: hàm giá trị trạng thái $V(s)$, đại diện cho giá trị tối ưu từ trạng thái s, và hàm giá trị hành động $Q(s,a)$, mô tả giá trị tối ưu khi thực hiện hành động a tại trạng thái s. Với việc phân rã bài toán thành các bài toán con, phương trình Bellman không chỉ giúp đánh giá mà còn hỗ trợ tối ưu hóa các quyết định trong các hệ thống phức tạp, đảm bảo tính tối ưu cục bộ dẫn đến tối ưu toàn cục.

2.3.2. Định nghĩa

Phương trình Bellman được xây dựng dựa trên nguyên tắc đệ quy, biểu diễn mối quan hệ giữa giá trị tối ưu tại một trạng thái cụ thể và giá trị tối ưu của các trạng thái liền kề có thể đạt được từ đó. Nó phân rã một vấn đề tối ưu hóa phức tạp thành các bài toán con nhỏ hơn, dễ giải quyết hơn.

Nguyên tắc tối ưu hóa của Bellman, hay *"Principle of Optimality,"* là nền tảng để xây dựng phương trình này. Nguyên tắc phát biểu rằng: *"Một chính sách tối ưu có tính chất rằng, bất kể trạng thái ban đầu và hành động đầu tiên là gì, chuỗi các hành động còn lại phải hình thành một chính sách tối ưu cho trạng thái đạt được từ hành động đầu tiên."*

Theo như nhận xét trên, "trạng thái" và "hành động" là thành phần quan trọng nhất trong việc xây dựng phương trình Bellman. Do vậy phương trình Bellman có 2 dạng, tương ứng với 2 biến quan trọng "trạng thái" và "hành động", đó là State-Value và Action-Value.

2.3.3. Phương trình Bellman

- Về cơ bản, Phương trình Bellman phân tích các hàm State-Value và Action-Value giá trị thành hai phần:

1. Phần thưởng ngay lập tức (R)

2. Hàm giá trị tương lai chiết khấu (G)

- Khi đó:

$\mathbf{V}(\mathbf{s})$ là giá trị kỳ vọng của tổng phần thưởng nhận được trong tương lai khi bắt đầu ở trạng thái **s** và làm theo chính sách

$\mathbf{Q}(\mathbf{s},\mathbf{a})$ là giá trị kỳ vọng của tổng phần thưởng nhận được trong tương lai khi bắt đầu ở trạng thái **s**, thực hiện hành động **a**, và sau đó làm theo chính sách π.

Từ đó, hàm giá trị trạng thái cho chúng ta biết mức độ tốt khi ở trạng thái đó. Trong khi, hàm giá trị hành động cho chúng ta biết nên thực hiện hành động tốt như thế nào trong trạng thái nhất định.

Hàm giá trị trạng thái có thể được phân tích thành:

$$\begin{aligned} V_\pi(s) &= \mathbb{E}[G_t|S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots |S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \ldots)|S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma G_{t+1}|S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma V_\pi(s_{t+1})|S_t = s] \end{aligned} \tag{2.3.1}$$

Hàm giá trị hành động có thể được phân tích thành:

$$Q_\pi(s,a) = \mathbb{E}\left[R_{t+1} + \gamma Q_\pi(s_{t+1}, a_{t+1}) \mid S_t = s, A_t = a\right] \tag{2.3.2}$$

2.3.4. Phương trình kỳ vọng Bellman

- Sau khi đã có hàm kì vọng của phương trình Bellman, phân tích rõ hơn là rất cần thiết để có thể xác định được thuật toán cần tối ưu. Bây giờ, phương trình Bellman được chuyển sang thành phương trình kỳ vọng Bellman như sau:

Hàm giá trị trạng thái $V_\pi(s)$ có những thông tin sau cơ bản sau:

1. Trạng thái hiện tại S

2. Nhiều hành động có thể được xác định bởi chính sách ngẫu nhiên $\pi(a|s)$

3. Mỗi hành động có thể được liên kết với một hàm giá trị hành động $Q_\pi(s,a)$ trả về giá trị của cụ thể đó hoạt động

4. Nhân các hành động có thể với hàm giá trị hành động và việc cộng chúng lại cho chúng ta biết được điều đó tốt như thế nào trong trạng thái đó

- Phương trình kết quả:

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

- Kết luận: Giá trị trạng thái = tổng (chính sách xác định hành động * giá trị hành động tương ứng)

Hàm giá trị hành động $Q_\pi(s,a)$ có những thông tin cơ bản sau:

- Với danh sách các hành động có thể thực hiện nhiều lần, có một danh sách các trạng thái tiếp theo có thể s' liên quan đến:

1. Hàm giá trị trạng thái $V_\pi(s')$.
2. Hàm xác suất chuyển tiếp $P^a_{ss'}$. Xác định trạng thái mà tác nhân có thể hiện thực hoá dựa trên hành động.
3. Thưởng R^a_s vì đã thực hiện hành động.

- Tổng hợp phần thưởng và hàm xác suất chuyển đổi liên quan đến hàm giá trị trạng thái cung cấp cho tác nhân một dấu hiệu về việc thực hiện các hành động như thế nào là tốt cho tình trạng hiện tại của tác nhân.

- Phương trình kết quả:

$$Q_\pi(s,a) = R^a_s + \gamma \sum_{s' \in S} P^a_{ss'} V_\pi(s')$$

- Kết luận: Giá trị hành động = phần thưởng + tổng (kết quả chuyển tiếp xác định trạng thái * giá trị trạng thái tương ứng).

Kết quả chuyển đổi cuối cùng:

- Thay thế hàm giá trị hành động vào hàm giá trị trạng thái:

$$V_\pi(s) = \sum_{a \in A} \pi(a|s)(R^a_s + \gamma \sum_{s' \in S} P^a_{ss'} V_\pi(s')) \tag{2.3.3}$$

- Thay thế hàm giá trị trạng thái vào hàm giá trị hành động:

$$Q_\pi(s,a) = R^a_s + \gamma \sum_{s' \in S} P^a_{ss'} \sum_{a' \in A} \pi(a'|s') Q(s',a') \tag{2.3.4}$$

2.3.5. Tối ưu Policy - Optimal Policy

Nhắc lại phần trước:

- Hàm giá trị trạng thái $V_\pi(s)$ là giá trị kỳ vọng của tổng phần thưởng thu được khi bắt đầu từ trạng thái s và tuân theo chính sách π.
- Hàm giá trị hành động $Q_\pi(s,a)$ là giá trị kỳ vọng của tổng phần thưởng khi thực hiện hành động a tại trạng thái s và sau đó tuân theo chính sách

π.

Từ đó, chính sách tối ưu π_* được định nghĩa như sau:

$$\pi_* = \arg\max_{\pi} V_\pi(s) = \arg\max_{\pi} Q_\pi(s,a)$$

Điều này có nghĩa là chính sách tối ưu sẽ chọn hành động tối ưu tại mỗi trạng thái sao cho giá trị kỳ vọng đạt cực đại.

Cách tính toán argmax của các hàm giá trị: Để xác định hành động tối ưu (tính toán arg max của hàm giá trị), cần dựa trên công thức cập nhật của hàm giá trị hành động $Q_\pi(s,a)$, được định nghĩa bởi:

$$Q_\pi(s,a) = \mathbb{E}\left[R_{t+1} + \gamma Q_\pi(s_{t+1}, a_{t+1}) \,|\, S_t = s, A_t = a\right]$$

Trong đó:

- R_{t+1} là phần thưởng nhận được ngay sau khi thực hiện hành động a tại trạng thái s.
- γ là hệ số chiết khấu (discount factor), phản ánh mức độ ưu tiên phần thưởng trong tương lai.
- s_{t+1} và a_{t+1} lần lượt là trạng thái và hành động tiếp theo theo chính sách π.
- $\mathbb{E}$ là kỳ vọng toán học, biểu diễn giá trị trung bình của các phần thưởng dự kiến.

Công thức trên cho phép chúng ta tính toán giá trị kỳ vọng của một hành động cụ thể, từ đó tìm được hành động tối ưu tại mỗi trạng thái bằng cách tìm arg max.

Tóm lại, quá trình học một chính sách tối ưu bao gồm:

1. Xác định giá trị tối ưu tại mỗi trạng thái và hành động thông qua các hàm giá trị.

2. Sử dụng các hàm giá trị này để chọn hành động tối ưu bằng cách giải bài toán arg max.

2.3.6. Phương trình tối ưu Bellman

- Ở trong phần trước, chính sách tối ưu π_* –> giá trị trạng thái tối ưu và hàm giá trị hành động –> argmax của hàm giá trị

$$\pi_* = \arg\max_\pi V_\pi(s) = \arg\max_\pi Q_\pi(s,a)$$

- Cuối cùng với phương trình kỳ vọng Bellman được suy ra từ Bellman Các phương trình, chúng ta có thể suy ra các phương trình cho argmax của giá trị của chúng ta chức năng

- Hàm giá trị trạng thái tối ưu

$$V_*(s) = \arg\max_\pi V_\pi(s)$$

- Khi:

$$V_\pi(s) = \sum_{a\in A} \pi(a|s)(R_s^a + \gamma \sum_{s'\in S} P_{ss'}^a V_\pi(s'))$$

- Thì phương trình tối ưu của State-Value là:

$$V_*(s) = \max_{a\in A}(R_s^a + \gamma \sum_{s'\in S} P_{ss'}^a V_*(s')))$$

- Hàm giá trị hành động tối ưu

$$Q_*(s) = \arg\max_\pi Q_\pi(s)$$

- Khi:

$$Q_\pi(s,a) = R_s^a + \gamma \sum_{s'\in S} P_{ss'}^a \sum_{a'\in A} \pi(a'|s')Q(s',a')$$

- Thì phương trình tối ưu của Action-Value là:

$$Q_*(s,a) = R_s^a + \gamma \sum_{s'\in S} P_{ss'}^a max_{a'\in A} Q_*(s',a')$$

2.4. Tối Ưu Hóa Chính Sách Gần Nhất (PPO) 2017

Trước khi đi vào phần lý thuyết trọng tâm của luận văn, một nhận xét tóm gọn sẽ được đưa ra nhằm bắt đầu lý thuyết đó một cách hiệu quả: "Thuật toán" PPO sinh ra để "giải phương trình" Bellman, phương trình Bellman sinh ra để "phương trình hóa giá trị tối ưu" và "mô tả đệ quy" bài toán MDP. MDP sinh ra để "mô hình hóa toán học" vấn đề ra quyết định trong thực tế.

2.4.1. Giới thiệu

Việc giải các phương trình Bellman là bước quan trọng trong việc xác định chính sách tối ưu cho bài toán MDP, giúp định hướng hành động theo giá trị mong muốn.

Để giải phương trình Bellman một cách hiệu quả, cần đến các phương pháp lập trình động. Hai kỹ thuật kinh điển là Value Iteration và Policy Iteration. Value Iteration liên tục cập nhật giá trị của từng trạng thái cho đến khi hội tụ, trong khi Policy Iteration kết hợp cải thiện chính sách và cập nhật giá trị để tìm ra chính sách tối ưu. Cả hai phương pháp đều khai thác cấu trúc đệ quy của phương trình Bellman, giúp giảm thiểu chi phí tính toán, đặc biệt là trong các bài toán có quy mô lớn.

Tuy nhiên, trong các bài toán phức tạp hơn, đặc biệt trong robot tự hành và trí tuệ nhân tạo, các phương pháp truyền thống như Value Iteration và Policy Iteration có thể gặp phải hạn chế. Vì vậy, các thuật toán hiện đại như Q-Learning, SARSA, và những phương pháp tiên tiến như Deep Q-Network (DQN) hay Proximal Policy Optimization (PPO) đã được phát triển. Những thuật toán này kết hợp mạng nơ-ron và gradient chính sách để mở rộng khả năng giải quyết các bài toán có không gian trạng thái lớn và phức tạp, từ đó giúp tăng cường hiệu quả cho học tăng cường.

2.4.2. Định nghĩa

Proximal Policy Optimization (PPO) là một thuật toán học tăng cường (Reinforcement Learning - RL) thuộc nhóm **Policy Gradient**. Mục tiêu của PPO là tối ưu hóa chính sách (policy) của agent để tối đa hóa tổng lợi nhuận (reward) mà agent có thể thu được khi tương tác với môi trường. Trong PPO, thuật toán được phát triển từ cơ sở của phương pháp **Actor-Critic**, kết hợp giữa việc ước lượng giá trị (value function) và tối ưu hóa trực tiếp chính sách, giúp đạt được hiệu suất học tập cao và tính ổn định trong quá trình huấn luyện.

1. Policy Gradient trong PPO

PPO là một thuật toán Policy Gradient, có nghĩa là nó tối ưu hóa chính sách trực tiếp thay vì tính toán giá trị của các trạng thái hay hành động. Trong các thuật toán Policy Gradient, ta tối ưu hóa một hàm mục tiêu $J(\theta)$ để tìm ra chính sách tốt nhất π_θ thông qua việc tính toán gradient của hàm lợi nhuận với tham số θ của chính sách.

Hàm mục tiêu trong Policy Gradient có thể được viết dưới dạng:

$$J(\theta) = \mathbb{E}_{\pi_\theta}[R(\tau)] \tag{2.4.1}$$

Trong đó:

- $\mathbb{E}_{\pi_\theta}$ là kì vọng tính theo π_θ
- $R(\tau)$ là tổng lợi nhuận mà agent nhận được trong một chuỗi hành động τ

Thuật toán tối ưu chính sách π_θ bằng cách tính toán gradient của $J(\theta)$ và điều chỉnh tham số θ theo hướng làm tăng $J(\theta)$ khi $J(\theta)$ là một hàm khả vi có cực trị toàn cục đạt cự đại. Cập nhật chính sách được thực hiện theo công thức:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$

Trong đó, α là tỷ lệ học (learning-rate) và $\nabla_\theta J(\theta)$ là gradient của hàm mục tiêu đối với tham số θ

2. Policy Gradient trong PPO

Hàm mục tiêu $J(\theta)$ trong Policy Gradient có thể có các cực trị, nhưng không phải lúc nào cũng có cực đại rõ ràng. Điều này là do tính phức tạp của không gian tham số và môi trường học, dẫn đến việc có thể có nhiều điểm cực trị (cực đại, cực tiểu hoặc điểm yên ngừng). Để đảm bảo $J(\theta)$ có cực trị cực đại, nó phải thỏa mãn một số điều kiện lý thuyết:

(a) Liên Tục và Khả Vi Hàm mục tiêu $J(\theta)$ phải là một hàm liên tục và khả vi đối với các tham số θ. Trong trường hợp của PPO (và các thuật

toán Policy Gradient khác), chính sách π_θ thường được mô hình hóa bằng mạng nơ-ron, giúp tính toán gradient của $J(\theta)$ một cách dễ dàng.

(b) Điều Kiện Cực Trị (Stationary Point)

Để tìm các điểm cực trị của $J(\theta)$, ta tính gradient và tìm giá trị θ sao cho gradient bằng 0:

$$\nabla_\theta J(\theta) = 0$$

Những giá trị θ tại đó gradient bằng 0 sẽ là các điểm cực trị của hàm mục tiêu. Tuy nhiên, không phải tất cả các điểm này đều là cực đại, có thể có các điểm cực tiểu hoặc các điểm yên ngừng (saddle points).

(c) Điều Kiện Để Tìm Cực Đại Để đảm bảo điểm cực trị tìm được là cực đại, ta cần kiểm tra điều kiện bậc hai, tức là xem xét dấu của ma trận Hessian (ma trận đạo hàm bậc hai) tại điểm cực trị:

$$H = \frac{\partial^2 J(\theta)}{\partial \theta^2}$$

Nếu ma trận Hessian có các giá trị riêng âm, thì điểm đó là cực đại. Tuy nhiên, trong thực tế, các bài toán RL thường rất phức tạp, và hàm mục tiêu có thể có nhiều điểm cực trị (bao gồm cực đại, cực tiểu và điểm yên ngừng), dẫn đến việc tối ưu hóa $J(\theta)$ không phải lúc nào cũng đơn giản.

3. Actor-Critic trong PPO

PPO sử dụng mô hình actor-critic, trong đó có hai thành phần chính:

- **Actor**: Là phần thực hiện tối ưu hóa chính sách (policy). Actor quyết định hành động của agent tại mỗi trạng thái dựa trên chính sách $\pi_\theta(a|s)$ (với tham số θ). Actor học cách chọn hành động sao cho đạt được lợi nhuận tối đa.
- **Critic**: Đánh giá chính sách của Actor bằng cách ước lượng **hàm giá trị** $\mathbf{V}(\mathbf{s})$, tức là giá trị kỳ vọng của tổng lợi nhuận mà agent sẽ nhận được khi bắt đầu từ trạng thái **s**. Critic giúp cải thiện sự chính xác trong việc đánh giá và cập nhật chính sách của Actor.

Sự kết hợp giữa Actor và Critic giúp PPO học hiệu quả hơn, với Critic cung cấp các ước lượng cần thiết để điều chỉnh chính sách trong khi Actor đảm nhận vai trò quyết định hành động.

4. Lý do chọn ước lượng và tối ưu $V(s)$ thay vì $Q(s,a)$

a. Tăng phương sai trong ước lượng

Khi sử dụng $Q(s,a)$, Q ước lượng giá trị cho từng hành động a tại trạng thái s. Tuy nhiên, $Q(s,a)$ thường có phương sai lớn hơn $V(s)$ vì nó phụ thuộc vào cả trạng thái s và hành động a.

Trong PPO, lợi thế (advantage) $A(s,a)$ được định nghĩa như:

$$A(s,a) = Q(s,a) - V(s),$$

giúp trung hòa ảnh hưởng của trạng thái và chỉ tập trung vào giá trị tương đối giữa các hành động. Điều này giúp giảm phương sai khi tính toán gradient và tối ưu chính sách.

Nếu chỉ sử dụng $Q(s,a)$, gradient sẽ dao động mạnh hơn, dẫn đến việc học không ổn định.

b. Phức tạp hóa mô hình Actor-Critic

Trong mô hình Actor-Critic:

- Critic được thiết kế để đánh giá hiệu quả tổng thể của trạng thái bằng $V(s)$. Nếu thay $V(s)$ bằng $Q(s,a)$, Critic phải dự đoán giá trị cho từng cặp (s,a), làm tăng độ phức tạp tính toán.
- Actor sẽ cần thông tin từ $Q(s,a)$ của tất cả các hành động, không chỉ hành động hiện tại, gây tốn kém chi phí tính toán hơn.

c. Không tối ưu trong không gian hành động liên tục

Với không gian hành động liên tục (như robot), việc ước lượng $Q(s,a)$ cho tất cả các hành động a là không thực tế, do không thể duyệt qua mọi giá trị khả dĩ của a.

Ngược lại, $A(s,a)$ chỉ yêu cầu giá trị $Q(s,a)$ cho hành động hiện tại và $V(s)$, giúp giảm đáng kể chi phí tính toán.

d. Mất tính ổn định của chính sách

PPO sử dụng hàm loss dựa trên lợi thế $A(s,a)$ và áp dụng clipping:

$$L^{\text{CLIP}}(\theta) = \mathbb{E}\left[\min\left(r_t(\theta)\cdot A(s,a), \text{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\cdot A(s,a)\right)\right],$$

với $r_t(\theta) = \frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}$.

Nếu thay thế $A(s,a)$ bằng $Q(s,a)$, clipping trở nên khó áp dụng vì $Q(s,a)$ không có tính chất tương tự lợi thế (được trung hòa và giảm nhiễu).

e. Ưu điểm của lợi thế $A(s,a)$

Sử dụng lợi thế $A(s,a)$ giúp tách biệt hai yếu tố:

- Giá trị tổng thể của trạng thái $V(s)$: giúp Actor không bị phụ thuộc hoàn toàn vào hành động cụ thể.
- Giá trị tương đối $Q(s,a) - V(s)$: cho phép Actor tập trung cải thiện chính sách dựa trên các hành động tốt hơn trung bình.

f. Các thuật toán dựa trên $Q(s,a)$ đã tồn tại

Các thuật toán như **Deep Deterministic Policy Gradient (DDPG)** và **Soft Actor-Critic (SAC)** đã sử dụng $Q(s,a)$ để hướng dẫn Actor. Tuy nhiên:

- Các thuật toán này yêu cầu một mạng riêng để tối ưu $Q(s,a)$ dựa trên phương trình Bellman.
- Actor không sử dụng trực tiếp hàm loss dựa trên $Q(s,a)$ như trong PPO.

g. Kết luận

Thay $V(s)$ bằng $Q(s,a)$ và sử dụng hàm loss $\pi(a|s)\cdot Q(s,a)$ sẽ làm tăng phương sai, phức tạp hóa tính toán và mất ổn định chính sách. Cách tiếp cận dựa trên $A(s,a)$ trong PPO giảm nhiễu, đơn giản hóa Critic, và duy trì ổn định khi học chính sách.Nếu sử dụng $Q(s,a)$, các thuật toán

như DDPG hoặc SAC sẽ phù hợp hơn, nhưng chúng yêu cầu thiết kế khác biệt để xử lý hiệu quả.

2.4.3. Công thức chính

1. Công thức hàm

Hàm mất mát chính trong thuật toán PPO với **Clipping Objective** được biểu diễn như sau:

$$L^{\text{Clip}}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta) \cdot A_t, \text{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon) \cdot A_t \right) \right] \qquad (2.4.2)$$

Trong đó:

- $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$: Tỷ lệ giữa xác suất hành động hiện tại và xác suất hành động cũ.
- $\pi_\theta(a_t|s_t)$: Chính sách hiện tại, biểu diễn phân phối xác suất.
- $A_t = G_t - V(s_t)$: Lợi thế tại thời điểm t, với:
  - G_t: Tổng phần thưởng thu được từ thời điểm t.
  - $V(s_t)$: Giá trị dự đoán của trạng thái s_t.
- ε: Hệ số điều chỉnh clip, thường có giá trị nhỏ (ví dụ: 0.1 hoặc 0.2).
- Note: Công thức chi tiết của G và V sẽ được nêu chi tiết trong chương sau!

2. Ý nghĩa các biến

- **Hàm chính sách** $\pi_\theta(a_t|s_t)$: Đây là một phân phối xác suất phụ thuộc vào μ (mean) và σ (standard deviation), tạo thành một phân phối bậc hai (parabolic).
  - Nếu $A_t > 0$: Gradient tăng xác suất lựa chọn hành động a_t, giúp chính sách ưu tiên các hành động có lợi thế cao.
  - Nếu $A_t < 0$: Gradient giảm xác suất lựa chọn hành động a_t, loại bỏ các hành động kém hiệu quả.
- **Lợi thế** $A_t = G_t - V(s_t)$: Phản ánh độ chênh lệch giữa phần thưởng thực nhận (G_t) và giá trị dự đoán ($V(s_t)$).
  - Khi $G_t > V(s_t)$: Hành động tốt hơn dự đoán, xác suất lựa

chọn hành động đó tăng.

- Khi $G_t < V(s_t)$: Hành động kém hơn dự đoán, xác suất lựa chọn hành động đó giảm.

- **Clipping Objective**: Giới hạn giá trị $r_t(\theta)$ trong khoảng $[1-\varepsilon, 1+\varepsilon]$, tránh các cập nhật quá mức, đảm bảo sự ổn định trong việc tối ưu hóa. Điều này ngăn không cho xác suất lựa chọn một hành động tăng quá nhanh, tránh việc chính sách quá phụ thuộc vào một hành động cụ thể và giảm thiểu nguy cơ mất tính tổng quát của agent.

3. Vấn đề nếu không có clipping

a) Vấn đề với sự ổn định của gradient

Một trong những điều quan trọng trong PPO là việc tối ưu hàm giá trị mục tiêu (objective function) mà không làm cho gradient quá lớn, dẫn đến các cập nhật quá mức và không ổn định. Khi không có clipping, nếu tỉ lệ $r_t(\theta)$ tăng quá nhiều (ví dụ $r_t(\theta) > 1+\varepsilon$), gradient của hàm mục tiêu sẽ lớn hơn rất nhiều và khiến cho tham số θ thay đổi quá nhanh. Điều này có thể làm cho chính sách học được trở nên không ổn định và không thể hội tụ.

b) Rủi ro làm mất tính tổng quát (overfitting)

Khi giá trị $r_t(\theta)$ vượt quá ngưỡng cho phép (tức là không bị clipping), mô hình sẽ ưu tiên quá mức cho hành động nào đó mà có xác suất cao hơn rất nhiều so với chính sách cũ, dù cho hành động đó có thể không phải là lựa chọn tốt nhất. Điều này có thể dẫn đến việc chính sách bị quá phụ thuộc vào một hành động cụ thể trong một số tình huống, khiến cho mô hình mất tính tổng quát và không thể thích nghi tốt với các tình huống khác.

4. Giải thích vấn đề về dấu của A(t) và Clipping

Lợi thế là tích cực: Giả sử lợi thế cho cặp trạng thái-hành động đó là tích cực, trong trường hợp đó, đóng góp của nó cho mục tiêu giảm xuống

$$L(s,a,\theta_k,\theta) = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, (1+\varepsilon)\right) A^{\pi_{\theta_k}}(s,a).$$

Vì lợi thế là tích cực, mục tiêu sẽ tăng nếu hành động có nhiều khả năng xảy ra hơn—tức là nếu $\pi_\theta(a|s)$ tăng. Nhưng min trong thuật ngữ này đặt ra giới hạn cho mức độ mục tiêu có thể tăng. Khi $\pi_\theta(a|s) > (1+\varepsilon)\pi_{\theta_k}(a|s)$, min có hiệu lực và thuật ngữ này đạt đến mức trần là $(1+\varepsilon)A^{\pi_{\theta_k}}(s,a)$. Do đó, chính sách mới không được hưởng lợi khi đi xa khỏi chính sách cũ.

Lợi thế là tiêu cực: Giả sử lợi thế cho cặp trạng thái-hành động đó là tiêu cực, trong trường hợp đó, đóng góp của nó cho mục tiêu giảm xuống

$$L(s,a,\theta_k,\theta) = \max\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, (1-\varepsilon)\right) A^{\pi_{\theta_k}}(s,a).$$

Vì lợi thế là tiêu cực, mục tiêu sẽ tăng nếu hành động trở nên ít có khả năng xảy ra hơn—tức là nếu $\pi_\theta(a|s)$ giảm. Nhưng mức tối đa trong thuật ngữ này đặt ra giới hạn cho mức mục tiêu có thể tăng. Khi $\pi_\theta(a|s) < (1-\varepsilon)\pi_{\theta_k}(a|s)$, mức tối đa có hiệu lực và thuật ngữ này đạt đến mức trần là $(1-\varepsilon)A^{\pi_{\theta_k}}(s,a)$. Do đó, một lần nữa, chính sách mới không được hưởng lợi khi đi xa khỏi chính sách cũ.

5. Hàm xác suất phân phối chuẩn

Hàm xác suất $\pi_\theta(a_t|s_t)$ được định nghĩa như sau:

$$\pi_\theta(a_t|s_t) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{(a_t-\mu)^2}{2\sigma^2}}$$

Ở đây:

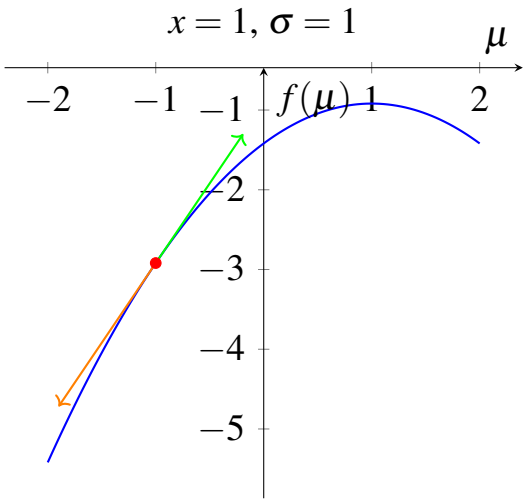
- μ là trung tâm (mean) của phân phối.
- σ là độ lệch chuẩn (standard deviation), quyết định độ rộng của phân phối.

6. Hàm mục tiêu và gradient

Hàm mục tiêu trong tối ưu chính sách:

$$J(\theta) = \mathbb{E}[\log \pi_\theta(a_t|s_t) \cdot A_t]$$

$$\begin{aligned}
\log \pi_\theta(a_t|s_t) &= \log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) + \log\left(\exp\left(-\frac{(a_t-\mu)^2}{2\sigma^2}\right)\right) \\
&= \log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) + \left(-\frac{(a_t-\mu)^2}{2\sigma^2}\right) \\
&= -\frac{1}{2}\log(2\pi\sigma^2) - \frac{(a_t-\mu)^2}{2\sigma^2} \\
&= -\frac{1}{2}\log(2\pi) - \log(\sigma) - \frac{(a_t-\mu)^2}{2\sigma^2}
\end{aligned}$$



Hình 2.1: Hàm Log của phân phối chuẩn, với gradient

Gradient theo μ:

$$\nabla_\mu J(\theta) = A_t \cdot \frac{(a_t-\mu)}{\sigma^2}$$

Gradient theo σ:

$$\nabla_\sigma J(\theta) = A_t \left[\frac{(a_t-\mu)^2}{\sigma^3} - \frac{1}{\sigma}\right]$$

7. Cách gradient điều chỉnh μ

Vì Đạo hàm trong không gian 1 chiều luôn chỉ hướng tăng của biến và Gradient trong không gian nhiều chiều luôn chỉ hướng tăng nhanh nhất của hàm số đối với tất cả các biến.

Điều chỉnh μ

Khi $A_t > 0$ (vetor màu xanh) gradient descent là:

$$\mu \leftarrow \mu + \alpha \cdot \nabla_\mu J(\theta)$$

Khi $A_t < 0$ (vetor màu cam) gradient descent là:

$$\mu \leftarrow \mu - \alpha \cdot \nabla_\mu J(\theta)$$

Hình dung gradient như độ dốc của ngọn đồi:

- Gradient dương: Dốc hướng lên về phía phải $\rightarrow$ đi về phía phải để μ tiến gần tới a_t.
- Gradient âm: Dốc hướng lên về phía trái $\rightarrow$ đi về phía trái để μ tiến ra xa a_t.

Kết quả:

Khi A>0 thì tăng xác suất chọn hành động a hay kéo μ tiến gần tới a_t và là cho giá trị của hàm phân phối chuẩn tăng hay tăng xác suất chọn hành động a.

Khi A<0 thì giảm xác suất chọn hành động a hay kéo μ tiến gần tới a_t và là cho giá trị của hàm phân phối chuẩn tăng hay giảm xác suất chọn hành động a.

8. Kết luận

Thuật toán PPO với **Clipping Objective** hoạt động dựa trên nguyên tắc:

> "Khi $A_t > 0$, hàm mất mát L^{Clip} làm tăng xác suất lựa chọn hành động a_t, bởi hành động này đã mang lại phần thưởng lớn hơn giá trị dự đoán $V(s_t)$ của Critic. Điều này giúp mạng Actor ưu tiên các hành động hiệu quả hơn tại trạng thái s_t. Ngược lại, nếu $A_t < 0$, xác suất lựa chọn các hành động kém hiệu quả sẽ giảm đi."

CHƯƠNG 3: MÔ HÌNH ĐỀ XUẤT

Trong chương này, tác giả sẽ đưa ra chi tiết về các bộ phần tử được thiết trong mô hình dữ tương ứng với từng bộ phần tử có trong mô hình Markov. Tiếp đó, là sơ đồ thiết kế của mô hình huấn luyện robot Cassie sử dụng thuật toán PPO, mô tả luồng dữ liệu được xử lý và hoạt động thành một chu trình kín từ lúc thu thập dữ liệu đến huấn luyện và lặp lại. Phần tiếp theo của chương sẽ đưa ra các công thức có trong dự án.

3.1. Triển khai Mô hình Markov

Phần này trình bày chi tiết các giá trị có trong từng không gian trạng thái của mô hình Markov, từ đó có thể áp dụng giải thuật PPO để tìm ra lời giải cho mô hình.

Nhắc lại công thức của quá trình Markov:

Một quá trình quyết định Markov có thể được biểu diễn như một bộ 5 phần tử:

$$(\mathbf{S}, \mathbf{A}, \mathbf{P}(\cdot,\cdot), \mathbf{R}(\cdot,\cdot), \gamma), \tag{3.1.1}$$

Vì PPO là một thuật toán trong môi trường free-model tức là không có xác suất chuyển trạng thái nên sẽ không có P.

Vậy nên quyết định Markov được biểu diễn trong mô hình đề xuất gồm bộ 4 phân tử sau:

$$(\mathbf{S}, \mathbf{A}, \mathbf{R}(\cdot,\cdot), \gamma), \tag{3.1.2}$$

Sau đó để tối ưu được quá trình ra quyết định, "Policy- π" là thành phần tiếp theo được xác định cùng với "Environment" thích hợp. Và để có thể thiết kế mục tiêu cho quá trình tối ưu hóa, $G(t)$ và $A(t)$, $V(t)$ sẽ được nêu ra trong phần dưới đây.

3.1.1. Môi trường tương tác cho agent

Vì agent được mô phỏng bằng Mujoco, nên tập trạng thái, hành động, sẽ được đọc và điều khiển thông qua trình mô phỏng Mujoco.

Môi trường "Environment" trong mô hình đề xuất là trình mô phỏng Mujoco, được tích hợp vào trong chương trình chính thông qua thư viện Mujoco.

Khi agent tương tác với môi trường thì mô hình của môi trường được triển khai là free-model, chính vì thế sẽ không có xác suất chuyển trạng thái, thay vào đó là sự tương tác qua lại giữa agent và môi trường. Agent sẽ sử dụng những bộ dẫn chuyền cơ học để di chuyển trong môi trường. Môi trường đáp lại bằng cách tạo ra trọng lực (g=9.8m/s), lực ma sát, địa hình bằng phẳng,... Từ đó agent có thể xác định được môi trường đã phản hồi lại thông tin gì thông qua các cảm biến như: Cảm biến lực, cảm biến va chạm, cảm biến imu (IMU (Inertial Measurement Unit) là thiết bị được kết hợp từ hai bộ cảm biến Accelerometer (cảm biến gia tốc) và cảm biến Gyroscope (cảm biến con quay hồi chuyển), và có thể gắn thêm các cảm biến ngoại vi khác).

3.1.2. Bộ trạng thái S_t

Bộ trang thái được định nghĩa bên trong chương trình như sau:

```
@dataclass
class AgentState:
    joint_positions: np.ndarray
    joint_velocities: np.ndarray
    left_foot_touch: np.ndarray
    right_foot_touch: np.ndarray
    left_foot_force: np.ndarray
    right_foot_force: np.ndarray
    left_foot_speed: np.ndarray
    right_foot_speed: np.ndarray
    pelvis_orientation: np.ndarray
    pelvis_velocity: np.ndarray
    pelvis_angular_velocity: np.ndarray
```

```
    pelvis_linear_acceleration: np.ndarray
```

Trong đó:

joint_positions: np.ndarray # (6D vector) Vị trí góc quay của khớp

joint_velocities: np.ndarray # (6D vector) Vận tốc góc của khớp

left_foot_touch: np.ndarray # (1D vector) Tín hiệu tiếp xúc chân trái của pelvis

right_foot_touch: np.ndarray # (1D vector) Tín hiệu tiếp xúc chân phải

left_foot_force: np.ndarray # (3D vector) Lực tiếp xúc chân trái của pelvis

right_foot_force: np.ndarray # (3D vector) Lực tiếp xúc chân phải

left_foot_speed: np.ndarray # (3D vector) Vận tốc chân trái của pelvis

right_foot_speed: np.ndarray # (3D vector) Vận tốc chân phải của pelvis

pelvis_orientation: np.ndarray # (4D vector) Hướng xoay của pelvis

pelvis_velocity: np.ndarray # (3D vector) Vận tốc của pelvis

pelvis_angular_velocity: np.ndarray # (3D vector) Vận tốc xoay của pelvis

pelvis_linear_acceleration: np.ndarray # (3D vector) Gia tốc của pelvis

Những dữ liệu trên được đọc từ các cảm biến gắn trên agent-Cassie và được khai báo trong file XML như sau:

Listing 1: Sensor Configuration

```
<sensor>
    <!-- Joint positions of actuator -->
    <jointpos name="left-hip-roll-output" joint="left-
      hip-roll" noise="2e-4"/>
    <jointpos name="left-hip-yaw-output" joint="left-
      hip-yaw" noise="2e-4"/>
    <jointpos name="left-hip-pitch-output" joint="left-
      hip-pitch" noise="2e-4"/>
    <jointpos name="left-knee-output" joint="left-knee"
       noise="2e-4"/>
    <jointpos name="left-foot-output" joint="left-foot"
       noise="2e-4"/>
```

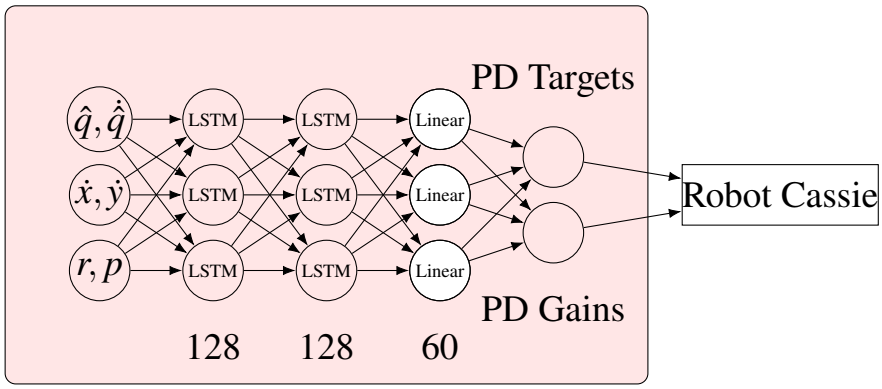
```
<jointpos name="right-hip-roll-output" joint="right
  -hip-roll" noise="2e-4"/>
<jointpos name="right-hip-yaw-output" joint="right-
  hip-yaw" noise="2e-4"/>
<jointpos name="right-hip-pitch-output" joint="
  right-hip-pitch" noise="2e-4"/>
<jointpos name="right-knee-output" joint="right-
  knee" noise="2e-4"/>
<jointpos name="right-foot-output" joint="right-
  foot" noise="2e-4"/>
<!-- Joint velocities of actuator -->
<jointvel name="left-hip-roll-velocity" joint="left
  -hip-roll" noise="2e-4"/>
<jointvel name="left-hip-yaw-velocity" joint="left-
  hip-yaw" noise="2e-4"/>
<jointvel name="left-hip-pitch-velocity" joint="
  left-hip-pitch" noise="2e-4"/>
<jointvel name="left-knee-velocity" joint="left-
  knee" noise="2e-4"/>
<jointvel name="left-foot-velocity" joint="left-
  foot" noise="2e-4"/>
<jointvel name="right-hip-roll-velocity" joint="
  right-hip-roll" noise="2e-4"/>
<jointvel name="right-hip-yaw-velocity" joint="
  right-hip-yaw" noise="2e-4"/>
<jointvel name="right-hip-pitch-velocity" joint="
  right-hip-pitch" noise="2e-4"/>
<jointvel name="right-knee-velocity" joint="right-
  knee" noise="2e-4"/>
<jointvel name="right-foot-velocity" joint="right-
  foot" noise="2e-4"/>
<!-- Joint positions -->
```

```
<jointpos name="left-shin-output" joint="left-shin"
    noise="2e-4"/>
<jointpos name="left-tarsus-output" joint="left-
   tarsus" noise="2e-4"/>
<jointpos name="right-shin-output" joint="right-
   shin" noise="2e-4"/>
<jointpos name="right-tarsus-output" joint="right-
   tarsus" noise="2e-4"/>
<!-- Joint velocities -->
<jointvel name="left-shin-velocity" joint="left-
   shin" noise="2e-4"/>
<jointvel name="left-tarsus-velocity" joint="left-
   tarsus" noise="2e-4"/>
<jointvel name="right-shin-velocity" joint="right-
   shin" noise="2e-4"/>
<jointvel name="right-tarsus-velocity" joint="right
   -tarsus" noise="2e-4"/>
<!-- Force sensors -->
<touch name="left-foot-touch" site="left-heel-foot"
    noise="1e-3" cutoff="50"/>
<touch name="right-foot-touch" site="right-heel-
   foot" noise="1e-3" cutoff="50"/>
<force name="left-foot-force" site="left-ankle"
   noise="1e-3" cutoff="50"/>
<force name="right-foot-force" site="right-ankle"
   noise="1e-3" cutoff="50"/>
<!-- Velocity sensors -->
<velocimeter name="left-foot-speed" site="left-
   ankle" noise="1e-3" cutoff="50"/>
<velocimeter name="right-foot-speed" site="right-
   ankle" noise="1e-3" cutoff="50"/>
<!-- Pelvis sensors -->
```

```
    <framequat name="pelvis-orientation" objtype="site"
        objname="imu"/>
    <accelerometer name="pelvis-linear-acceleration"
       site="imu" noise="1e-2" cutoff="157"/>
    <velocimeter name="pelvis-velocity" site="imu"
       noise="1e-3" cutoff="50"/>
    <gyro name="pelvis-angular-velocity" site="imu"
       noise="5e-4" cutoff="34.9"/>
</sensor>
```

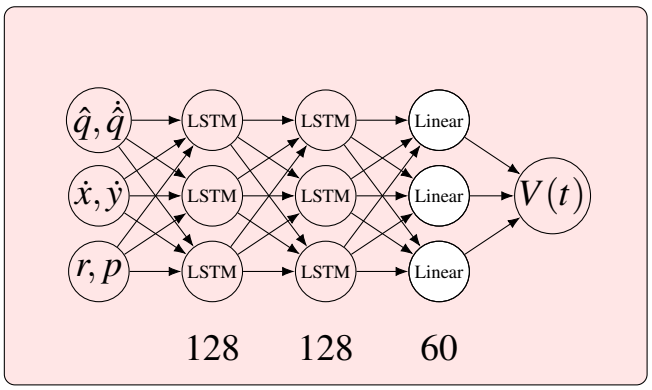
3.1.3. Thiết kế Policy $\pi_\theta(a|s)$

Policy là một mạng neuron có 2 lớp LSTM và 1 lớp Liner ở cuối, sau đó sẽ tách ra thành 2 phần: 10 node đầu sẽ là 10 giá trị $r(t)$ tương ứng với 10 vị trí mục tiêu của 10 khớp và sẽ được scale với giá trị min-max của góc quay từng khớp. 30 giá trị cuối là 30 giá trị của sigma, sẽ đi qua một lớp min giúp sigma luôn dương và lớn hơn giá trị mong muốn. Và 20 lớn còn lại là của K_p và K_d sẽ được giữ nguyên khi đi qua lớp Linear. Và sau đây là thiết kế cho mạng Actor:



3.1.4. Thiết kế Ước lượng giá trị state-value $V(t)$

Critic là một mạng được thiết kế các lớp layer LTSM giống Policy nhưng đầu ra cuả mạng chỉ có 1 giá trị là $V(t)$. Và sau đây là thiết kế cho mạng Critic:



3.1.5. Bộ hành động A_t

Bộ hành động được tạo ra gián tiếp từ Policy ở phần trước (3.1.3), và được tạo ra trực tiếp thông qua bước lấy mẫu theo phân phối chuẩn. Bộ hành động là bộ 30 tín hiệu điều khiển PD (Proportional- derivative) gồm 10 biến $r(t)$: là vị trí mục tiêu của actuator, 10 biến K_p là hệ số tỉ lệ cho vị trí, và 10 biến K_d là hệ số tỉ lệ cho vận tốc góc. Vì thế mô hình đề xuất sẽ thiết kế đầu ra mạng neuron cho actor có số node là 60, tương ứng với 30 giá trị μ và 30 giá trị σ. Và 60 đầu ra này sẽ làm đầu vào cho Policy được thiết kế là một phân phối chuẩn, sau đó policy sẽ lấy mẫu với 30 cặp μ và σ để cho ra 30 action. 30 action này chính là tín hiệu điều khiển cho bộ điều khiển PD PD (Proportional–Derivative-Bộ điều khiển tỉ lệ-đạo hàm)

Từ 30 biến trên và công thức điều khiển PD, chương trình sẽ tính toán được lực momen xoắn cần thiết cho 10 actuator được gắn ở 10 khớp chân tương ứng với 5 khớp mỗi chân (chân trái, chân phải), để điều khiển robot:

Công thức bộ điều khiển PD là:

$$u(t) = K_p e(t) + K_d \frac{d}{dt} e(t) \tag{3.1.3}$$

Trong đó:

- $u(t)$ là tín hiệu điều khiển (Momen xoắn).
- K_p là hệ số tỉ lệ (proportional gain).
- K_d là hệ số vi phân (derivative gain).

- $e(t) = r(t) - y(t)$ là sai số giữa tín hiệu tham chiếu, $r(t)$ là vị trí mục tiêu, $y(t)$ là vị trí hiện tại.
- $\frac{d}{dt}e(t)$ là đạo hàm của vị trí, hay vận tốc góc của actutor

Những actuator (Bộ truyền động) được gắn tại các khớp của robot, giúp điều khiển momen xoay, tạo ra lực xoay làm các khớp của robot có thể quay và robot có thể di chuyển.

Khi đã lấy được tín hiệu của 10 momen xoắn, agent sẽ thực 10 điều khiển actuator tại 10 vị trí khớp. Và việc điều khiển này được thực hiện qua API của thư viện Mujoco trong python như sau:

```
    def control_agent(self, control_signal):
        for i, (torque, idx) in enumerate(zip(
            control_signal, self.atr_ctrl_map.values())):
              self.agt_data.ctrl[idx] = torque
```

3.1.6. Bộ phần thưởng R_t

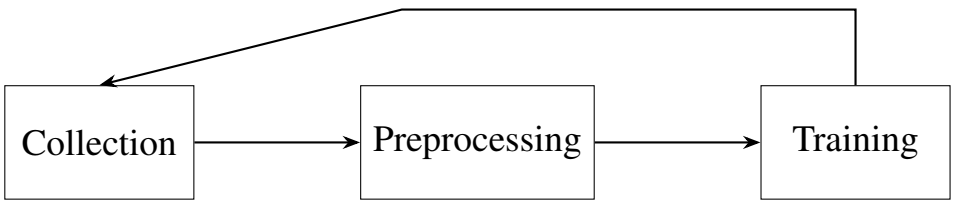
Bộ phần thưởng được đo lường giữa trạng thái hiện tại của agent và trạng thái mục tiêu được thiết kế trong công thức tính phần thưởng theo S_t. Bộ phần thưởng được tính toán theo S_t và đã được nêu ra ở phần 3.3.1.

3.1.7. Phần thưởng tích lũy G_t và ưu thế A_t

Phần thưởng tích lũy G_t và ưu thế A_t sẽ được tính theo phần thưởng R_t nên sẽ được đề cập trong phần 3.3.1.

3.2. Thiết kế mô hình huấn luyện Cassie trong python

Dựa theo tài liệu số [2], sơ đồ "tổng quát" được thiết kế lại như sau:



Sẽ có 3 giai đoạn:

- Giai đoạn 1 - Collection: Thu thập thông tin từ agent, điều khiển agent

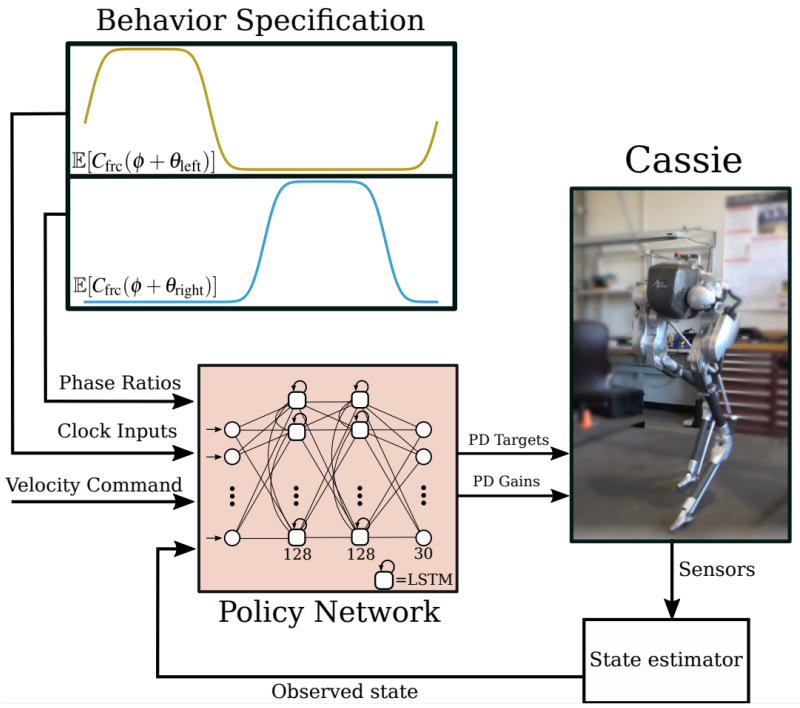
thực hiện hành động tại trạng thái S_t thực hiện hành động a_t và tính toán các thông số cần thiết. Quá trình này kết thúc khi thu thập đủ lượng sample yêu cầu.

- Giai đoạn 2 - Preprocessing: Từ dữ liệu thu thập được, phân ra thành cách batch nhỏ (mini-batch) có kích thước bằng nhau (batch cuối có thể khác). Sau đó chuyển dữ liệu đã phân thành các minibatch lên tensor và kết thúc.
- Giai đoạn 3 - Training: Sau khi đã phân thành các mini-batch, lần lượt các mini-batch sẽ được huấn luyện. Quá trình này kết thúc khi huấn luyện xong k epoch. Sau đó quá trình 1 được diễn ra. Và chương trình kết thúc tại giai đoạn này khi thu thập và huấn luyện đủ số sample mong muốn.

3.2.1. Quá trình thu thập trải nghiệm - Giai đoạn 1

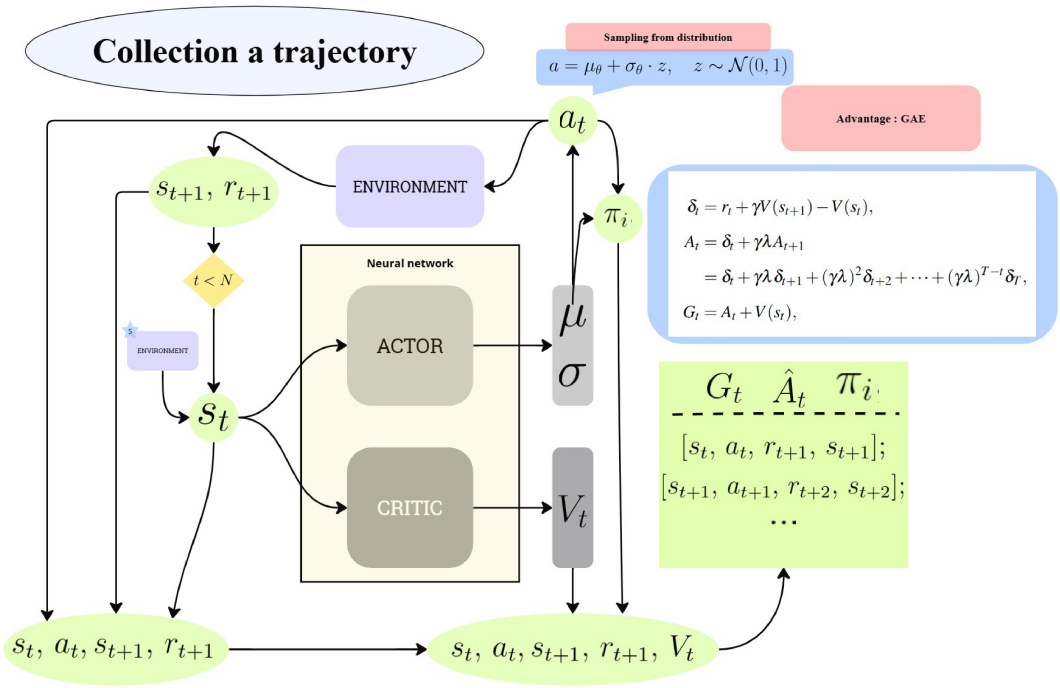
Việc thu thập trải nghiệm được lấy ý tưởng từ liệu số [2].

Mô hình tham khảo:



Hình 3.2: Điều khiển robot với policy của mạng Actor và nguyên lý điều khiển PD (Proportional-derivative)

Mô hình sử dụng trong dự án:



Hình 3.3: Quá trình thu thập trải nghiệm

Giải thích:

Sơ đồ mô tả cách dữ liệu được thu thập trong quá trình này, với khối "ENVIRONMENT" sẽ là nơi sử dụng thư viện mujoco trong python đọc dữ liệu mô phỏng của agent (Cassie robot), ra lệnh cho robot, quá trình thu thập bắt đầu ở khối hình chữ nhật nằm ngang bên trái có mũ tên chỉ tới S_t. Sau đó mô hình Actor-Critic có nhiệm vụ tạo đầu ra. Đầu ra của Actor là các biến μ và σ trong phân phối chuẩn để sinh ra hành động a_t. Khi có a_t, chương trình sẽ tính ra thông tin điều khiển cho robot (là các momen xoắn tại các actuator - bộ cơ cấu chấp hành) (giống như hình 3.2). Khi nhận được tín hiệu điều khiển, trình mô phỏng Mujoco trong python sẽ mô phỏng quá trình điều khiển dựa vào tín hiệu điều khiển được tính toán từ TD (gồm dTarget, dGain, pGain chính là a_t). Sau khi mô phỏng với tần số 2000HZ và thực hiện 50 bước mô phỏng, chương trình sẽ thu thập trạng thái S_{t+1} và tính toán $R_(t+1)$. Và từ đây cũng tính được log_policy, V_t từ mạng Critic. Sau khi đã có được S_t, a_t, $R_(t+1)$, V_t, log_policy ta sẽ lưu lại để sử dụng cho việc tính toán về sau. Dữ liệu này được gọi là sample. Chú ý, trong hình có hiển thị là thu thập cả S_{t+1}, nhưng S_{t+1} chỉ

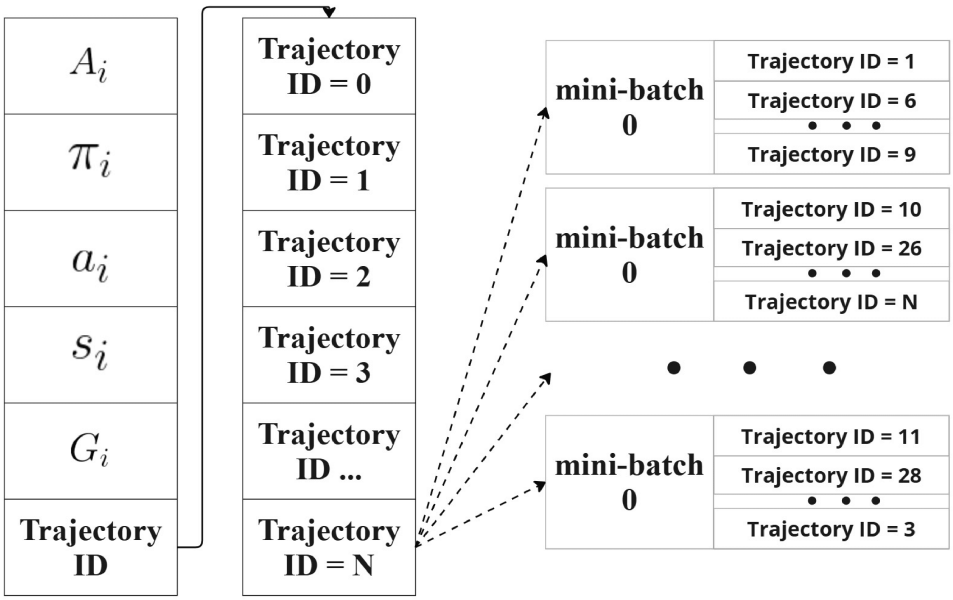
là dữ liệu được lấy ra từ agent từ lần thu thập 1 sample tiếp theo và không được lưu trữ lại. Cách mô tả đó chỉ có chức năng hiển thị, đó là một vòng lặp.

Và cứ tiếp tục thu thập đến khi nào gặp trạng thái dừng (Robot ngã) hoặc đạt được số lượng tối đa của một trajectory là N sample thì dừng. Sau khi thu thập đủ 1 trajectory thì sẽ tính được G_t và A_t cho từng sample dựa vào cả trajectory đó. Vì G_t là phần thưởng tích lũy nên, theo công thức, thì phải biết R của sample tiếp theo cho đến sample cuối cùng của 1 trajectory. Khi đó mới tính được G_t và A_t.

Và giai đoạn này kết thúc khi thu thập đủ số lượng sample yêu cầu.

3.2.2. Quá trình tiền xử lý dữ liệu - Giai đoạn 2

Mô hình sử dụng:



Hình 3.4: Quá trình tiền xử lý dữ liệu

Giải thích: Sau khi đã thu thập đủ số lượng trajectory, thì chương trình thực hiện phân chia batch cho mô hình Actor và Critic huấn luyện.

Vì lưu trữ của quá trình thu thập trải nghiệm được lưu trong một dictionary, nên dữ liệu của các key ("state", "action", "value",...) sẽ được sắp xếp theo id của trajectory.

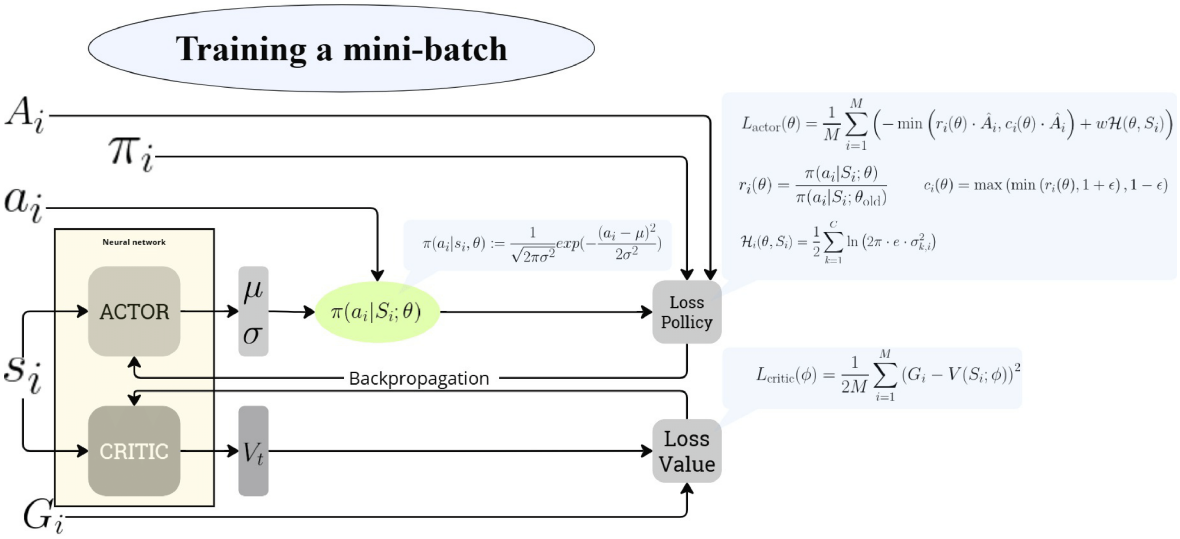
Ví dụ:

```
self.buffer = {
    "states": [
        [1.0, 0.5, -0.3], [0.9, 0.4, -0.2],
        [0.8, 0.3, -0.1], [0.7, 0.2, 0.0],
        [0.6, 0.1, 0.1], [0.5, 0.0, 0.2],
        [0.4, -0.1, 0.3], [0.3, -0.2, 0.4]
    ],
    "actions": [0, 1, 0, 1, 0, 1, 0, 1],
    "rewards": [1.0, 0.8, 0.6, 0.4, 0.2, 0.0, -0.2,
      -0.4],
    "log_probs": [
        -0.693, -0.510, -0.405, -0.300,
        -0.200, -0.100, -0.050, -0.020
    ],
    "values": [0.8, 0.9, 0.85, 0.7, 0.6, 0.5, 0.4,
      0.3],
    "trajectory_ids": [1, 1, 1, 2, 3, 3, 4, 5],
}
```

Nên lúc này, việc của quá trình tiền xử lý dữ liệu sẽ là, gộp dữ liệu theo trajectory tương ứng, trajectoryID nào thì sẽ ở trong dict của trajectoryID đó. Sau khi đã phân ra thành cách trajectory, thì bước tiếp theo là chia các trajectory thành các batch. Ví dụ nếu size batch là 32 thì sẽ có tối đa 32 trajectory nằm trong 1 mini-batch. Và mini-batch cuối cùng có thể có số lượng trajectory ít hơn. Trong giai đoạn này có một bước quan trọng là tạo padding cho từng mini-batch. Việc tạo padding này giúp sinh ra matrix không bị lỗi, vì maxtrix mới làm việc được trong Pytorch. Lý do cần padding là vì số lượng sample trong 1 trajectory giữa các trajectory không đồng đều, nên không thể ghép lại thành ma trận được. Chính vì thế, cần phải xử lý padding để tất cả các trajectory có cùng số lượng sample (hay chiều thứ 2 của ma trận 3 chiều).

3.2.3. Quá trình huấn luyện - Giai đoạn 3

Mô hình sử dụng:



Hình 3.5: Quá trình training

Giải thích:

Khi đã có dữ liệu của mini-batch, quá trình này sẽ lấy thông tin bên trong của từng mini-batch, là giá trị cần có để đưa vào công thức tính toán. Mini-batch sẽ cấu trúc dữ liệu dưới dạng dictionnary, key là biến tương ứng trong công thức, value là dữ liệu có kiểu là tensor.

Đối với quá trình này, tác giả sẽ đẩy dữ liệu lên GPU để việc tính toán diễn ra nhanh hơn. Vì tensor trong Pytorch tối ưu trên GPU hơn là CPU.

3.3. Các công thức toán học có trong mô hình

3.3.1. Quá trình thu thập trải nhiệm

1. Dữ liệu đầu vào Actor và Critic:

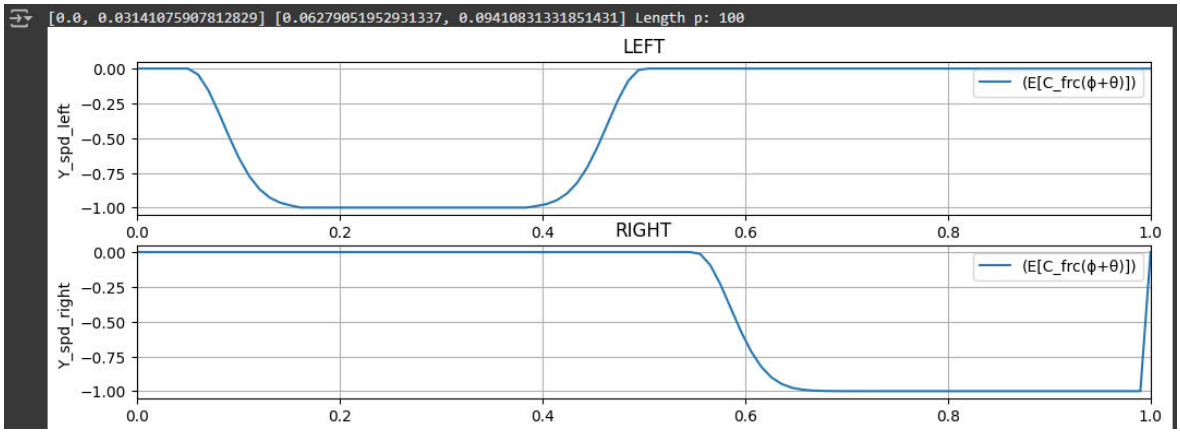
$$X_t = \begin{cases} \hat{q}, \dot{\hat{q}} & \text{trạng thái của robot} \\ \dot{x}_{\text{desiredd}}, \dot{y}_{\text{desired}} & \text{vận tốc mục tiêu} \\ r, p & \text{tỷ lệ pha và đầu vào xung nhịp} \end{cases}$$

2. Đầu vào xung nhịp:

$$p = \left\{ \sin\left(\frac{2\pi(\phi+\theta_{\text{left}})}{L}\right), \sin\left(\frac{2\pi(\phi+\theta_{\text{right}})}{L}\right) \right.$$

3. Xác Suất Pha:

Đây là một hàm xác suất sử dụng cho việc tạo ra một xung tín hiệu có vai trò trong việc tính phần thưởng R_{t+1} tại trạng thái S_{t+1} khi thực hiện hành động a_t ở trạng thái S_t. Sau đây là hình ảnh trực quan của xung tín hiệu được tác giả vẽ bằng Matplotlib Pyplot trong Google Colab, và có đồ thị như hình sau:



Hình 3.6: Xung tín hiệu được tạo ra từ hàm mũ

Xung tín hiệu này có tác dụng thay thế cho dữ liệu bước thực thế. Thay vì phải gắn những cảm biến trên người để thu thập dữ liệu bước ở từng thời điểm giống như cách làm truyền thống, thì xung này sẽ tạo ra một biết đồ dạng sóng với trục tung ($\mathbb{E}[C_{\text{frc}}(\phi+\theta)]$) là giá trị lực chân tiếp xúc với mặt đất được chuẩn hóa trong khoảng (-1 đến 0), trục hoành (ϕ) là thời điểm trong 1 chu kì bước. Khi giá trị của hàm xung này = 0, thì đó là lúc hạ chân xuống và = -1 là lúc nhấc chân lên. Robot sẽ dựa vào sung này để học theo giá trị lực trong từng thời điểm của xung. Khi nhân giá trị của xung này với lực thực tế sẽ chính là phần thưởng phạt. Ví dụ, nếu như ở thời điểm t, giá trị của xung là -1 (pha nhấc chân) mà giá trị của lực tại chân đó >0, ví dụ F = 50N, hay có lực tại chân thì sẽ bị phạt -1*50 . Hay R sẽ bị giảm 50 đơn vị. Tương tự nếu như ko có lực tại

thời điểm nhấc chân lên thì sẽ ko bị phạt (R(phạt) = -1*0 = 0).

4. Phần thưởng tổng:

$$\begin{aligned}\mathbb{E}[R_{\text{multi}}(s,\phi)] = {} & 0.400 \cdot \mathbb{E}[R_{\text{bipedal}}(s,\phi)] \\ & + 0.300 \cdot \mathbb{E}[R_{\text{cmd}}(s)] \\ & + 0.100 \cdot \mathbb{E}[R_{\text{smooth}}(s)] \\ & + 0.100 \cdot (\omega - 1) \cdot q_{\text{standing cost}}(s) \\ & + 1.\end{aligned} \tag{3.3.1}$$

5. Công thức cho $R_{\text{bipedal}}(s,\phi)$

$$\begin{aligned}\mathbb{E}[R_{\text{bipedal}}(s,\phi)] = {} & \mathbb{E}[C_{\text{frc}}(\phi + \theta_{\text{left}})] \cdot q_{\text{left frc}}(s) \\ & + \mathbb{E}[C_{\text{frc}}(\phi + \theta_{\text{right}})] \cdot q_{\text{right frc}}(s) \\ & + \mathbb{E}[C_{\text{spd}}(\phi + \theta_{\text{left}})] \cdot q_{\text{left spd}}(s) \\ & + \mathbb{E}[C_{\text{spd}}(\phi + \theta_{\text{right}})] \cdot q_{\text{right spd}}(s)\end{aligned} \tag{3.3.2}$$

6. Công thức cho $R_{\text{cmd}}(s)$ và $R_{\text{smooth}}(s)$:

$$\begin{aligned}R_{\text{cmd}}(s) = {} & (-1) \cdot q_{\dot{x}}(s) \\ & + (-1) \cdot q_{\dot{y}}(s) \\ & + (-1) \cdot q_{\text{orientation}}(s)\end{aligned}$$

$$\begin{aligned}R_{\text{smooth}}(s) = {} & (-1) \cdot q_{\text{action diff}}(s) \\ & + (-1) \cdot q_{\text{torque}}(s) \\ & + (-1) \cdot q_{\text{pelvis acc}}(s)\end{aligned}$$

7. Công thức cho ω:

$$\omega = \left(1 + \exp\left(-50 \cdot (r_{\text{swing}} - 0.15)\right)\right)^{-1}$$

8. Công thức cho các thành phần của phần thưởng:

Quantity q_i	**Detail**
$q_{\text{left/right frc}}$	$1-\exp\left(-\omega\|\text{raw\_foot\_frc}\|_2^2/100\right)$
$q_{\text{left/right spd}}$	$1-\exp\left(-2\cdot\omega\|\text{raw\_foot\_spd}\|_2^2\right)$
$q_{\dot{x}}$	$1-\exp\left(-2\cdot\omega\lvert\dot{x}_{\text{desired}}-\dot{x}_{\text{actual}}\rvert\right)$
$q_{\dot{y}}$	$1-\exp\left(-2\cdot\omega\lvert\dot{y}_{\text{desired}}-\dot{y}_{\text{actual}}\rvert\right)$
$q_{\text{orientation}}$	$1-\exp\left(-3\cdot\left(1-\left((\text{quat}_{\text{actual}})^T\cdot\text{quat}_{\text{des}}\right)^2\right)\right)$
$q_{\text{action diff}}$	$1-\exp\left(-5\cdot\|a_t-a_{t-1}\|\right)$
q_{torque}	$1-\exp\left(-0.05\cdot\|t\|\right)$
$q_{\text{pelvis acc}}$	$1-\exp\left(-0.10\cdot(\|\text{pelvis}_{\text{rot}}\|+\|\text{pelvis}_{\text{acc}}\|)\right)$

Bảng 3.2: Công thức chi tiết cho các thành phần bên trong công thức tính phần thưởng

9. Công thức Tính Returns và Advantages:

$$
\begin{aligned}
\delta_t &= r_t + \gamma V(s_{t+1}) - V(s_t),\\
A_t &= \delta_t + \gamma\lambda A_{t+1}\\
&= \delta_t + \gamma\lambda\,\delta_{t+1} + (\gamma\lambda)^2\delta_{t+2} + \cdots + (\gamma\lambda)^{T-t}\delta_T,\\
G_t &= A_t + V(s_t),
\end{aligned}
$$

Trong đó

- t: Thời điểm hiện tại trong trajectory.
- T: Tổng số bước trong trajectory.
- r_t: Phần thưởng nhận được tại thời điểm t.
- $V(s_t)$: Giá trị trạng thái s_t dựa trên critic.
- $V(s_{t+1})$: Giá trị trạng thái s_{t+1} dựa trên critic.
- γ: Hệ số chiết khấu (discount factor).
- λ: Hệ số GAE (Generalized Advantage Estimation).
- δ_t: Sai lệch (temporal difference error) giữa phần thưởng nhận được và giá trị dự đoán.
- A_t: Advantage (lợi thế) tại thời điểm t.

- G_t: Return (tổng phần thưởng tích lũy) tại thời điểm t. Chú ý, G_t và Q_t là giống nhau. Và A_t = Q_t - V_t.

3.3.2. Quá trình huấn luyện

1. Công thức *L_clip* cho Ator:

$$L_{\text{actor}} = \frac{1}{N}\sum_{i=1}^{N}\frac{M_{i,t}\cdot\min\left(r_{i,t}\cdot\hat{A}_{i,t},\text{clip}\left(r_{i,t},1-\varepsilon,1+\varepsilon\right)\cdot\hat{A}_{i,t}\right)}{\sum_{t=1}^{T}M_{i,t}+10^{-8}} \tag{3.3.3}$$

Trong đó:

- $M_{i,t}$: **Mask**, chỉ định mẫu hợp lệ tại batch i và thời điểm t.
- $r_{i,t}$: **Tỷ lệ xác suất**, được tính bằng:

$$r_{i,t} = \exp\left(\sum_{a}\left(\log\pi_{\theta}(a|s)-\log\pi_{\theta_{\text{old}}}(a|s)\right)\cdot M_{i,t}\right)$$

- $\hat{A}_{i,t}$: **Lợi thế chuẩn hóa**, được tính như sau:

$$\hat{A}_{i,t} = \frac{A_{i,t}-\mu_A}{\sigma_A+10^{-8}}$$

- μ_A: **Trung bình lợi thế** trên batch.
- σ_A: **Độ lệch chuẩn lợi thế** trên batch.
- ε: **Hệ số clip**, thường là 0.2.
- λ: **Hệ số entropy**, điều chỉnh mức độ đóng góp của entropy vào mất mát tổng.
- N: **Số lượng mẫu** trong batch.
- T: **Chiều dài của mỗi trajectory**.

3. Công thức hàm loss cho Critic:

$$L_{\text{critic}} = \frac{1}{N}\sum_{i=1}^{N}\frac{M_{i,t}\cdot(V_{\theta}(s_{i,t})-\hat{R}_{i,t})^2}{\sum_{t=1}^{T}M_{i,t}+10^{-8}} \tag{3.3.4}$$

Trong đó:

- $M_{i,t}$: **Mask**, chỉ định mẫu hợp lệ tại batch i và thời điểm t.
- $V_\theta(s_{i,t})$: **Giá trị dự đoán** cho trạng thái $s_{i,t}$.
- $\hat{R}_{i,t}$: **Phần thưởng chuẩn hóa**, được tính như sau:

$$\hat{R}_{i,t} = \frac{R_{i,t} - \mu_R}{\sigma_R + 10^{-8}}$$

- μ_R: **Trung bình phần thưởng** trên batch.
- σ_R: **Độ lệch chuẩn phần thưởng** trên batch.
- N: **Số lượng mẫu** trong batch.
- T: **Chiều dài của mỗi trajectory**.

CHƯƠNG 4: KẾT QUẢ THỰC NGHIỆM

4.1. Cấu hình máy và tham số chương trình

4.1.1. Cấu hình

Agent được huấn luyện trên máy tính có cấu hình như sau:

1. CPU: Intel(R) Core(TM) i3-10105F CPU @ 3.70GHz
2. GPU: NIDIA GeForce RTX 2060
3. RAM: 16GB
4. Hệ điều hành: Microsoft Windows 11
5. The IDE: PyCharm

4.1.2. Tham số chương trình

1. Tổng sample: 20,000,000(sample)
2. Số lượng sample tối đa trong 1 trajectory: 300(sample)
3. Số bước thời gian cho 1 chu kì bước: 28(step or sample)
4. Pha ban đầu của tần số bước: 0.1
5. Tần số chính sách: 40Hz
6. Tần số bộ điều khiển PD: 2000Hz
7. Số luồng thu thập: 4(luồng)
8. Số lượng sample cần thu thập cho mỗi lần huấn luyện: 50,000(sample)
9. Vận tốc mục tiêu: $\dot{x} : 0.96(m/s), \dot{y} : 0(m/s)$
10. Hướng mục tiêu: $quat_{des} : [1,0,0,0]$
11. Độ dài pha nhấc chân: $r = 0.6$
12. Số epoch cho 1 lần huấn luyện: 4(epoch)
13. Tốc độ học(lr) cho mạng Actor: 0.0001
14. Tốc độ học(lr) cho mạng Critic: 0.0001
15. Ngưỡng Clip (ε) cho thuật toán PPOClip: 0.2
16. Giá trị tối thiểu của sigma trong quá trình WarmUp: 0.8
17. Giá trị tối thiểu của sigma sau quá trình WarmUp: 0.4

18. Thời điểm dừng quá trình WarmUp: Thu thập đủ 0.9% số lượng sample cần thu thập

4.1.3. Lý do lựa chọn tham số mô hình

1. Tổng sample: 20,000,000(sample), lý do chọn 20,000,000 sample để huấn luyện thay vì 150,000,000 sample như trong tài liệu tham khảo [2]: Do cấu hình máy tính khi để công suất tối đa để hoàn thành chương trình với 150,000,000 mẫu là khoảng 10 ngày. Vậy nên rất khó để có thể hoàn thành. Vậy nên tác giả chọn 20,000,000 mẫu và sẽ mất 1 ngày để huấn luyện, và chắc chắn agent sẽ không học được việc đi bộ như mục tiêu đã yêu cầu. Chính vì thế việc thu thập này sẽ có mục tiêu quan sát xem liệu agent có học tốt lên theo thời gian hay không, nếu quá trình này cho ra kết quả tốt thì tác giả sẽ có hướng đi sau khi hoàn thành đồ án tốt nghiệp này.
2. Số lượng sample tối đa trong 1 trajectory: 300(sample). Lý do chọn: dựa theo hướng dẫn trong tài liệu tham khảo [2].
3. Số bước thời gian cho 1 chu kì bước: 28(step or sample). Lý do chọn: số bươc này được tính dựa theo khoảng cách bước của chân và tốc độ di chuyển mục tiêu.
4. Pha ban đầu của tần số bước: 0.1. Lý do chọn: mong muốn pha bước được thực hiện sớm ở chân phải.
5. Tần số chính sách: 40Hz. Lý do chọn: dựa theo hướng dẫn trong tài liệu tham khảo [2].
6. Tần số bộ điều khiển PD: 2000Hz. Lý do chọn: dựa theo hướng dẫn trong tài liệu tham khảo [2].
7. Số luồng thu thập: 4(luồng). Lý do chọn: vì công suất tối đa mà chương trình có thể thu thập trải nghiệm là lúc phân quá trình thu thập trải nghiệm thành 4 luồng riêng biệt.
8. Số lượng sample cần thu thập cho mỗi lần huấn luyện: 50,000(sample). Lý do chọn: dựa theo hướng dẫn trong tài liệu tham khảo [2].

9. Vận tốc mục tiêu: $\dot{x} : 0.96(m/s), \dot{y} : 0(m/s)$. Lý do chọn: là vận tốc nằm trong khoảng đi bộ.
10. Hướng mục tiêu: $quat_{des} : [1, 0, 0, 0]$. Lý do chọn: mong muốn robot đi với dáng đi thẳng, không bị nghiêng khi di chuyển.
11. Độ dài pha nhấc chân: $r = 0.6$. Lý do chọn: dựa theo hướng dẫn trong tài liệu tham khảo [2]. r càng lớn thì thời gian nhấc chân lên so với thời gian chân chạm đất sẽ càng lớn. r tăng sẽ khiến cho việc đi bộ chuyển sang đang nhảy lò cò. Và ngược lại, nếu như r giảm thì sẽ khiến cho việc đi bộ chuyển sang đứng yên tại chỗ.
12. Số epoch cho 1 lần huấn luyện: 4(epoch). Lý do chọn: dựa theo hướng dẫn trong tài liệu tham khảo [2].
13. Tốc độ học(lr) cho mạng Actor: 0.0001. Lý do chọn: dựa theo hướng dẫn trong tài liệu tham khảo [2].
14. Tốc độ học(lr) cho mạng Critic: 0.0001. Lý do chọn: dựa theo hướng dẫn trong tài liệu tham khảo [2].
15. Ngưỡng Clip (ε) cho thuật toán PPOClip: 0.2. Lý do chọn: dựa theo kết quả của bài báo số [7].
16. Giá trị tối thiểu của σ trong quá trình WarmUp: 0.8. Lý do chọn: vì giá trị min-max của vị trí khớp là (-2.86,1.40) nên khi chọn $\sigma = 0.8$ thì $\sigma^2 = 0.64$, tạo ra khoảng rộng đủ để agent có thể tăng khả năng khám phá.
17. Giá trị tối thiểu của sigma sau quá trình WarmUp: 0.4. Lý do chọn: tương tự như việc chọn σ trong quá trình WarmUp, σ tạo ra khoảng rộng không quá nhỏ khiến agent quá tập trung vào hành động đã học được.
18. Thời điểm dừng quá trình WarmUp: Thu thập đủ 0.9% số lượng sample cần thu thập. Lý do chọn: vì lượng sample thu thập là khoảng 13% so với đề xuất trong tài liệu tham khảo [2] nên khá hợp lý khi chọn 0.9% để agent có thể khám phá được nhiều hơn trong giai đoạn đầu.

4.2. Phân tích kết quả

Trong quá trình thu thập và huấn luyện, chương trình sẽ lưu thông tin sau mỗi một epoch, hay sẽ có 4 giá trị được lưu khi huấn luyện 50,000 sample, các giá trị được lưu đó là:

- Phần thưởng trung bình của 50,000 sample.
- Giá trị hàm Loss Actor trung bình của toàn bộ các mini-batch qua mỗi lần train trong 1 epoch.
- Giá trị hàm Loss Critic trung bình của toàn bộ các mini-batch qua mỗi lần train trong 1 epoch.
- Entropy trung bình của toàn bộ các mini-batch qua mỗi lần train trong 1 epoch.

Công thức tính entropy

$$H = \frac{1}{2}\sum_{j=1}^{T}\left(1+\log(2\pi\sigma_j^2)\right), \quad H \in \mathbb{R}^{B\times T\times 1}$$

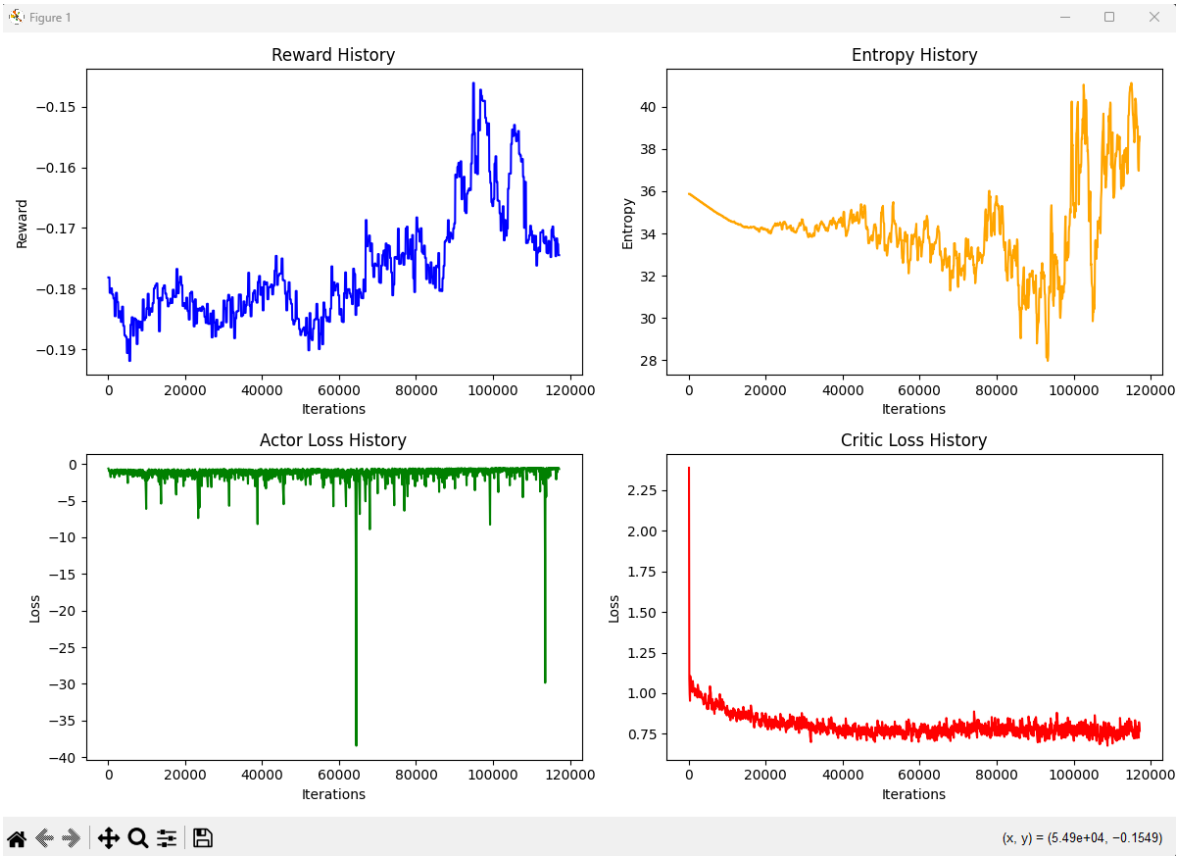
Do công thức tính entropy liên quan trực tiếp đến bình phương của σ, khi σ tăng, entropy cũng tăng theo. Entropy được sử dụng để đo lường mức độ không chắc chắn hoặc lượng thông tin chưa biết trong một phân phối xác suất.

Nếu số lần lấy mẫu từ một phân phối rất lớn, hình dạng phân phối sẽ được xác định rõ ràng hơn. Ngược lại, nếu số lần lấy mẫu nhỏ, ví dụ chỉ 1 hoặc 2 lần, chúng ta không thể chắc chắn rằng các mẫu thu được phản ánh chính xác phân phối thực tế.

Chẳng hạn, khi chọn một mẫu x_1 từ phân phối và giả sử giá trị lý thuyết của xác suất $f(x_1) = 0.7$. Dù lý thuyết chỉ ra xác suất x_1 chiếm 70%, thực tế khi lấy mẫu 10 lần không đảm bảo rằng có đúng 7 lần chọn được x_1. Điều này là do tính ngẫu nhiên và mức độ không chắc chắn trong việc lấy mẫu.

Entropy chính là thước đo để đánh giá mức độ không chắc chắn này. Giá trị entropy càng lớn thể hiện sự phân tán của xác suất càng cao, đồng nghĩa với việc chính sách hoặc mô hình chưa hội tụ rõ ràng.

Khi entropy dao động, điều này cho thấy mức độ không chắc chắn hoặc phạm vi thăm dò trong phân phối thay đổi. Nếu entropy ổn định và giảm dần theo thời gian, đó thường là dấu hiệu tốt cho việc mạng hoặc chính sách đang hội tụ. Ngược lại, nếu entropy dao động bất thường, có thể cần điều chỉnh lại các thông số hoặc chiến lược huấn luyện.



Hình 4.7: Kết quả sau khi huấn luyện

4.2.1. Diễn giải từ các thông số huấn luyện và ý nghĩa:

- **Số lượng mẫu (samples):**
  - Mục tiêu ban đầu là huấn luyện với $150,000,000$ samples, nhưng dữ liệu hiện tại chỉ sử dụng $20,000,000$ samples ($\sim 13.33\%$ của tổng số).
  - Điều này cho thấy mô hình vẫn chưa đạt mức tối ưu hóa hoàn toàn.
- **Số lượng trajectory:**
  - Ban đầu số lương trajectory thu thập đươc là khoảng 85×32 trajectory, tức các episode của robot thường kết thúc sớm (do

robot mất thăng bằng hoặc thất bại trong nhiệm vụ).

  - Sau khi train đến $20,000,000$ samples, số trajectory giảm xuống 65×32 trajectory, tức robot di chuyển lâu hơn và ổn định hơn trước khi gặp trạng thái kết thúc (fail state).
  - Điều này chứng tỏ robot đã học được chính sách tốt hơn, cải thiện khả năng cân bằng và di chuyển.

- **Không gian trạng thái lớn:**
  - Với robot Cassie, không gian trạng thái bao gồm hàng trăm giá trị từ IMU, góc khớp, tốc độ, lực, và các tín hiệu cảm biến khác. Điều này đòi hỏi chính sách tối ưu để giải quyết bài toán phức tạp này. Vì thế với số lượng sample ít sẽ không thể tối ưu bải toán này.

4.2.2. Phân tích từng biểu đồ và liên hệ với thông tin bổ sung:

- **Entropy History (Biểu đồ màu vàng):**
  - **Ý nghĩa:** Entropy đo lường mức độ không chắc chắn của chính sách (policy).
  - **Nhận xét:**
    - Ban đầu, entropy cao (~ 40), cho thấy chính sách còn ngẫu nhiên, robot đang thử nghiệm nhiều hành động khác nhau.
    - Khi huấn luyện tiến triển (khoảng từ $20,000$ đến $80,000$ iterations), entropy giảm dần, chính sách trở nên tập trung hơn vào các hành động hiệu quả.
    - Sau $100,000$ iterations, entropy dao động mạnh, có thể do việc duy trì exploration để tránh stuck trong các giải pháp cục bộ hoặc do mô hình cần thêm dữ liệu (samples) để hội tụ tốt hơn. *"Điều này được lý giải vì do có ít thông tin về môi trường nên robot chưa học được trạng thái tốt mà hầu như là học trạng thái xấu. Dẫn đến, robot sẽ tập trung giảm xác suất hành động mang đến lợi thế A_t*

âm. Vì thế entropy sẽ cao lên vì muốn giảm xác suất của hành động đó thì σ phải tăng lên."

- **Reward History (Biểu đồ màu xanh):**
  - **Ý nghĩa:** Reward biểu thị hiệu suất chính sách. Reward cao đồng nghĩa với robot thực hiện nhiệm vụ hiệu quả hơn.
  - **Nhận xét:**
    * Ban đầu, reward thấp (~ -0.18) và dao động, chứng tỏ chính sách chưa tốt, robot thường xuyên thất bại trong nhiệm vụ.
    * Reward dần tăng (khoảng từ $20,000$ đến $80,000$ iterations), phản ánh robot cải thiện khả năng di chuyển và hoàn thành nhiệm vụ.
    * Sau $100,000$ iterations, reward giảm nhẹ và dao động mạnh, có thể do entropy tăng hoặc sự mất cân bằng trong quá trình tối ưu hóa Actor-Critic.
- **Actor Loss History (Biểu đồ màu xanh lá):**
  - **Ý nghĩa:** Actor Loss đo mức độ tốt hay xấu của chính sách dựa vào lợi thế A_t. Loss càng giảm chứng tỏ Critic đã ước lượng tốt nhưng nhìn vào hình thì hầu như chỉ toàn giá trị âm, điều đó chứng tỏ mô hình hầu như chỉ học cách giảm xác xuất lựa chọn hành động mang đến lợi ích tiêu cực.
  - **Nhận xét:**
    * Ban đầu, loss dao động lớn (do entropy cao), nhưng giảm dần trong quá trình huấn luyện là do Critic đã giảm.
- **Critic Loss History (Biểu đồ màu đỏ):**
  - **Ý nghĩa:** Critic Loss đo mức độ chính xác của Critic khi ước lượng giá trị trạng thái.
  - **Nhận xét:**
    * Critic Loss giảm nhanh và ổn định (~ 0.75) từ $20,000$

iterations trở đi, cho thấy Critic đã học cách đánh giá trạng thái chính xác hơn.

* Loss duy trì ổn định ở mức thấp (dưới 1.0) trong giai đoạn cuối, phản ánh Critic đã hội tụ tốt.

4.2.3. Liên hệ giữa các biểu đồ và thông tin bổ sung:

- **Entropy và Reward:**
  - Khi entropy giảm (giai đoạn giữa), reward tăng, chứng tỏ chính sách trở nên hiệu quả hơn.
  - Sự dao động mạnh của entropy ở giai đoạn cuối gây ảnh hưởng tiêu cực đến reward, dẫn đến giảm hiệu suất và làm agent khám phá nhiều hơn và gặp được nhiều trạng thái không tốt khiến cho reward giảm.
- **Reward và Trajectory:**
  - Reward tăng đồng thời số trajectory giảm, chứng minh robot đã di chuyển được lâu hơn trong mỗi episode mà không gặp phải trạng thái kết thúc sớm.
- **Critic Loss và Actor Loss:**
  - Critic Loss ổn định ở mức thấp giúp hỗ trợ Actor Loss giảm, cho phép Actor cập nhật chính sách hiệu quả hơn.

4.2.4. Kết luận:

- **Tiến bộ rõ ràng:**
  - Số trajectory giảm từ 85×32 xuống 65×32.
  - Reward tăng dần ở giai đoạn giữa.
  - Critic Loss ổn định và thấp, hỗ trợ Actor hiệu quả.
- **Vấn đề ở giai đoạn cuối:**
  - Entropy dao động mạnh làm giảm reward và gây bất ổn cho Actor Loss.
- **Đề xuất:**
  - Tiếp tục train đến số lượng mẫu lớn hơn ($\sim 150,000,000$ samples).

CHƯƠNG 5: KẾT LUẬN

5.1. Đề xuất cải tiến

1. **Tiến bộ trong quá trình huấn luyện:**
    - Robot Cassie đã đạt được tiến bộ rõ rệt trong việc cải thiện chính sách:
        - Số lượng trajectory giảm từ 85×32 xuống 65×32, chứng tỏ robot di chuyển lâu hơn và ổn định hơn trước khi gặp trạng thái kết thúc (fail state).
        - Reward tăng dần trong giai đoạn giữa, phản ánh chính sách đã trở nên hiệu quả hơn.
        - Critic Loss ổn định ở mức thấp (< 1.0), hỗ trợ Actor cải thiện chính sách.
2. **Vấn đề tồn tại ở giai đoạn cuối:**
    - Entropy dao động mạnh ở giai đoạn cuối làm giảm reward, gây bất ổn cho Actor Loss.
    - Điều này có thể là hậu quả của việc exploration quá mức hoặc mô hình cần thêm dữ liệu (samples) để đạt mức hội tụ.
3. **Không gian trạng thái phức tạp:**
    - Với robot Cassie, không gian trạng thái lớn (bao gồm IMU, góc khớp, tốc độ, lực, và các tín hiệu cảm biến khác) khiến việc hội tụ hoàn toàn đòi hỏi lượng dữ liệu huấn luyện lớn hơn so với mức hiện tại ($20,000,000$ samples chỉ đạt khoảng 13.33% so với mục tiêu $150,000,000$).

5.2. Đề xuất cải tiến

1. **Tăng số lượng mẫu huấn luyện (samples):**
    - Tiếp tục huấn luyện đến $150,000,000$ samples để đạt mức hội tụ hoàn toàn, tận dụng tối đa không gian trạng thái phức tạp.

- Theo dõi các chỉ số như *entropy* hoặc *rewards* để đảm bảo quá trình tối ưu hóa diễn ra ổn định.

2. **Điều chỉnh hyperparameters:**
    - **Giảm hệ số entropy (entropy coefficient):** Giảm giá trị này ở giai đoạn cuối để hạn chế dao động mạnh của entropy, giúp ổn định chính sách.
    - **Giảm learning rate:** Ở giai đoạn cuối, giảm learning rate để tránh các bước cập nhật lớn gây bất ổn cho Actor-Critic.
3. **Tăng cường chiến lược exploration:**
    - Sử dụng các kỹ thuật như *curriculum learning* hoặc *reward shaping* để hướng dẫn robot exploration hiệu quả hơn ở giai đoạn đầu, từ đó tăng tốc độ hội tụ.
    - Áp dụng *adaptive exploration* để giảm exploration một cách thông minh khi mô hình tiến gần đến hội tụ.
4. **Phân tích thêm dữ liệu từ mô hình:**
    - Đánh giá sâu hơn mối quan hệ giữa entropy và reward ở giai đoạn cuối để tìm ra nguyên nhân cụ thể của sự dao động.
    - Theo dõi các giá trị trung gian của mạng Actor và Critic để phát hiện các vấn đề tiềm ẩn trong quá trình cập nhật chính sách.
5. **Tối ưu hóa không gian trạng thái:**
    - Kiểm tra lại không gian trạng thái để loại bỏ các tín hiệu cảm biến không cần thiết hoặc ít ảnh hưởng, nhằm giảm độ phức tạp.

TÀI LIỆU THAM KHẢO

[1] Fangzhou Yu and Ryan Batke and Jeremy Dao and Jonathan Hurst and Kevin Green and Alan Fern., "Dynamic Bipedal Maneuvers through Sim-to-Real Reinforcement Learning" Computer Science > Robotics, 2022. arXiv: 2207.07835.

[2] Jonah Siekmann, Yesh Godse, Alan Fern, Jonathan Hurst., "Sim-to-Real Learning of All Common Bipedal Gaits via Periodic Reward Composition" Computer Science > Robotics, 2021. arXiv: 2011.01387.

[3] Zhaoming Xie, Patrick Clary, Jeremy Dao, Pedro Morais, Jonathan Hurst, Michiel van de Panne., "Iterative Reinforcement Learning Based Design of Dynamic Locomotion Skills for Cassie" Computer Science > Robotics, 2019. arXiv: 1903.09537.

[4] Haiming Mou ,Jie Xue ,Jian Liu ,Zhen Feng ,Qingdu Li andJianwei Zhang., "A Multi-Agent Reinforcement Learning Method for Omnidirectional Walking of Bipedal Robots" Computer Science > Robotics, 16 December 2023. DOI: https://doi.org/10.3390/biomimetics8080616.

[5] Jonah Siekmann, Srikar Valluri, Jeremy Dao, Lorenzo Bermillo, Helei Duan, Alan Fern, Jonathan Hurst., Learning Memory-Based Control for Human-Scale Bipedal Locomotion" Computer Science > Robotics, 2020. arXiv: 2006.02402.

[6] Fangzhou Yu and Ryan Batke and Jeremy Dao and Jonathan Hurst and Kevin Green and Alan Fern., "Dynamic Bipedal Maneuvers through Sim-to-Real Reinforcement Learning" Computer Science > Robotics, 2020. arXiv: 2006.02402.

[7] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov., "Proximal Policy Optimization Algorithms" Computer Science > Machine Learning, 2017. arXiv: 1707.06347.

[8] de Araujo, Paulo Ricardo Marques and Noureldin, Aboelmagd and Givigi, Sidney., Toward Land Vehicle Ego-Velocity Estimation Using Deep Learning and Automotive Radars" IEEE Transactions on Radar Systems, vol. 2, pp. 460-470, 2024. DOI: 10.1109/TRS.2024.3392439.

[9] Zhaoming Xie, Glen Berseth, Patrick Clary, Jonathan Hurst, Michiel van de Panne., "Feedback Control For Cassie With Deep Reinforcement Learning" Computer Science > Robotics, 2018. arXiv: 1803.05580.

[10] Xiulei Song, Yizhao Jin, Greg Slabaugh, Simon Lucas., "Joint action loss for proximal policy optimization" Computer Science > Machine Learning, 2022. arXiv: 2301.10919.

[11] MathWorks and MathLab., "Proximal Policy Optimization (PPO) Agent" Online. Available: proximal-policy-optimization-agents.html.

[12] OpenAI., "Proximal Policy Optimization" Online. Available: ppo.html

[13] Saeed Saeedvand., "Reinforcement Learning, (NTNU Course)" Online, 2023. Available: Course.

[14] Mujoco., "mujoco-document)" Online. Available: mujoco-overview.

[15] Agility Robotics., "Cassie-mujoco-sim)" Online, 2018. Available: osudrl/cassie-mujoco-sim.

[16] kevinzakka and copybara-github., "Agility Cassie Description (MJCF)" Online, 2021. Available: google-deepmind/mujoco_menagerie/agility_cassie.