Reinforcement Learning – Học Tăng Cường

Nguyễn Quản Bá Hồng*

Ngày 28 tháng 9 năm 2025

Tóm tắt nội dung

This text is a part of the series *Some Topics in Advanced STEM & Beyond*:
URL: https://nqbh.github.io/advanced_STEM/.
Latest version:

- *Reinforcement Learning – Học Tăng Cường.*
  PDF: URL: .pdf.
  TEX: URL: .tex.
- .
  PDF: URL: .pdf.
  TEX: URL: .tex.

Mục lục

1 Basic Reinforcement Learning

1.1 Miguel Morales. Grokking Deep Reinforcement Learning. 2020

- **Foreword.** So here's thing about RL. Difficult to learn & difficult to teach, for a number of reasons. 1st, it's quite a technical topic. There is a great deal of math & theory behind it. Conveying right amount of background without drowning in it is a challenge in & of itself.

  – Vậy thì đây là vấn đề về RL. Khó học & khó dạy, vì 1 số lý do. Thứ nhất, đây là 1 chủ đề khá chuyên môn. Có rất nhiều lý thuyết toán học & ẩn chứa trong đó. Việc truyền tải đúng lượng kiến thức nền tảng mà không bị chìm nghỉm trong đó quả là 1 thách thức.

  2nd, RL encourages a conceptual error. RL is both a way of thinking about decision-making problems & a set of tools for solving those problem. By "a way of thinking", i.e., RL provides a framework for making decisions: it discusses states & reinforcement signals, among other details. When say "a set of tools", mean: when discuss RL, find ourselves using terms like *Markov decision processes & Bellman updates*. Remarkably easy to confuse way of thinking with mathematical tools we use in response to that way of thinking.

  – Thứ hai, RL khuyến khích 1 lỗi khái niệm. RL vừa là 1 cách suy nghĩ về các vấn đề ra quyết định & 1 bộ công cụ để giải quyết những vấn đề đó. "1 cách suy nghĩ", i.e., RL cung cấp 1 khuôn khổ để ra quyết định: nó thảo luận về các trạng thái & tín hiệu củng cố, cùng nhiều chi tiết khác. Khi nói "một bộ công cụ", có nghĩa là: khi thảo luận về RL, chúng ta thấy mình đang sử dụng các thuật ngữ như *Markov decision processes & Bellman updates*. Thật dễ nhầm lẫn giữa cách suy nghĩ với các công cụ toán học mà chúng ta sử dụng để đáp ứng với cách suy nghĩ đó.

  Finally, RL is implementable in a wide variety of ways. Because RL is a way of thinking, can discuss it by trying to realize framework in a very abstract way, or ground it in code, or, for that matter, in neurons. Substrate one decides to use makes these 2 difficulties even more challenging – which bring us to DRL.

*A scientist- & creative artist wannabe, a mathematics & computer science lecturer of Department of Artificial Intelligence & Data Science (AIDS), School of Technology (SOT), UMT Trường Đại học Quản lý & Công nghệ TP.HCM, Hồ Chí Minh City, Việt Nam.
E-mail: nguyenquanbahong@gmail.com & hong.nguyenquanba@umt.edu.vn. Website: https://nqbh.github.io/. GitHub: https://github.com/NQBH.

– Cuối cùng, RL có thể được triển khai theo nhiều cách khác nhau. Bởi vì RL là 1 cách tư duy, nên ta có thể thảo luận về nó bằng cách cố gắng hiện thực hóa khuôn khổ 1 cách rất trừu tượng, hoặc dựa trên mã, hoặc thậm chí là trên các tế bào thần kinh. Chất nền mà ta quyết định sử dụng khiến 2 khó khăn này càng trở nên khó khăn hơn – & điều này dẫn chúng ta đến DRL.

Focusing on DRL nicely compounds all these problems at once. There is background on RL, & background on deep neural networks. Both are separately worthy of study & have developed in completely different ways. Working out how to explain both in context of developing tools is no easy task. Also, do not forget that understanding RL requires understanding not only tools & their realization in deep networks, but also understanding way of thinking about RL; otherwise, you cannot generalize beyond examples you study directly. Again, teaching RL is hard, & there are so many ways for teaching deep RL to go wrong – which brings us to Miguel Morales & this book.

– Việc tập trung vào DRL sẽ kết hợp tốt tất cả những vấn đề này cùng 1 lúc. Có nền tảng về RL, & nền tảng về mạng nơ-ron sâu. Cả 2 đều đáng được nghiên cứu riêng & đã phát triển theo những cách hoàn toàn khác nhau. Việc tìm ra cách giải thích cả 2 trong bối cảnh phát triển công cụ không phải là 1 nhiệm vụ dễ dàng. Ngoài ra, đừng quên rằng việc hiểu RL đòi hỏi không chỉ hiểu các công cụ & việc triển khai chúng trong mạng sâu, mà còn hiểu cách tư duy về RL; nếu không, bạn không thể khái quát hóa vượt ra ngoài các ví dụ bạn đã nghiên cứu trực tiếp. 1 lần nữa, việc dạy RL rất khó, & có rất nhiều cách để dạy RL sâu trở nên sai lầm – điều này đưa chúng ta đến với Miguel Morales & cuốn sách này.

This book is very well put together. It explains in technical but clear language what ML is, what DL is, what RL is. It follows reader to understand larger context of where field is & what you can do with techniques of deep RL, but also way of thinking that ML, RL, & deep RL present. Clear & concise. Thus, it works as both a learning guide & a reference, & as a source of some inspiration.

– Cuốn sách này được biên soạn rất tốt. Nó giải thích bằng ngôn ngữ chuyên môn nhưng rõ ràng về ML, DL & RL. Cuốn sách giúp người đọc hiểu bối cảnh rộng hơn về lĩnh vực này & những gì bạn có thể làm với các kỹ thuật của RL sâu, cũng như cách tư duy mà ML, RL, & RL sâu thể hiện. Rõ ràng & súc tích. Vì vậy, nó vừa là 1 hướng dẫn học tập & 1 tài liệu tham khảo, & 1 nguồn cảm hứng.

- **Preface.** RL is an exciting field with potential to make a profound impact on history of humankind. Several technologies have influenced history of our world & changed course of humankind, from fire, to wheel, to electricity, to internet. Each technological discovery propels next discovery in a compounding way. Without electricity, personal computer wouldn't exist; without it, internet wouldn't exist; without it, search engines wouldn't exist.

  – RL là 1 lĩnh vực thú vị với tiềm năng tạo ra tác động sâu sắc đến lịch sử nhân loại. Nhiều công nghệ đã ảnh hưởng đến lịch sử thế giới & thay đổi tiến trình của nhân loại, từ lửa, bánh xe, điện, đến Internet. Mỗi khám phá công nghệ đều thúc đẩy những khám phá tiếp theo theo cách tích lũy. Không có điện, máy tính cá nhân sẽ không tồn tại; không có điện, Internet sẽ không tồn tại; không có điện, công cụ tìm kiếm sẽ không tồn tại.

  Most exciting aspect of RL & AI, in general, is not so much to merely have other intelligent entities next to us, which is pretty exciting, but instead, what comes after that. Believe RL, being a robust framework for optimizing specific tasks autonomously, has potential to change world. In addition to task automation, creation of intelligent machines may drive understanding of human intelligence to places we have never been before. Arguably, if you can know with certainty how to find optimal decisions for every problem, you likely understand algorithm that finds those optimal decisions. I have a feeling that by creating intelligent entities, humans can become more intelligent beings.

  – Khía cạnh thú vị nhất của RL & AI, nói chung, không chỉ đơn thuần là có những thực thể thông minh khác bên cạnh chúng ta, điều này khá thú vị, mà thay vào đó là những gì xảy ra sau đó. Tin rằng RL, là 1 khuôn khổ mạnh mẽ để tối ưu hóa các tác vụ cụ thể 1 cách tự động, có tiềm năng thay đổi thế giới. Bên cạnh việc tự động hóa tác vụ, việc tạo ra các máy móc thông minh có thể đưa sự hiểu biết về trí thông minh của con người đến những tầm cao mà chúng ta chưa từng đạt tới. Có thể nói, nếu bạn có thể biết chắc chắn cách tìm ra quyết định tối ưu cho mọi vấn đề, thì có lẽ bạn đã hiểu thuật toán tìm ra những quyết định tối ưu đó. Tôi có cảm giác rằng bằng cách tạo ra các thực thể thông minh, con người có thể trở nên thông minh hơn.

  But we are far away from this point, & to fulfill these wild dreams, need more minds at work. RL is not only in its infancy, but it's been in that state for a while, so there is much work ahead. Reason I wrote this book: get more people grokking deep RL, & RL in general, & to help you contribute.

  – Nhưng chúng ta còn rất xa mới đạt được điều đó, & để hiện thực hóa những giấc mơ hoang đường này, cần nhiều trí tuệ hơn nữa cùng chung tay. Đời sống thực tế không chỉ đang ở giai đoạn sơ khai, mà nó đã ở trong trạng thái đó 1 thời gian dài, vì vậy còn rất nhiều việc phải làm ở phía trước. Lý do tôi viết cuốn sách này: để giúp nhiều người hiểu sâu hơn về đời sống thực tế, & đời sống thực tế nói chung, & để giúp bạn đóng góp.

  Even though RL framework is intuitive, most of resources out there are difficult to understand for newcomers. Goal was not to write a book that provides code examples only, & most definitely not to create a resource that teaches theory of RL. Instead, goal was to create a resource that can bridge gap between theory & practice. I don't shy away from equations; they are essential if you want to grok a research field. &, even if your goal is practical, to build quality RL solutions, you still need that theoretical foundation. However, I also don't solely rely on equations because not everybody interested in RL is fond of math. Some people are more comfortable with code & concrete examples, so this book provides practical side of this fantastic field.

– Mặc dù framework RL rất trực quan, nhưng hầu hết các tài nguyên hiện có đều khó hiểu đối với người mới bắt đầu. Mục tiêu không phải là viết 1 cuốn sách chỉ cung cấp các ví dụ mã, & chắc chắn không phải là tạo ra 1 tài nguyên dạy lý thuyết RL. Thay vào đó, mục tiêu là tạo ra 1 tài nguyên có thể thu hẹp khoảng cách giữa lý thuyết & thực hành. Tôi không ngại sử dụng các phương trình; chúng rất cần thiết nếu bạn muốn hiểu sâu về 1 lĩnh vực nghiên cứu. &, ngay cả khi mục tiêu của bạn là thực tế, để xây dựng các giải pháp RL chất lượng, bạn vẫn cần nền tảng lý thuyết đó. Tuy nhiên, tôi cũng không chỉ dựa vào các phương trình vì không phải ai quan tâm đến RL cũng thích toán học. 1 số người cảm thấy thoải mái hơn với mã & các ví dụ cụ thể, vì vậy cuốn sách này cung cấp khía cạnh thực tế của lĩnh vực tuyệt vời này.

Most of my effort during this 3-year project went into bridging this gap; I don't shy away from intuitively explaining theory, & I don't just plop down code examples. I do both, & in a very detail-oriented fashion. Those who have a hard time understanding textbooks & lectures can more easily grasp words top researchers use: why those specific words, why not other words. & those who know words & love reading equations but have trouble seeing those equations in code & how they connect can more easily understand practical side of RL.

– Phần lớn nỗ lực của tôi trong suốt dự án 3 năm này là thu hẹp khoảng cách này; tôi không ngại giải thích lý thuyết 1 cách trực quan, & tôi không chỉ đưa ra các ví dụ mã. Tôi làm cả hai, & theo cách rất chi tiết. Những người gặp khó khăn trong việc hiểu sách giáo khoa & bài giảng có thể dễ dàng nắm bắt những từ ngữ mà các nhà nghiên cứu hàng đầu sử dụng: tại sao lại là những từ cụ thể đó, tại sao không phải là những từ khác. & những người biết từ & thích đọc phương trình nhưng gặp khó khăn khi nhìn các phương trình đó trong mã & cách chúng kết nối có thể dễ dàng hiểu được khía cạnh thực tế của RL hơn.

- **About this book.** *Grokking DRL* bridges gap between theory & practice of DRL. Book's target audience is folks familiar with ML techniques, who want to learn RL. Book begins with foundations of DRL. It then provides an in-depth exploration of algorithms & techniques for DRL. Lastly, it provides a survey of advanced techniques with potential for making an impact.

  – *Grokking DRL* thu hẹp khoảng cách giữa lý thuyết & thực hành DRL. Đối tượng mục tiêu của cuốn sách là những người đã quen thuộc với các kỹ thuật ML, những người muốn tìm hiểu về RL. Cuốn sách bắt đầu với nền tảng của DRL. Sau đó, nó cung cấp 1 cái nhìn sâu sắc về các thuật toán & kỹ thuật cho DRL. Cuối cùng, nó cung cấp 1 cái nhìn tổng quan về các kỹ thuật tiên tiến có tiềm năng tạo ra tác động.

  - **Who should read this book.** Folks who are comfortable with a research field, Python code, a bit of math here & there, lots of intuitive explanations, & fun & concrete examples to drive learning will enjoy this book. However, any person only familiar with Python can get a lot, given enough interest in learning. Even though basic DL knowledge is assumed, this book provides a brief refresher on neural networks, backpropagation, & related techniques. Bottom line: this book is self contained, & anyone wanting to play around with AI agents & emerge grokking DRL can use this book to get them there.

    – Những người đã quen thuộc với lĩnh vực nghiên cứu, mã Python, 1 chút kiến thức toán học, nhiều giải thích trực quan, ví dụ cụ thể & thú vị để cung cấp việc học sẽ thích cuốn sách này. Tuy nhiên, bất kỳ ai chỉ thuộc về Python đều có thể học được rất nhiều, miễn là có đủ thú vị. Mặc dù cơ sở kiến thức về DL được mặc định, nhưng cuốn sách này cung cấp 1 số kiến thức cơ bản về mạng nơ-ron, lan truyền ngược & các kỹ thuật liên quan. Tóm lại: cuốn sách này hoàn toàn độc lập, bất kỳ ai cũng muốn tìm hiểu về các tác nhân AI & làm quen với DRL đều có thể sử dụng cuốn sách này để đạt được mục tiêu.

  - **How this book is organized: a roadmap.** This book has 13 chaps divided into 2 parts. In part 1, Chap. 1 introduces field of DRL & sets expectations for journey ahead. Chap. 2 introduces a framework for designing problems that RL agents can understand. Chap. 3 contains details of algorithms for solving RL problems when agent knows dynamic of world. Chap. 4 contains details of algorithms for solving simple RL problems when agent does not know dynamics of world. Chap. 5 introduces methods for solving prediction problem, which is a foundation for advanced RL methods.

    – Sách này gồm 13 chương, chia thành 2 phần. Trong phần 1, Chương 1 giới thiệu lĩnh vực DRL & đặt ra kỳ vọng cho hành trình phía trước. Chương 2 giới thiệu 1 khuôn khổ để thiết kế các bài toán mà các tác nhân RL có thể hiểu được. Chương 3 trình bày chi tiết về các thuật toán để giải các bài toán RL khi tác nhân biết động lực của thế giới. Chương 4 trình bày chi tiết về các thuật toán để giải các bài toán RL đơn giản khi tác nhân không biết động lực của thế giới. Chương 5 giới thiệu các phương pháp giải bài toán dự đoán, là nền tảng cho các phương pháp RL nâng cao.

    In part 2, Chap. 6 introduces methods for solving control problem, methods that optimize policies purely from trial-&-error learning. Chap. 7 teaches more advanced methods for RL, including methods that use planning for more sample efficiency. Chap. 8 introduces use of function approximation in RL by implementing a simple RL algorithm that uses neural networks for function approximation. Chap. 9 dives into more advanced techniques for using function approximation for solving RL problems. Chap. 10 teaches some of best techniques for further improving methods introduced so far. Chap. 11 introduces a slightly different technique for using DL models with RL that has proven to reach state-of-art performance in multiple deep RL benchmarks. Chap. 12 dives into more advanced methods for deep RL, state-of-art algorithms, & techniques commonly used for solving real-world problems. Chap. 13 surveys advanced research areas in RL that suggest best path for progress toward artificial general intelligence.

    – Trong phần 2, Chương 6 giới thiệu các phương pháp giải bài toán điều khiển, các phương pháp tối ưu hóa chính sách hoàn toàn từ học thử nghiệm. Chương 7 hướng dẫn các phương pháp nâng cao hơn cho RL, bao gồm các phương pháp sử dụng lập kế hoạch để có hiệu quả mẫu cao hơn. Chương 8 giới thiệu việc sử dụng xấp xỉ hàm trong RL bằng cách triển khai 1 thuật toán RL đơn giản sử dụng mạng nơ-ron để xấp xỉ hàm. Chương 9 đi sâu vào các kỹ thuật nâng cao hơn để sử dụng xấp xỉ hàm để giải các bài toán RL. Chương 10 hướng dẫn 1 số kỹ thuật tốt nhất để cải thiện hơn nữa các phương pháp đã giới thiệu cho đến nay. Chương 11 giới thiệu 1 kỹ thuật hơi khác để sử dụng các mô hình DL với RL đã được chứng minh là đạt hiệu suất tiên tiến trong nhiều chuẩn mực RL sâu. Chương 12 đi sâu vào các phương pháp nâng cao hơn cho RL sâu,

các thuật toán tiên tiến, & các kỹ thuật thường được sử dụng để giải quyết các vấn đề trong thế giới thực. Chương 13 khảo sát các lĩnh vực nghiên cứu tiên tiến trong RL gợi ý con đường tốt nhất để tiến tới trí tuệ nhân tạo tổng quát.

- **About code.** In many cases, original source code has been reformatted; added line breaks, renamed variables, & reword indentation to accommodate available page space in book. In rare cases, even this was not enough, & code includes line-continuation operator in Python, backslash, to indicate that a statement is continued on next line. Additionally, comments in source code have often been removed from boxes, & code is described in text. Code annotations point out important concepts.

  – Trong nhiều trường hợp, mã nguồn gốc đã được định dạng lại; thêm ngắt dòng, đổi tên biến, thụt lề & để phù hợp với không gian trang còn trống trong sách. Trong 1 số ít trường hợp, ngay cả điều này cũng không đủ, & code bao gồm toán tử tiếp tục dòng trong Python, dấu gạch chéo ngược, để chỉ ra rằng 1 câu lệnh được tiếp tục ở dòng tiếp theo. Ngoài ra, chú thích trong mã nguồn thường bị xóa khỏi các hộp, & code được mô tả bằng văn bản. Chú thích mã chỉ ra các khái niệm quan trọng.

- **1. Introduction to deep reinforcement learning.** You will learn: what DRL is & how it is different from other ML approaches, recent progress in DRL & what it can do for a variety of problems, what to expect from this book & how to get most out of it. Humans naturally pursue feelings of happiness. From picking out our meals to advancing our careers, every action we choose is derived from our drive to experience rewarding moments in life. Whether these moments are self-centered pleasures or more generous of goals, whether they bring us immediate gratification or long-term success, they're still our perception of how important & valuable they are. & to some extent, these moments are reason for our existence.

  – Bạn sẽ học được: DRL là gì & nó khác với các phương pháp ML khác như thế nào, những tiến bộ gần đây trong DRL & nó có thể làm gì cho nhiều vấn đề khác nhau, những gì mong đợi từ cuốn sách này & cách tận dụng tối đa nó. Con người vốn dĩ theo đuổi cảm giác hạnh phúc. Từ việc chọn bữa ăn cho đến thăng tiến trong sự nghiệp, mọi hành động chúng ta lựa chọn đều xuất phát từ động lực trải nghiệm những khoảnh khắc đáng giá trong cuộc sống. Cho dù những khoảnh khắc này là niềm vui ích kỷ hay những mục tiêu lớn lao hơn, cho dù chúng mang lại cho chúng ta sự hài lòng tức thời hay thành công lâu dài, chúng vẫn là nhận thức của chúng ta về tầm quan trọng & giá trị của chúng. & ở 1 mức độ nào đó, những khoảnh khắc này là lý do tồn tại của chúng ta.

  Our ability to achieve these precious moments seems to be correlated with intelligence; "intelligence" is defined as ability to acquire & apply knowledge & skills. People who are deemed by society as intelligent are capable of trading not only immediate satisfaction for long-term goals, but also a good, certain future for a possibly better, yet uncertain, one. Goals that take longer to materialize & that have unknown long-term value are usually hardest to achieve, & those who can withstand challenges along way are exception, leaders, intellectuals of society.

  – Khả năng đạt được những khoảnh khắc quý giá này dường như có mối tương quan với trí thông minh; "trí thông minh" được định nghĩa là khả năng tiếp thu & áp dụng kiến thức & kỹ năng. Những người được xã hội coi là thông minh có khả năng đánh đổi không chỉ sự hài lòng tức thời để lấy những mục tiêu dài hạn, mà còn cả 1 tương lai tốt đẹp, chắc chắn để lấy 1 tương lai có thể tốt đẹp hơn nhưng vẫn còn bấp bênh. Những mục tiêu mất nhiều thời gian hơn để hiện thực hóa & có giá trị lâu dài chưa được biết đến thường khó đạt được nhất, & những người có thể vượt qua thử thách trên đường đi là những ngoại lệ, những nhà lãnh đạo, những trí thức của xã hội.

  In this book, learn about an approach, known as DRL, involved with creating computer programs that can achieve goals that require intelligence. In this chap, introduce DRL & give suggestions to get most out of this book.

  – Trong cuốn sách này, bạn sẽ tìm hiểu về 1 phương pháp, được gọi là DRL, liên quan đến việc tạo ra các chương trình máy tính có thể đạt được các mục tiêu đòi hỏi trí thông minh. Trong chương này, hãy giới thiệu DRL & đưa ra các gợi ý để tận dụng tối đa cuốn sách này.

  - **1.1. What is DRL?** Deep reinforcement learning (DRL) is a ML approach to AI concerned with creating computer programs that can solve problems requiring intelligence. Distinct property of DRL programs is learning through trial & error from feedback that's simultaneously sequential, evaluative, & sampled by leveraging powerful nonlinear function approximation.

    – Học tăng cường sâu (DRL) là 1 phương pháp tiếp cận ML đối với AI, tập trung vào việc tạo ra các chương trình máy tính có thể giải quyết các vấn đề đòi hỏi trí tuệ. Đặc điểm nổi bật của các chương trình DRL là học thông qua thử nghiệm & sai từ phản hồi, vừa tuần tự, vừa đánh giá, & lấy mẫu bằng cách tận dụng phép xấp xỉ hàm phi tuyến tính mạnh mẽ.

    Want to unpack this definition for you 1 bit at a time. But, don't get too caught up with details because it will take me whole book to get you grokking DRL. Following is introduction to what you learn about in this book. As such, it's repeated & explained in detail in chaps ahead. If I succeed with my goal for this book, after you complete it, you should understand this definition precisely. You should be able to tell why I used words that I used, & why I didn't use more or fewer words. But, for this chap, simply sit back & plow through it.

    – Tôi muốn giải thích định nghĩa này cho bạn từng chút một. Tuy nhiên, đừng quá chú trọng vào chi tiết vì tôi sẽ phải mất cả cuốn sách để giúp bạn hiểu rõ DRL. Sau đây là phần giới thiệu về những gì bạn sẽ học được trong cuốn sách này. Vì vậy, nó sẽ được lặp lại & giải thích chi tiết trong các chương tiếp theo. Nếu tôi đạt được mục tiêu của mình trong cuốn sách này, sau khi bạn hoàn thành nó, bạn sẽ hiểu chính xác định nghĩa này. Bạn sẽ có thể hiểu tại sao tôi sử dụng những từ ngữ đó, & tại sao tôi không sử dụng nhiều hơn hoặc ít hơn từ ngữ. Nhưng, đối với chương này, chỉ cần ngồi lại & đọc kỹ nó.

    * **1.1.1. DRL is a ML approach to AI.** AI is a branch of CS involved in creation of computer programs capable of demonstrating intelligence. Traditionally, any piece of software that displays cognitive abilities e.g. perception, search, planning, & learning is considered part of AI. Several examples of functionality produced by AI software:

· Pages returned by a search engine
· Route produced by a GPS app
· Voice recognition & synthetic voice of smart-assistant software
· Products recommended by e-commerce sites
· Follow-me feature in drones

All computer programs that display intelligence are considered AI, but not all examples of AI can learn. ML is area of AI concerned with creating computer programs that can solve problems requiring intelligence by learning from data. There are 3 main branches of ML: supervised, unsupervised, & reinforcement learning.

– Trí tuệ nhân tạo (AI) là 1 nhánh của Khoa học máy tính (CS) liên quan đến việc tạo ra các chương trình máy tính có khả năng thể hiện trí thông minh. Theo truyền thống, bất kỳ phần mềm nào thể hiện khả năng nhận thức, ví dụ như nhận thức, tìm kiếm, lập kế hoạch, & học tập đều được coi là 1 phần của AI. 1 số ví dụ về chức năng do phần mềm AI tạo ra:

· Các trang được trả về bởi công cụ tìm kiếm
· Lộ trình do ứng dụng GPS tạo ra
· Nhận dạng giọng nói & giọng nói tổng hợp của phần mềm trợ lý thông minh
· Các sản phẩm được các trang thương mại điện tử đề xuất
· Tính năng theo dõi tôi trên máy bay không người lái

Tất cả các chương trình máy tính thể hiện trí thông minh đều được coi là AI, nhưng không phải tất cả các ví dụ về AI đều có thể học. Học máy (ML) là lĩnh vực của AI liên quan đến việc tạo ra các chương trình máy tính có thể giải quyết các vấn đề đòi hỏi trí thông minh bằng cách học hỏi từ dữ liệu. Có 3 nhánh chính của Học máy: học có giám sát, học không giám sát & học tăng cường.

Main branches of ML. 1. These types SL, UL, RL of ML tasks are all important, & they aren't mutually exclusive. 2. Best examples of AI combine many different techniques.

– **Các nhánh chính của ML.** 1. Các loại nhiệm vụ SL, UL, RL của ML đều quan trọng & chúng không loại trừ lẫn nhau. 2. Các ví dụ tốt nhất về AI kết hợp nhiều kỹ thuật khác nhau.

Supervised learning (SL) is task of learning from labeled data. In SL, a human decides which data to collect & how to label it. Goal in SL is to generalize. A classic example of SL is a handwritten-digit-recognition application: a human gathers images with handwritten digits, labels those images, & trains a model to recognize & classify digits in images correctly. Trained model is expected to generalize & correctly classify handwritten digits in new images.

– Học có giám sát (SL) là nhiệm vụ học từ dữ liệu được gắn nhãn. Trong SL, con người quyết định dữ liệu nào cần thu thập & cách gắn nhãn. Mục tiêu của SL là khái quát hóa. 1 ví dụ điển hình về SL là ứng dụng nhận dạng chữ số viết tay: con người thu thập hình ảnh có chữ số viết tay, gắn nhãn cho hình ảnh đó, & huấn luyện mô hình để nhận dạng & phân loại chính xác các chữ số trong hình ảnh. Mô hình đã được huấn luyện được kỳ vọng sẽ khái quát hóa & phân loại chính xác các chữ số viết tay trong hình ảnh mới.

Unsupervised learning (UL) is task of learning from unlabeled data. Even though data no longer needs labeling, methods used by computer to gather data still need to be designed by a human. Goal in UL is to compress. A classic example of UL is a customer segmentation application; a human collects customer data & trains a model to group customers into clusters. These clusters compress information, uncovering underlying relationships in customers.

– Học không giám sát (UL) là nhiệm vụ học từ dữ liệu chưa được gắn nhãn. Mặc dù dữ liệu không còn cần được gắn nhãn, các phương pháp máy tính sử dụng để thu thập dữ liệu vẫn cần được thiết kế bởi con người. Mục tiêu của UL là nén. 1 ví dụ điển hình về UL là ứng dụng phân khúc khách hàng; con người thu thập dữ liệu khách hàng & huấn luyện mô hình để nhóm khách hàng thành các cụm. Các cụm này nén thông tin, khám phá các mối quan hệ cơ bản trong khách hàng.

Reinforcement learning (RL) is task of learning through trial & error. In this type of task, no human labels data, & no human collects or explicitly designs collection of data. Goal in RL is to act. A classic example of RL is a Pong-playing agent; agent repeatedly interacts with a Pong emulator & learns by taking actions & observing their effects. Trained agent is expected to act in such a way that is successfully plays Pong.

– Học tăng cường (RL) là nhiệm vụ học thông qua thử & sai. Trong loại nhiệm vụ này, không có con người nào dán nhãn dữ liệu, không có con người nào thu thập hoặc thiết kế rõ ràng bộ sưu tập dữ liệu. Mục tiêu trong RL là hành động. 1 ví dụ điển hình về RL là 1 tác nhân chơi Pong; tác nhân liên tục tương tác với trình giả lập Pong & học bằng cách thực hiện các hành động & quan sát hiệu ứng của chúng. Tác nhân được đào tạo được kỳ vọng sẽ hành động theo cách chơi Pong thành công.

A powerful recent approach to ML, called *deep learning* (DL), involves using multi-layered nonlinear function approximation, typically neural networks. DL isn't a separate branch of ML, so it's not a different task than those described previously. DL is a collection of techniques & methods for using neural networks to solve ML tasks, whether SL, UL, or RL. DRL is simply use of DL to solve RL tasks.

– 1 phương pháp tiếp cận mạnh mẽ gần đây đối với ML, được gọi là *deep learning* (DL), liên quan đến việc sử dụng phép xấp xỉ hàm phi tuyến tính đa lớp, thường là mạng nơ-ron. DL không phải là 1 nhánh riêng biệt của ML, vì vậy nó không khác biệt so với các nhiệm vụ đã mô tả trước đây. DL là tập hợp các kỹ thuật & phương pháp sử dụng mạng nơ-ron để giải quyết các nhiệm vụ ML, dù là SL, UL hay RL. DRL đơn giản là việc sử dụng DL để giải quyết các nhiệm vụ RL.

DL is a powerful toolbox. Important thing here is DL is a toolbox, & any advancement in field of DL is felt in all of ML. DRL is intersection of RL & DL. Bottom line: DRL is an approach to a problem. Field of AI defines problem:

creating intelligent machines. 1 of approaches to solving that problem is DRL. Throughout book, will you find comparisons between RL & other ML approaches, but only in this chap will you find definitions & a historical overview of AI in general. Important to note: field of RL includes field of DRL, so although I make a distinction when necessary, when refer to RL, remember DRL is included.

– **DL là 1 bộ công cụ mạnh mẽ.** Điều quan trọng ở đây là DL là 1 bộ công cụ, & bất kỳ tiến bộ nào trong lĩnh vực DL đều được cảm nhận trong toàn bộ ML. DRL là giao điểm của RL & DL. Tóm lại: DRL là 1 cách tiếp cận vấn đề. Lĩnh vực AI định nghĩa vấn đề: tạo ra máy móc thông minh. 1 trong những cách tiếp cận để giải quyết vấn đề đó là DRL. Xuyên suốt cuốn sách, bạn sẽ tìm thấy những so sánh giữa RL & các cách tiếp cận ML khác, nhưng chỉ trong chương này, bạn mới tìm thấy các định nghĩa & tổng quan về lịch sử AI nói chung. Lưu ý quan trọng: lĩnh vực RL bao gồm cả lĩnh vực DRL, vì vậy mặc dù tôi có phân biệt khi cần thiết, nhưng khi đề cập đến RL, hãy nhớ rằng DRL cũng được bao gồm.

∗ **1.1.2. DRL is concerned with creating computer programs.** At its core, DRl is about complex sequential decision-making problems under uncertainty. But, this is a topic of interest in many fields; e.g., *control theory* (CT) studies ways to control complex known dynamic systems. In CT, dynamics of systems we try to control are usually known in advance. *Operations research* (OR), another instance, also studies decision-making under uncertainty, but problems in this field often have much larger action spaces than those commonly seen in DRL. *Psychology* studies human behavior, which is partly same "complex sequential decision-making under uncertainty" problem.

– DRL liên quan đến việc tạo ra các chương trình máy tính. Về bản chất, DRl tập trung vào các vấn đề ra quyết định tuần tự phức tạp trong điều kiện không chắc chắn. Tuy nhiên, đây là 1 chủ đề được quan tâm trong nhiều lĩnh vực; ví dụ, *lý thuyết điều khiển* (CT) nghiên cứu các cách kiểm soát các hệ thống động phức tạp đã biết. Trong CT, động lực của các hệ thống mà chúng ta cố gắng kiểm soát thường được biết trước. *nghiên cứu vận hành* (OR), 1 ví dụ khác, cũng nghiên cứu việc ra quyết định trong điều kiện không chắc chắn, nhưng các vấn đề trong lĩnh vực này thường có không gian hành động lớn hơn nhiều so với những vấn đề thường thấy trong DRL. *tâm lý học* nghiên cứu hành vi con người, 1 phần cũng là vấn đề "ra quyết định tuần tự phức tạp trong điều kiện không chắc chắn".

Synergy between similar fields. 1. All of these fields (& many more) study complex sequential decision-making under uncertainty. 2. As a result, there's a synergy between these fields. E.g., RL & optimal control both contribute to research of model-based methods. 3. Or RL & operations research both contribute to study of problems with large action spaces. 4. Downside is an inconsistency in notation, definitions, & so on, that makes it hard for newcomers to find their way around.

– **Sự tương tác giữa các lĩnh vực tương tự.** 1. Tất cả các lĩnh vực này (& nhiều lĩnh vực khác nữa) đều nghiên cứu quá trình ra quyết định tuần tự phức tạp trong điều kiện không chắc chắn. 2. Do đó, có sự tương tác giữa các lĩnh vực này. Ví dụ, RL & điều khiển tối ưu đều góp phần vào việc nghiên cứu các phương pháp dựa trên mô hình. 3. Hoặc RL & nghiên cứu hoạt động đều góp phần vào việc nghiên cứu các bài toán có không gian hành động lớn. 4. Nhược điểm là sự thiếu nhất quán trong ký hiệu, định nghĩa, v.v., khiến người mới khó nắm bắt.

Bottom line: you have come to a field that's influenced by a variety of others. Although this is a good thing, it also brings inconsistencies in terminologies, notations, & so on. My take is CS approach to this problem, so this book is about building computer programs that solve complex decision-making problems under uncertainty, & as such, you can find code examples throughout book.

– Tóm lại: bạn đã đến 1 lĩnh vực chịu ảnh hưởng của nhiều lĩnh vực khác. Mặc dù điều này là tốt, nhưng nó cũng mang lại sự không nhất quán về thuật ngữ, ký hiệu, v.v. Quan điểm của tôi là tiếp cận vấn đề này theo hướng Khoa học Máy tính, vì vậy cuốn sách này nói về việc xây dựng các chương trình máy tính giải quyết các vấn đề ra quyết định phức tạp trong điều kiện không chắc chắn, & do đó, bạn có thể tìm thấy các ví dụ mã trong suốt cuốn sách.

In DRL, these computer programs are called *agents*. An agent is a decision maker only & nothing else. I.e., if you are training a robot to pick up objects, robot arm isn't part of agent. Only code that makes decisions is referred to as agent.

– Trong DRL, các chương trình máy tính này được gọi là *agents*. 1 tác nhân chỉ là người ra quyết định & không gì khác. Ví dụ, nếu bạn đang huấn luyện 1 robot nhặt đồ vật, thì cánh tay robot không phải là 1 phần của tác nhân. Chỉ có mã lệnh đưa ra quyết định mới được gọi là tác nhân.

∗ **1.1.3. DRL agents can solve problems that require intelligence.** On other side of agent is *environment*. Environment is everything outside agent; everything agent has no total control over. Again, imagine you are training a robot to pick up objects. Objects to be picked up, tray where objects lay, wind, & everything outside decision maker are part of environment. I.e., robot arm is also part of environment because it isn't part of agent. & even though agent can decide to move arm, & actual arm movement is noisy, & thus arm is part of environment.

– DRL agent có thể giải quyết các vấn đề đòi hỏi trí thông minh. Phía bên kia của agent là *environment*. Môi trường là mọi thứ bên ngoài agent; mọi thứ mà agent không thể kiểm soát hoàn toàn. 1 lần nữa, hãy tưởng tượng bạn đang huấn luyện 1 robot nhặt đồ vật. Các đồ vật cần nhặt, khay chứa đồ vật, gió, & mọi thứ bên ngoài bộ phận ra quyết định đều là 1 phần của môi trường. I.e., cánh tay robot cũng là 1 phần của môi trường vì nó không phải là 1 phần của agent. & mặc dù agent có thể quyết định di chuyển cánh tay, & chuyển động thực tế của cánh tay bị nhiễu, & do đó cánh tay là 1 phần của môi trường.

This strict boundary between agent & environment is counterintuitive at 1st, but decision maker, agent, can only have a single role: making decisions. Everything that comes after decision gets bundled into environment.

– Ranh giới chặt chẽ giữa tác nhân & môi trường này thoạt đầu có vẻ trái ngược với trực giác, nhưng người ra quyết định, tức tác nhân, chỉ có thể có 1 vai trò duy nhất: ra quyết định. Mọi thứ diễn ra sau quyết định đều được gói gọn trong môi trường.

Boundary between agent & environment. 1. An agent is decision-making portion of code. 2. Environment is everything outside agent. In this case that includes network latencies, motor's noise, camera noise, & so on. This may seem counterintuitive at 1st, but it helps in understanding algorithms.
– **Ranh giới giữa tác nhân & môi trường.** 1. Tác nhân là phần mã ra quyết định. 2. Môi trường là mọi thứ bên ngoài tác nhân. Trong trường hợp này, điều đó bao gồm độ trễ mạng, tiếng ồn của động cơ, tiếng ồn của camera, & vân vân. Điều này thoạt đầu có vẻ trái ngược, nhưng nó giúp hiểu các thuật toán.
Chap. 2 provides an in-depth survey of all components of DRL. Following is a preview of what you will learn in Chap. 2. Environment is represented by a set of variables related to problem. E.g., in robotic arm example, location & velocities of arm would be part of variables that make up environment. This set of variables & all possible values that they can take are referred to as *state space*. A state is an instantiation of state space, a set of values variables take.
– Chương 2 cung cấp 1 cái nhìn sâu sắc về tất cả các thành phần của DRL. Sau đây là phần tóm tắt những gì bạn sẽ học trong Chương 2. Môi trường được biểu diễn bằng 1 tập hợp các biến liên quan đến vấn đề. Ví dụ, trong ví dụ về cánh tay robot, vị trí & vận tốc của cánh tay sẽ là 1 phần của các biến tạo nên môi trường. Tập hợp các biến này & tất cả các giá trị khả dĩ mà chúng có thể nhận được được gọi là *không gian trạng thái*. 1 trạng thái là 1 thể hiện của không gian trạng thái, 1 tập hợp các giá trị mà các biến nhận được.
Interestingly, often, agents don't have access to actual full state of environment. Part of a state that agent can observe is called an *observation*. Observations depend on states but are what agent can see. E.g., in robotic arm example, agent may only have access to camera images. While an exact location of each object exists, agent doesn't have access to this specific state. Instead, observations agent perceives are derived from states. You will often see in literature observations & states being used interchangeably, including in this book. Apologize for inconsistencies. Simply know differences & be aware of lingo; that's what matters.
– Điều thú vị là, thông thường, các tác nhân không có quyền truy cập vào trạng thái đầy đủ thực tế của môi trường. 1 phần của trạng thái mà tác nhân có thể quan sát được gọi là *observation*. Các quan sát phụ thuộc vào trạng thái nhưng là những gì tác nhân có thể nhìn thấy. Ví dụ, trong ví dụ về cánh tay robot, tác nhân có thể chỉ có quyền truy cập vào hình ảnh camera. Mặc dù vị trí chính xác của từng vật thể tồn tại, nhưng tác nhân không có quyền truy cập vào trạng thái cụ thể này. Thay vào đó, các quan sát mà tác nhân nhận thức được bắt nguồn từ các trạng thái. Bạn sẽ thường thấy trong các tài liệu, các thuật ngữ "quan sát" & "trạng thái" được sử dụng thay thế cho nhau, kể cả trong cuốn sách này. Xin lỗi vì sự không nhất quán. Chỉ cần biết sự khác biệt & lưu ý đến thuật ngữ chuyên ngành; đó mới là điều quan trọng.
State vs. observations. State: true locations. 1. States are perfect & complete information related to task at hand. Observation: just an image. 2. While observations are information agent receives, this could be noisy of incomplete information.
– **Trạng thái so với quan sát.** Trạng thái: vị trí thực. 1. Trạng thái là thông tin hoàn hảo & đầy đủ liên quan đến nhiệm vụ đang thực hiện. Quan sát: chỉ là hình ảnh. 2. Mặc dù quan sát là thông tin mà tác nhân nhận được, nhưng thông tin này có thể bị nhiễu hoặc không đầy đủ.
At each state, environment makes available a set of actions agent can choose from. Agent influences environment through these actions. Environment may change states as a response to agent's action. Function that's responsible for this mapping is called *transition function*. Environment may also provide a reward signal as a response. Function responsible for this mapping is called *reward function*. Set of transition & reward functions is referred to as *model* of environment.
– Ở mỗi trạng thái, môi trường cung cấp 1 tập hợp các hành động mà tác nhân có thể lựa chọn. Tác nhân tác động đến môi trường thông qua các hành động này. Môi trường có thể thay đổi trạng thái như 1 phản ứng với hành động của tác nhân. Hàm chịu trách nhiệm cho việc ánh xạ này được gọi là *hàm chuyển tiếp*. Môi trường cũng có thể cung cấp tín hiệu thưởng như 1 phản ứng. Hàm chịu trách nhiệm cho việc ánh xạ này được gọi là *hàm thưởng*. Tập hợp các hàm chuyển tiếp & phần thưởng được gọi là *mô hình* của môi trường.
RL cycle. 1. Cycle begins with agent observing environment. → Observation & reward. 2. Agent uses this observation & reward to attempt to improve at task. → Improve. 3. It then sends an action to environment in an attempt to control it in a favorable way. → Action. 4. Finally, environment transitions & its internal state (likely) changes as a consequence of previous state & agent's action. Then, cycle repeats.
– **Chu kỳ RL.** 1. Chu kỳ bắt đầu với việc tác nhân quan sát môi trường. → Quan sát & phần thưởng. 2. Tác nhân sử dụng quan sát & phần thưởng này để cố gắng cải thiện nhiệm vụ. → Cải thiện. 3. Sau đó, nó gửi 1 hành động đến môi trường để cố gắng kiểm soát nó theo cách thuận lợi. → Hành động. 4. Cuối cùng, môi trường chuyển đổi & trạng thái bên trong của nó (có thể) thay đổi do hậu quả của trạng thái & hành động trước đó của tác nhân. Sau đó, chu kỳ lặp lại.
Environment commonly has a well-defined task. Goal of this task is defined through reward function. Reward-function signals can be simultaneously sequential, evaluative, & sampled. To achieve goal, agent needs to demonstrate intelligence, or at least cognitive abilities commonly associated with intelligence, e.g. long-term thinking, information gathering, & generalization.
– Môi trường thường có 1 nhiệm vụ được xác định rõ ràng. Mục tiêu của nhiệm vụ này được xác định thông qua hàm thưởng. Các tín hiệu hàm thưởng có thể đồng thời tuần tự, đánh giá & lấy mẫu. Để đạt được mục tiêu, tác nhân cần thể hiện trí thông minh, hoặc ít nhất là các khả năng nhận thức thường gắn liền với trí thông minh, ví dụ như tư duy dài hạn, thu thập thông tin & khái quát hóa.
Agent has a 3-step process: agent interacts with environment, agent evaluates its behavior, & agent improves its responses. Agent can be designed to learn mappings from observations to actions called *policies*. Agent can be designed to learn model of environment on mappings called *models*. Agent can be designed to learn to estimate reward-to-go on mappings called *value functions*.

– Tác nhân có quy trình 3 bước: tác nhân tương tác với môi trường, tác nhân đánh giá hành vi của mình, & tác nhân cải thiện phản hồi của mình. Tác nhân có thể được thiết kế để học các phép ánh xạ từ quan sát đến hành động được gọi là *policies*. Tác nhân có thể được thiết kế để học mô hình môi trường trên các phép ánh xạ được gọi là *models*. Tác nhân có thể được thiết kế để học cách ước tính phần thưởng-để-đi trên các phép ánh xạ được gọi là *value functions*.

* **1.1.4. DRL agents improve their behavior through trial-&-error learning.** Interactions between agent & environment go on for several cycles. Each cycle is called a *time step*. At each time step, agent observes environment, takes action, & receives a new observation & reward. Set of state, action, reward, & new state is called an *experience*. Every experience has an opportunity for learning & improving performance.

– Các tác nhân DRL cải thiện hành vi của chúng thông qua quá trình học thử & sai. Tương tác giữa tác nhân & môi trường diễn ra trong nhiều chu kỳ. Mỗi chu kỳ được gọi là *bước thời gian*. Tại mỗi bước thời gian, tác nhân quan sát môi trường, thực hiện hành động, & nhận được 1 quan sát mới & phần thưởng. Tập hợp trạng thái, hành động, phần thưởng, & trạng thái mới được gọi là *trải nghiệm*. Mỗi trải nghiệm đều có cơ hội để học & cải thiện hiệu suất.

Task agent is trying to solve may or may not have a natural ending. Tasks that have a natural ending, e.g. a game, are called *episodic tasks*. Conversely, tasks that don't are called *continuing tasks*, e.g. learning forward motion. Sequence of time steps from beginning to end of an episodic task is called an *episode*. Agents may take several time steps & episodes to learn to solve a task. Agents learn through trial & error: they try something, observe, learn, try something else, & so on.

– Nhiệm vụ mà tác nhân đang cố gắng giải quyết có thể có hoặc không có kết thúc tự nhiên. Các nhiệm vụ có kết thúc tự nhiên, ví dụ: trò chơi, được gọi là *là nhiệm vụ theo từng tập*. Ngược lại, các nhiệm vụ không có kết thúc tự nhiên được gọi là *là nhiệm vụ liên tục*, ví dụ: học chuyển động tiến về phía trước. Chuỗi các bước thời gian từ đầu đến cuối của 1 nhiệm vụ theo từng tập được gọi là *là tập*. Các tác nhân có thể mất nhiều bước thời gian & tập để học cách giải quyết 1 nhiệm vụ. Các tác nhân học hỏi thông qua thử & sai: họ thử 1 cái gì đó, quan sát, học hỏi, thử 1 cái gì đó khác, & cứ thế.

Start learning more about this cycle in Chap. 4, which contains a type of environment with a single step per episode. Starting with Chap. 5, learn to deal with environments that require more than a single interaction cycle per episode.

– Bắt đầu tìm hiểu thêm về chu trình này trong Chương 4, trong đó có 1 loại môi trường với 1 bước duy nhất cho mỗi tập. Bắt đầu từ Chương 5, hãy học cách xử lý các môi trường đòi hỏi nhiều hơn 1 chu trình tương tác cho mỗi tập.

* **1.1.5. DRL agents learn from sequential feedback.** Action taken by agent may have delayed consequences. Reward may be sparse & only manifest after several time steps. Thus agent must be able to learn from sequential feedback. Sequential feedback gives rise to a problem referred to as *temporal credit assignment problem*. Temporal credit assignment problem is challenge of determining which state &/or action is responsible for a reward. When there's a temporal component to a problem, & actions have delayed consequences, it's challenging to assign credit for rewards.

– Các tác nhân DRL học hỏi từ phản hồi tuần tự. Hành động do tác nhân thực hiện có thể có hậu quả bị trì hoãn. Phần thưởng có thể thưa thớt & chỉ biểu hiện sau 1 vài bước thời gian. Do đó, tác nhân phải có khả năng học hỏi từ phản hồi tuần tự. Phản hồi tuần tự dẫn đến 1 vấn đề được gọi là *vấn đề gán tín dụng tạm thời*. Vấn đề gán tín dụng tạm thời là thách thức trong việc xác định trạng thái &/hoặc hành động nào chịu trách nhiệm cho phần thưởng. Khi 1 vấn đề có yếu tố thời gian, & các hành động có hậu quả bị trì hoãn, việc gán tín dụng cho phần thưởng sẽ rất khó khăn.

Difficulty of temporal credit assignment problem. In Chap. 3, study ins & outs of sequential feedback in isolation. I.e., your programs learn from simultaneously sequential, supervised (as opposed to evaluative), & exhaustive (as opposed to sampled) feedback.

– **Khó khăn của bài toán gán tín chỉ tạm thời.** Trong Chương 3, hãy nghiên cứu chi tiết & chi tiết về phản hồi tuần tự 1 cách riêng biệt. I.e., chương trình của bạn học từ phản hồi tuần tự, có giám sát (khác với phản hồi đánh giá), & toàn diện (khác với phản hồi lấy mẫu) đồng thời.

* **1.1.6. DRL agents learn from evaluative feedback.** Reward received by agent may be weak, in sense: it may provide no supervision. Reward may indicate goodness & not correctness, meaning it may contain no information about other potential rewards. Thus agent must be able to learn from *evaluative feedback*. Evaluative feedback gives rise to need for exploration. Agent must be able to balance gathering of information with exploitation of current information. This is also referred to as *exploration vs. exploitation trade-off*.

– Các tác nhân DRL học hỏi từ phản hồi đánh giá. Phần thưởng mà tác nhân nhận được có thể yếu, theo nghĩa nào đó: nó có thể không cung cấp sự giám sát. Phần thưởng có thể chỉ ra sự tốt đẹp & không phải sự chính xác, nghĩa là nó có thể không chứa thông tin về các phần thưởng tiềm năng khác. Do đó, tác nhân phải có khả năng học hỏi từ *phản hồi đánh giá*. Phản hồi đánh giá làm nảy sinh nhu cầu khám phá. Tác nhân phải có khả năng cân bằng giữa việc thu thập thông tin với việc khai thác thông tin hiện có. Điều này cũng được gọi là *sự đánh đổi giữa khám phá & khai thác*.

Fig: Difficult of exploration vs. exploitation trade-off. In Chap. 4, study ins & outs of evaluative feedback in isolation. I.e., your programs will learn from feedback that is simultaneously 1-shot (as opposed to sequential), evaluative, & exhaustive (as opposed to sampled).

– Hình: Khó khăn trong việc đánh đổi giữa khám phá & khai thác. Trong Chương 4, hãy nghiên cứu kỹ lưỡng về phản hồi đánh giá 1 cách riêng biệt. I.e., chương trình của bạn sẽ học từ phản hồi vừa mang tính 1 lần (khác với phản hồi tuần tự), vừa mang tính đánh giá, vừa mang tính toàn diện (khác với phản hồi lấy mẫu).

* **1.1.7. DRL agents learn from sampled feedback.** Reward received by agent is merely a sample, & agent doesn't have access to reward function. Also, state & action spaces are commonly large, even infinite, so trying to learn from sparse & weak feedback becomes a harder challenge with samples. Therefore, agent must be able to learn from sampled feedback, & it must be able to generalize.

– Các tác nhân DRL học hỏi từ phản hồi lấy mẫu. Phần thưởng mà tác nhân nhận được chỉ là 1 mẫu, & tác nhân không có

quyền truy cập vào hàm thưởng. Ngoài ra, không gian trạng thái & hành động thường rất lớn, thậm chí vô hạn, vì vậy việc cố gắng học hỏi từ phản hồi thưa thớt & yếu trở nên khó khăn hơn với các mẫu. Do đó, tác nhân phải có khả năng học hỏi từ phản hồi lấy mẫu, & nó phải có khả năng khái quát hóa.

Fig: Difficulty of learning from sampled feedback. Agents that are designed to approximate policies are called *policy-based*; agents, that are designed to approximate value functions are called *value-based*; agents that are designed to approximate models are called *model-based*; & agents that are designed to approximate both policies & value functions are called *actor-critic*. Agents can be designed to approximate 1 or more of these components.

– Hình: Khó khăn trong việc học từ phản hồi lấy mẫu. Các tác nhân được thiết kế để ước lượng các chính sách được gọi là *policy-based*; các tác nhân được thiết kế để ước lượng các hàm giá trị được gọi là *value-based*; các tác nhân được thiết kế để ước lượng các mô hình được gọi là *model-based*; & các tác nhân được thiết kế để ước lượng cả 2 chính sách & các hàm giá trị được gọi là *actor-critic*. Các tác nhân có thể được thiết kế để ước lượng 1 hoặc nhiều thành phần này.

∗ **1.1.8. DRL agents use powerful nonlinear function approximation.** Agent can approximate functions using a variety of ML methods & techniques, from decision trees to SVMs to neural networks. However, in this book, use only neural networks; this is what "deep" part of DRL refers to, after all. Neural networks aren't necessarily best solution to every problem; neural networks are data hungry & challenging to interpret, & you must keep these facts in mind. However, neural networks are among most potent function approximations available, & their performance is often the best.

– Các tác nhân DRL sử dụng phép xấp xỉ hàm phi tuyến tính mạnh mẽ. Tác nhân có thể xấp xỉ các hàm bằng nhiều phương pháp & kỹ thuật ML khác nhau, từ cây quyết định đến SVM & mạng nơ-ron. Tuy nhiên, trong cuốn sách này, chúng tôi chỉ sử dụng mạng nơ-ron; xét cho cùng, đây chính là phần "sâu" của DRL. Mạng nơ-ron không nhất thiết là giải pháp tốt nhất cho mọi vấn đề; mạng nơ-ron rất cần dữ liệu & khó diễn giải, & bạn phải ghi nhớ những điều này. Tuy nhiên, mạng nơ-ron là 1 trong những phép xấp xỉ hàm mạnh mẽ nhất hiện có, & hiệu suất của chúng thường là tốt nhất.

Artificial neural networks (ANN) are multi-layered nonlinear function approximators loosely inspired by biological neural networks in animal brains. An ANN isn't an algorithm, but a structure composed of multiple layers of mathematical transformations applied to input values.

– Mạng nơ-ron nhân tạo (ANN) là các hàm xấp xỉ phi tuyến tính nhiều lớp, lấy cảm hứng từ mạng nơ-ron sinh học trong não động vật. ANN không phải là 1 thuật toán, mà là 1 cấu trúc bao gồm nhiều lớp phép biến đổi toán học được áp dụng cho các giá trị đầu vào.

From Chaps. 3–7, only deal with problems in which agents learn from exhaustive (as opposed to sampled) feedback. Starting with Chap. 8, study full DRL problem; i.e., using DNNs so that agents can learn from sampled feedback. Remember, DRL agents learn from feedback that's simultaneously sequential, evaluative, & sampled.

– Từ Chương 3 đến Chương 7, chỉ đề cập đến các vấn đề mà tác nhân học hỏi từ phản hồi toàn diện (thay vì phản hồi lấy mẫu). Bắt đầu từ Chương 8, hãy nghiên cứu toàn bộ bài toán DRL; i.e., sử dụng DNN để tác nhân có thể học hỏi từ phản hồi lấy mẫu. Hãy nhớ rằng, tác nhân DRL học hỏi từ phản hồi vừa tuần tự, vừa đánh giá, vừa lấy mẫu.

◦ **1.2. Past, present, & future of DRL.** History isn't necessary to gain skills, but it can allow you to understand context around a topic, which in turn can help you gain motivation, & therefore, skills. History of AI & DRL should help you set expectations about future of this powerful technology. At times, feel hype surrounding AI is actually productive; people get interested. But, right after that, when it's time to put in work, hype no longer helps, & it's a problem. Although I'd like to be excited about AI, also need to set realistic expectations.

– Quá khứ, hiện tại, & tương lai của DRL. Lịch sử không nhất thiết phải là thứ giúp bạn trau dồi kỹ năng, nhưng nó có thể giúp bạn hiểu bối cảnh xung quanh 1 chủ đề, từ đó có thể giúp bạn có thêm động lực, & kỹ năng. Lịch sử AI & DRL sẽ giúp bạn đặt ra kỳ vọng về tương lai của công nghệ mạnh mẽ này. Đôi khi, cảm thấy sự cường điệu xung quanh AI thực sự hiệu quả; mọi người bắt đầu quan tâm. Nhưng ngay sau đó, khi đến lúc phải bắt tay vào thực hiện, sự cường điệu đó không còn hữu ích nữa, & đó là 1 vấn đề. Mặc dù tôi muốn hào hứng với AI, nhưng cũng cần đặt ra những kỳ vọng thực tế.

∗ **1.2.1. Recent history of AI & DRL.** History isn't necessary to gain skills, but it can allow you to understand context around a topic, which in turn can help you gain motivation, & therefore, skills. History of AI & DRL should help you set expectations about future of this powerful technology. At times, feel hyper surrounding AI is actually productive; people get interested. But, right after that, when it's time to put in work, hype no longer helps, & it's a problem. Although like to be excited about AI, also need to set realistic expectations.

– Lịch sử không nhất thiết phải là thứ giúp bạn trau dồi kỹ năng, nhưng nó có thể giúp bạn hiểu bối cảnh xung quanh 1 chủ đề, từ đó giúp bạn có thêm động lực, & từ đó, có thêm kỹ năng. Lịch sử AI & DRL sẽ giúp bạn đặt ra kỳ vọng về tương lai của công nghệ mạnh mẽ này. Đôi khi, bạn cảm thấy hào hứng với AI thực sự hiệu quả; mọi người bắt đầu quan tâm. Nhưng ngay sau đó, khi đến lúc phải bắt tay vào thực hiện, sự cường điệu không còn hiệu quả nữa, & đó chính là vấn đề. Mặc dù bạn muốn hào hứng với AI, nhưng cũng cần đặt ra những kỳ vọng thực tế.

· **Recent history of AI & DRL.** Beginnings of DRL could be traced back many years, because humans have been intrigued by possibility of intelligent creatures other than ourselves since antiquity. But a good beginning could be Alan Turing's work in 1930s, 1940s, & 1950s that paved way for modern CS & AI by laying down critical theoretical foundations that later scientists leveraged.

– Lịch sử gần đây của AI & DRL. Khởi đầu của DRL có thể được bắt nguồn từ nhiều năm trước, bởi vì con người đã bị hấp dẫn bởi khả năng tồn tại những sinh vật thông minh khác ngoài chúng ta từ thời cổ đại. Nhưng 1 khởi đầu tốt đẹp có thể là công trình của Alan Turing vào những năm 1930, 1940, & 1950, đã mở đường cho Khoa học Máy tính & AI hiện đại bằng cách đặt nền tảng lý thuyết quan trọng mà các nhà khoa học sau này đã tận dụng.

Most well-known of these is Turing Test, which proposes a standard for measuring machine intelligence: if a humane interrogator is unable to distinguish a machine from another human on a chat Q&A session, the computer is said to

count as intelligent. Though rudimentary, Turing Test allowed generations to wonder about possibilities of creating smart machines by setting a goal that researchers could pursue.
– Nổi tiếng nhất trong số này là Bài kiểm tra Turing, đề xuất 1 tiêu chuẩn để đo lường trí thông minh của máy móc: nếu 1 người hỏi không thể phân biệt được máy móc với người khác trong 1 buổi hỏi đáp trực tuyến, máy tính được coi là thông minh. Tuy còn thô sơ, Bài kiểm tra Turing đã cho phép nhiều thế hệ suy nghĩ về khả năng tạo ra máy móc thông minh bằng cách đặt ra 1 mục tiêu mà các nhà nghiên cứu có thể theo đuổi.
Formal beginnings of AI as an academic discipline can be attributed to John McCarthy, an influential AI researcher who made several notable contributions to field. To name a few, McCarthy is credited with coining term "AI" in 1955, leading 1st AI conference in 1956, inventing Lisp programming language in 1958, cofounding MIT AI Lab in 1959, & contributing important papers to development of AI as a field over several decades.
– Sự khởi đầu chính thức của AI như 1 ngành học thuật có thể được quy cho John McCarthy, 1 nhà nghiên cứu AI có ảnh hưởng, người đã có nhiều đóng góp đáng chú ý cho lĩnh vực này. Có thể kể đến 1 vài cái tên như McCarthy được ghi nhận là người đặt ra thuật ngữ "AI" vào năm 1955, chủ trì hội nghị AI đầu tiên vào năm 1956, phát minh ra ngôn ngữ lập trình Lisp vào năm 1958, đồng sáng lập Phòng thí nghiệm AI của MIT vào năm 1959, & đóng góp những bài báo quan trọng cho sự phát triển của AI như 1 lĩnh vực trong nhiều thập kỷ.

- **AI winters.** All work & progress of AI early on created a great deal of excitement, but there were also significant setbacks. Prominent AI researchers suggested we would create human-like machine intelligence within years, but this never came. Things got worst when a well-known researcher named James Lighthill compiled a report criticizing state of academic research in AI. All of these developments contributed to a long period of reduced funding & interest in AI research known as 1st *AI winter*.
– Mọi công trình & tiến bộ của AI thời kỳ đầu đều tạo nên sự phấn khích lớn, nhưng cũng có những trở ngại đáng kể. Các nhà nghiên cứu AI nổi tiếng đã đề xuất rằng chúng ta sẽ tạo ra trí tuệ máy móc giống con người trong vòng vài năm tới, nhưng điều này đã không bao giờ xảy ra. Mọi chuyện trở nên tồi tệ hơn khi 1 nhà nghiên cứu nổi tiếng tên là James Lighthill biên soạn 1 báo cáo chỉ trích tình trạng nghiên cứu học thuật về AI. Tất cả những phát triển này đã góp phần vào 1 giai đoạn dài giảm sút tài trợ & quan tâm đến nghiên cứu AI, được gọi là *AI winter* đầu tiên.
Field continued this pattern throughout years: researchers making progress, people getting overly optimistic, then overestimating – leading to reduced funding by government & industry partners.
– Field tiếp tục mô hình này trong nhiều năm: các nhà nghiên cứu đạt được tiến bộ, mọi người trở nên quá lạc quan, sau đó lại đánh giá quá cao – dẫn đến việc chính phủ & các đối tác trong ngành cắt giảm tài trợ.

- **Current state of AI.** We are likely in another highly optimistic time in AI history, so must be careful. Practitioners understand: AI is a powerful tool, but certain people think of AI as this magic black box that can take any problem in & out comes best solution ever. Nothing can be further from truth. Other people even worry about AI gaining consciousness, as if that was relevant, as Edsger W. Dijkstra famously said: "Question of whether a computer can think is no more interesting than question of whether a submarine can swim."
– Tình trạng hiện tại của AI. Chúng ta có thể đang ở trong 1 thời kỳ lạc quan khác trong lịch sử AI, vì vậy cần phải cẩn thận. Những người thực hành hiểu rằng: AI là 1 công cụ mạnh mẽ, nhưng 1 số người lại nghĩ về AI như 1 chiếc hộp đen ma thuật có thể xử lý bất kỳ vấn đề nào & đưa ra giải pháp tốt nhất từ trước đến nay. Không gì có thể xa rời sự thật hơn. Những người khác thậm chí còn lo lắng về việc AI sẽ đạt được ý thức, như thể điều đó có liên quan, như Edsger W. Dijkstra đã từng nói: "Câu hỏi liệu máy tính có thể suy nghĩ hay không cũng chẳng thú vị hơn câu hỏi liệu tàu ngầm có thể bơi hay không."
But, if set aside this Hollywood-instilled version of AI, can allow ourselves to get excited about recent progress in this field. Today, most influential companies in world make most substantial investments to AI research. Companies e.g. Google, Facebook, Microsoft, Amazon, & Apple have invested in AI research & have become highly profitable thanks, in part, to AI systems. Their significant & steady investments have created perfect environment for current pace of AI research. Contemporary researchers have best computing power available & tremendous amounts of data for their research, & teams of top researchers are working together, on same problems, in same location, at same time. Current AI research has become more stable & more productive. Have witnessed 1 AI success after another, & it doesn't seem likely to stop anytime soon.
– Nhưng nếu bỏ qua phiên bản AI do Hollywood đưa vào này, chúng ta có thể cho phép mình phấn khích về những tiến bộ gần đây trong lĩnh vực này. Ngày nay, các công ty có ảnh hưởng nhất trên thế giới thực hiện các khoản đầu tư đáng kể nhất vào nghiên cứu AI. Các công ty như Google, Facebook, Microsoft, Amazon, & Apple đã đầu tư vào nghiên cứu AI & đã trở nên cực kỳ có lợi nhuận, 1 phần là nhờ vào các hệ thống AI. Các khoản đầu tư & ổn định đáng kể của họ đã tạo ra môi trường hoàn hảo cho tốc độ nghiên cứu AI hiện tại. Các nhà nghiên cứu đương đại có sức mạnh tính toán tốt nhất hiện có & lượng dữ liệu khổng lồ cho nghiên cứu của họ, & các nhóm nghiên cứu hàng đầu đang làm việc cùng nhau, về cùng 1 vấn đề, tại cùng 1 địa điểm, cùng 1 thời điểm. Nghiên cứu AI hiện tại đã trở nên ổn định hơn & năng suất hơn. Đã chứng kiến thành công của AI này đến thành công khác, & dường như không có khả năng dừng lại trong thời gian tới.

- **Progress in DRL.** use of ANNs for RL problems started around 1990s. 1 of classics is backgammon-playing computer program, TD-Gammon, create by Gerald Tesauro et al. TD-Gammon learned to play backgammon by learning to evaluate table positions on its own through RL. Even though techniques implemented aren't precisely considered DRL, TD-Gammon was 1 of 1st widely reported success stories using ANNs to solve complex RL problems.
– Tiến bộ trong DRL. Việc sử dụng ANN cho các bài toán thực tế bắt đầu vào khoảng những năm 1990. 1 trong những ví dụ kinh điển là chương trình máy tính chơi cờ cá ngựa TD-Gammon, được tạo ra bởi Gerald Tesauro & cộng sự.

TD-Gammon học cách chơi cờ cá ngựa bằng cách tự đánh giá vị trí trên bàn cờ thông qua thực tế. Mặc dù các kỹ thuật được triển khai không được coi là DRL 1 cách chính xác, TD-Gammon là 1 trong những trường hợp thành công đầu tiên được báo cáo rộng rãi khi sử dụng ANN để giải các bài toán thực tế phức tạp.
Fig: TD-Gammon architecture. 1. Handcrafted features, not DL. 2. Not a "deep" network, but arguably beginnings of DRL. 3. Output of network was predicted probability of winning, given current game state.
– Hình: Kiến trúc TD-Gammon. 1. Các tính năng thủ công, không phải DL. 2. Không phải là mạng "sâu", nhưng có thể coi là khởi đầu của DRL. 3. Đầu ra của mạng là xác suất chiến thắng được dự đoán, dựa trên trạng thái trò chơi hiện tại.
In 2004, Andrew Ng et al. developed an autonomous helicopter that taught itself to fly stunts by observing hours of human-experts flights. They used a technique known as *inverse RL*, in which an agent learns from expert demonstrations. Same year, Nate Kohl & Peter Stone used a class of DRL methods known as *policy-gradient methods* to develop a soccer-playing robot for RoboCup tournament. They used RL to teach agent forward motion. After only 3 hours of training, robot achieved fastest forward-moving speed of any robot with same hardware.
– Năm 2004, Andrew Ng & cộng sự đã phát triển 1 máy bay trực thăng tự động có khả năng tự học cách bay nhào lộn bằng cách quan sát hàng giờ bay của các chuyên gia. Họ đã sử dụng 1 kỹ thuật được gọi là *inverse RL*, trong đó 1 tác nhân học hỏi từ các màn trình diễn của chuyên gia. Cùng năm đó, Nate Kohl & Peter Stone đã sử dụng 1 lớp phương pháp DRL được gọi là *policy-gradient methods* để phát triển 1 robot chơi bóng đá cho giải đấu RoboCup. Họ đã sử dụng RL để dạy tác nhân chuyển động về phía trước. Chỉ sau 3 giờ huấn luyện, robot đã đạt được tốc độ di chuyển về phía trước nhanh nhất so với bất kỳ robot nào có cùng phần cứng.
There were other successes in 2000s, but field of DRL really only started growing after DL field took off around 2010. In 2013 & 2015, Mnih et al. published a couple of papers presenting DQN algorithm. DQN learned to play Atari games from raw pixels. Using a CNN & a single set of hyperparameters, DQN performed better than a professional human player in 22 out of 49 games.
– Đã có những thành công khác vào những năm 2000, nhưng lĩnh vực DRL chỉ thực sự phát triển sau khi lĩnh vực DL cất cánh vào khoảng năm 2010. Năm 2013 & 2015, Mnih & cộng sự đã công bố 1 vài bài báo trình bày thuật toán DQN. DQN đã học cách chơi các trò chơi Atari từ các điểm ảnh thô. Sử dụng 1 mạng CNN & 1 bộ siêu tham số duy nhất, DQN đã hoạt động tốt hơn 1 người chơi chuyên nghiệp trong 22 trên 49 ván đấu.
Atari DQN network architecture. 1. Last 4 frames needed to infer velocities of ball, paddles, & so on. 2. Convolutions: Learned features through DL. 3. Feed-forward layers: Feed-forward ANN used learned features as inputs. 4. Output: Output layer return estimated expected value for each action.
– Kiến trúc mạng Atari DQN. 1. 4 khung hình cuối cùng cần thiết để suy ra vận tốc của bóng, mái chèo, v.v. 2. Tích chập: Các đặc điểm đã học thông qua DL. 3. Các lớp truyền thẳng: Mạng nơ-ron nhân tạo truyền thẳng sử dụng các đặc điểm đã học làm đầu vào. 4. Đầu ra: Lớp đầu ra trả về giá trị kỳ vọng ước tính cho mỗi hành động.
This accomplishment started a revolution in DRL community: In 2014, Silver et al. released deterministic policy gradient (DPQ) algorithm, & a year later Lillicrap et al. improved it with deep deterministic policy gradient (DDPG). In 2016, Schulman et al. released trust region policy optimization (TRPO) & generalized advantage estimation (GAE) methods, Sergey Levine et al. published Guided Policy Search (GPS), & Silver et al. demoed AlphaGo. Following year, Silver et al. demonstrated AlphaZero. Many other algorithms were released during these years: double deep Q-networks (DDQN), prioritized experience replay (PER), proximal policy optimization (PPO), actor-critic with experience replay (ACER), asynchronous advantage actor-critic (A3C), advantage actor-critic (A2C), actor-critic using Kronecker-factored trust region (ACKTR), Rainbow, Unicorn (these are actual names, BTW), & so on. In 2019, Oriol Vinyals et al. showed: AlphaStart agent beat professional players at game of StarCraft II. & a few months later, Jakub Pachocki et al. saw their team of Dota-2-playing bots, called Five, become 1st AI to beat world champions in an e-sports game.
– Thành tựu này đã bắt đầu 1 cuộc cách mạng trong cộng đồng DRL: Năm 2014, Silver & cộng sự đã phát hành thuật toán gradient chính sách xác định (DPQ), & 1 năm sau Lillicrap & cộng sự đã cải tiến nó bằng gradient chính sách xác định sâu (DDPG). Năm 2016, Schulman & cộng sự đã phát hành phương pháp tối ưu hóa chính sách vùng tin cậy (TRPO) & ước tính lợi thế tổng quát (GAE), Sergey Levine & cộng sự đã xuất bản Tìm kiếm chính sách có hướng dẫn (GPS), & Silver & cộng sự đã trình diễn AlphaGo. Năm sau, Silver & cộng sự đã trình diễn AlphaZero. Nhiều thuật toán khác đã được phát hành trong những năm này: mạng Q sâu kép (DDQN), phát lại kinh nghiệm ưu tiên (PER), tối ưu hóa chính sách gần (PPO), actor-critic với phát lại kinh nghiệm (ACER), actor-critic có lợi thế không đồng bộ (A3C), actor-critic có lợi thế (A2C), actor-critic sử dụng vùng tin cậy được xác định theo hệ số Kronecker (ACKTR), Rainbow, Unicorn (đây là tên thật, BTW), & vân vân. Năm 2019, Oriol Vinyals & cộng sự đã chứng minh: Dặc vụ AlphaStart đã đánh bại các game thủ chuyên nghiệp trong trò chơi StarCraft II. & vài tháng sau, Jakub Pachocki & cộng sự đã chứng kiến nhóm bot chơi Dota-2 của họ, có tên là Five, trở thành AI đầu tiên đánh bại các nhà vô địch thế giới trong 1 trò chơi thể thao điện tử.
Thanks to progress in DRL, have gone in 2 decades from solving backgammon, with its 10^{20} perfect-information states, to solving game of Go, with its 10^{170} perfect-information states, or better yet, to solving StarCraft II, with its 10^{270} imperfect-information states. Hard to conceive a better time to enter field. Can you imagine what next 2 decades will bring us? Will you be part of it? DRL is a booming field, & expect its rate of progress to continue.
– Nhờ những tiến bộ trong DRL, trong vòng 2 thập kỷ, chúng ta đã chuyển từ việc giải cờ thỏ cáo, với 10^{20} trạng thái thông tin hoàn hảo, sang giải cờ vây, với 10^{170} trạng thái thông tin hoàn hảo, hay thậm chí là giải StarCraft II, với 10^{270} trạng thái thông tin không hoàn hảo. Thật khó để hình dung ra thời điểm nào tốt hơn để bước vào lĩnh vực này. Bạn có thể tưởng tượng điều gì sẽ xảy ra trong 2 thập kỷ tới không? Bạn có muốn tham gia vào lĩnh vực này không? DRL là 1 lĩnh vực đang bùng nổ, & hãy kỳ vọng tốc độ tiến bộ của nó sẽ tiếp tục.

Fig. Game of Go: enormous branching factor. 1. From an empty board, there are many possible initial positions. 2. Out of each initial position, there are also many possible additional moves. 3. Branching continues until we have a total of 10^{127} states! That's more than number of atoms in observable universe.

– Hình. Cờ vây: hệ số phân nhánh cực lớn. 1. Từ 1 bàn cờ trống, có rất nhiều vị trí ban đầu khả dĩ. 2. Từ mỗi vị trí ban đầu, cũng có rất nhiều nước đi bổ sung khả dĩ. 3. Phân nhánh tiếp tục cho đến khi chúng ta có tổng cộng 10^{127} trạng thái! Con số này nhiều hơn số nguyên tử trong vũ trụ quan sát được.

· **Opportunities ahead.** Believe AI is a field with unlimited potential for positive change, regardless of what fear-mongers say. Back in 1750s, there was chaos due to start of industrial revolution. Powerful machines were replacing repetitive manual labor & mercilessly displacing humans. Everybody was concerned: machines that can work faster, more effectively, & more cheaply than humans? These machines will take all our jobs! What are we going to do for a living now? & it happened. But fact: many of these jobs were not only unfulfilling, but also dangerous.

– Cơ hội phía trước. Hãy tin rằng AI là 1 lĩnh vực có tiềm năng vô hạn cho những thay đổi tích cực, bất kể những kẻ gieo rắc nỗi sợ hãi nói gì. Trở lại những năm 1750, đã có sự hỗn loạn do cuộc cách mạng công nghiệp bắt đầu. Những cỗ máy mạnh mẽ đã thay thế lao động chân tay lặp đi lặp lại & thay thế con người 1 cách không thương tiếc. Ai cũng lo lắng: máy móc có thể làm việc nhanh hơn, hiệu quả hơn, & rẻ hơn con người? Những cỗ máy này sẽ chiếm hết việc làm của chúng ta! Bây giờ chúng ta sẽ làm gì để kiếm sống? & điều đó đã xảy ra. Nhưng sự thật: nhiều công việc trong số này không chỉ không thỏa mãn mà còn nguy hiểm.

100 years after industrial revolution, long-term effects of these changes were benefiting communities. People who usually owned only a couple of shirts & a pair of pants could get much more for a fraction of cost. Indeed, change was difficult, but long-term effects benefited entire world.

– 100 năm sau cuộc cách mạng công nghiệp, những tác động lâu dài của những thay đổi này đã mang lại lợi ích cho cộng đồng. Những người thường chỉ sở hữu 1 vài chiếc áo sơ mi & 1 chiếc quần dài có thể mua được nhiều hơn với chi phí chỉ bằng 1 phần nhỏ. Quả thực, thay đổi rất khó khăn, nhưng những tác động lâu dài đã mang lại lợi ích cho toàn thế giới.

Digital revolution started in 1970s with introduction of personal computers. Then, internet changed way we do things. Because of internet, got big data & cloud computing. ML used this fertile ground for sprouting into what it is today. In next couple of decades, changes & impact of AI on society may be difficult to accept at 1st, but long-lasting effects will be far superior to any setback along way. Expect in a few decades humans won't even need to work for food, clothing, or shelter because these things will be automatically produced by AI. Thrive with abundance.

– Cuộc cách mạng số bắt đầu vào những năm 1970 với sự ra đời của máy tính cá nhân. Sau đó, Internet đã thay đổi cách chúng ta làm việc. Nhờ Internet, dữ liệu lớn & điện toán đám mây ra đời. ML đã tận dụng mảnh đất màu mỡ này để phát triển thành những gì chúng ta thấy ngày nay. Trong vài thập kỷ tới, những thay đổi & tác động của AI lên xã hội có thể khó chấp nhận lúc đầu, nhưng những tác động lâu dài sẽ vượt trội hơn bất kỳ trở ngại nào trên chặng đường này. Dự kiến trong vài thập kỷ tới, con người thậm chí sẽ không cần phải làm việc để kiếm thức ăn, quần áo hay chỗ ở vì những thứ này sẽ được AI tự động tạo ra. Hãy phát triển thịnh vượng.

Workforce revolutions. mechanical engine, electricity, personal computer, AI?

As we continue to push intelligence of machines to higher levels, certain AI researchers think we might find an AI with intelligence superior to our own. At this point, unlock a phenomenon known as *singularity*; an AI more intelligent than humans allows for improvement of AI at a much faster pace, given that self-improvement cycle no longer has bottleneck, namely, humans. But we must be prudent, because this is more of an ideal than a practical aspect to worry about.

– Khi chúng ta tiếp tục thúc đẩy trí tuệ của máy móc lên những tầm cao mới, 1 số nhà nghiên cứu AI cho rằng chúng ta có thể tìm thấy 1 AI có trí tuệ vượt trội hơn chúng ta. Tại thời điểm này, hãy giải mã 1 hiện tượng được gọi là *singularity*; 1 AI thông minh hơn con người cho phép AI cải thiện với tốc độ nhanh hơn nhiều, vì chu trình tự cải thiện không còn bị giới hạn bởi con người nữa. Tuy nhiên, chúng ta phải thận trọng, bởi vì đây là 1 khía cạnh lý tưởng hơn là thực tế cần phải lo lắng.

Singularity could be a few decades away. More than me saying that singularity will happen, this graph is meant to explain what people refer to when they say "singularity".

While one must be always aware of implications of AI & strive for AI safety, singularity isn't an issue today. On other hand, many issues exist with current state of DRL. These issues make better use of our time.

– Mặc dù chúng ta phải luôn nhận thức được những tác động của AI & nỗ lực đảm bảo an toàn cho AI, nhưng tính kỳ dị không phải là vấn đề hiện nay. Mặt khác, vẫn còn nhiều vấn đề tồn tại với tình trạng DRL hiện tại. Những vấn đề này giúp chúng ta sử dụng thời gian hiệu quả hơn.

◦ **1.3. Suitability of DRL.** Could formulate any ML problem as a DRL problem, but this isn't always a good idea for multiple reasons. Should know pros & cons of using DRL in general, & you should be able to identify what kinds of problems & settings DRL is good & not so good for.

– Tính phù hợp của DRL. Có thể xây dựng bất kỳ bài toán ML nào thành bài toán DRL, nhưng điều này không phải lúc nào cũng tốt vì nhiều lý do. Bạn nên nắm rõ ưu & nhược điểm của việc sử dụng DRL nói chung, & xác định được loại vấn đề & cài đặt nào phù hợp & không phù hợp với DRL.

∗ **1.3.1. What are pros & cons?** Beyond a technological comparison, think about inherent advantages & disadvantages of using DRL for next project. See: each of points highlighted can be either a pro or a con depending on what kind of problem you're trying to solve. E.g., this field is about letting machine take control. Is this good or bad? Are you okay with letting computer make decisions for you? There's a reason why DRL research environments of choice are games: it could be costly & dangerous to have agents training directly in real world. Can you imagine a self-driving car agent

learning not to crash by crashing? In DRL, agents will have to make mistakes. Can you afford that? Are you willing to risk negative consequences – actual harm – to humans? Considered these questions before starting next DRL project.
– Ngoài việc so sánh về mặt công nghệ, hãy nghĩ về những ưu điểm & nhược điểm vốn có của việc sử dụng DRL cho dự án tiếp theo. Xem: mỗi điểm được nêu bật có thể là ưu điểm hoặc nhược điểm tùy thuộc vào loại vấn đề bạn đang cố gắng giải quyết. Ví dụ: lĩnh vực này là về việc để máy móc kiểm soát. Điều này tốt hay xấu? Bạn có đồng ý để máy tính đưa ra quyết định cho mình không? Có 1 lý do tại sao môi trường nghiên cứu DRL được lựa chọn là trò chơi: việc đào tạo các tác nhân trực tiếp trong thế giới thực có thể tốn kém & nguy hiểm. Bạn có thể tưởng tượng 1 tác nhân xe tự lái học cách không va chạm bằng cách va chạm không? Trong DRL, các tác nhân sẽ phải mắc lỗi. Bạn có đủ khả năng chi trả cho điều đó không? Bạn có sẵn sàng chấp nhận rủi ro về hậu quả tiêu cực – tác hại thực sự – đối với con người không? Đã xem xét những câu hỏi này trước khi bắt đầu dự án DRL tiếp theo.

DRL agents will explore! Can you afford mistakes?

Also need to consider how your agent will explore its environment. E.g., most value-based methods explore by randomly selecting an action. But other methods can have more strategic exploration strategies. Now, there are pros & cons to each, & this is a trade-off you will have to become familiar with.
– Ngoài ra, cần cân nhắc cách tác nhân của bạn sẽ khám phá môi trường của nó. Ví dụ, hầu hết các phương pháp dựa trên giá trị khám phá bằng cách chọn ngẫu nhiên 1 hành động. Tuy nhiên, các phương pháp khác có thể có các chiến lược khám phá mang tính chiến lược hơn. Mỗi phương pháp đều có ưu & nhược điểm riêng, & đây là 1 sự đánh đổi mà bạn sẽ phải làm quen.

Finally, training from scratch every time can be daunting, time consuming, & resource intensive. However, there are a couple of areas that study how to bootstrap previously acquired knowledge. 1st, there's *transfer learning*, which is about transferring knowledge gained in tasks to new ones. E.g., if you want to teach a robot to use a hammer & a screwdriver, you could reuse low-level actions learned on "pick up hammer" task & apply this knowledge to start learning "pick up screwdriver" task. This should make intuitive sense to you, because humans don't have to relearn low-level motions each time they learn a new task. Humans seem to form hierarchies of actions as we learn. Field of *hierarchical RL* tries to replicate this in DRL agents.
– Cuối cùng, việc đào tạo lại từ đầu mỗi lần có thể rất khó khăn, tốn thời gian & tài nguyên. Tuy nhiên, có 1 vài lĩnh vực nghiên cứu cách khởi động kiến thức đã học trước đó. Đầu tiên, có *chuyển giao học tập*, đó là về việc chuyển giao kiến thức thu được trong các nhiệm vụ sang nhiệm vụ mới. Ví dụ: nếu bạn muốn dạy 1 rô-bốt sử dụng búa & tua-vít, bạn có thể sử dụng lại các hành động cấp thấp đã học trong nhiệm vụ "nhấc búa" & áp dụng kiến thức này để bắt đầu học nhiệm vụ "nhấc tua-vít". Điều này có lẽ có ý nghĩa trực quan đối với bạn, vì con người không phải học lại các chuyển động cấp thấp mỗi khi họ học 1 nhiệm vụ mới. Con người dường như hình thành các hệ thống phân cấp hành động khi chúng ta học. Lĩnh vực *hierarchical RL* cố gắng sao chép điều này trong các tác nhân DRL.

* **DRL's strengths.** DRL is about mastering specific tasks. Unlike SL, in which generalization is goal, RL is good at concrete, well-specified tasks. E.g., each Atari game has a particular task. DRL agents aren't good at generalizing behavior across different tasks; it's not true that because train an agent to play Pong, this agent can also play Breakout. & if you naively try to teach your agent Pong & Breakout simultaneously, you will likely end up with an agent that isn't good at either. SL, on other hand, is pretty good a classifying multiple objects at once. Point is strength of DRL is well-defined single tasks.
– DRL liên quan đến việc thành thạo các nhiệm vụ cụ thể. Không giống như SL, trong đó mục tiêu là khái quát hóa, RL giỏi các nhiệm vụ cụ thể, được chỉ định rõ ràng. Ví dụ, mỗi trò chơi Atari đều có 1 nhiệm vụ cụ thể. Các tác nhân DRL không giỏi khái quát hóa hành vi trên các nhiệm vụ khác nhau; không đúng khi nói rằng chỉ cần huấn luyện 1 tác nhân chơi Pong thì tác nhân đó cũng có thể chơi Breakout. & nếu bạn ngây thơ cố gắng dạy tác nhân của mình chơi Pong & Breakout cùng lúc, rất có thể bạn sẽ có 1 tác nhân không giỏi cả hai. Mặt khác, SL khá giỏi trong việc phân loại nhiều đối tượng cùng 1 lúc. Điểm mạnh của DRL là các nhiệm vụ đơn lẻ được xác định rõ ràng.

In DRL, use generalization techniques to learn simple skills directly from raw sensory input. Performance of generalization techniques, new tips, & tricks on training deeper networks, & so on, are some of main improvements we have seen in recent years. Lucky for us, most DL advancements directly enable new research paths in DRL.
– Trong DRL, hãy sử dụng các kỹ thuật khái quát hóa để học các kỹ năng đơn giản trực tiếp từ dữ liệu cảm giác thô. Hiệu suất của các kỹ thuật khái quát hóa, mẹo mới, & thủ thuật đào tạo mạng lưới sâu hơn, & vân vân, là 1 số cải tiến chính mà chúng tôi đã thấy trong những năm gần đây. May mắn thay, hầu hết các tiến bộ của DL đều trực tiếp tạo ra các hướng nghiên cứu mới trong DRL.

* **DRL's weaknesses.** DRL isn't perfect. 1 of most significant issues you will find: in most problems, agents need millions of samples of learn well-performing policies. Humans, on other hand, can learn from a few interactions. Sample efficiency is probably 1 of top areas of DRL that could use improvements. We will touch on this topic in several chaps because it's a crucial one.
– Điểm yếu của DRL. DRL không hoàn hảo. 1 trong những vấn đề quan trọng nhất bạn sẽ gặp phải: trong hầu hết các bài toán, các tác nhân cần hàng triệu mẫu để học các chính sách hiệu suất cao. Mặt khác, con người có thể học hỏi từ 1 vài tương tác. Hiệu quả mẫu có lẽ là 1 trong những điểm mạnh nhất của DRL cần được cải thiện. Chúng ta sẽ đề cập đến chủ đề này trong 1 vài chương sau vì đây là 1 chủ đề quan trọng.

DRL agents need lots of interaction samples! Episode 2,324,532.

Another issue with DRL is with reward functions & understanding meaning of rewards. If a human expert will define rewards agent is trying to maximize, does that means that we are somewhat "supervising" this agent? & is this something good? Should reward be as dense as possible, which makes learning faster, or as sparse as possible, which makes solutions

more exciting & unique?

– 1 vấn đề khác với DRL là với các hàm phần thưởng & hiểu ý nghĩa của phần thưởng. Nếu 1 chuyên gia định nghĩa phần thưởng mà tác nhân đang cố gắng tối đa hóa, liệu điều đó có nghĩa là chúng ta đang "giám sát" tác nhân này không? & liệu đây có phải là điều tốt? Phần thưởng nên càng dày đặc càng tốt, giúp học nhanh hơn, hay càng thưa thớt càng tốt, giúp giải pháp thú vị hơn & độc đáo hơn?

We, as humans, don't seem to have explicitly defined rewards. Often, same person can see an event as positive or negative by simply changing their perspective. Additionally, a reward function for a task e.g. walking isn't straightforward to design. Is it forward motion that we should target, or is it falling? What is "perfect" reward function for a human walk?!

– Là con người, chúng ta dường như không có định nghĩa rõ ràng về phần thưởng. Thông thường, cùng 1 người có thể nhìn nhận 1 sự kiện là tích cực hay tiêu cực chỉ bằng cách thay đổi góc nhìn. Hơn nữa, việc thiết kế hàm thưởng cho 1 nhiệm vụ, ví dụ như đi bộ, không hề đơn giản. Chúng ta nên nhắm đến chuyển động về phía trước hay là chuyển động rơi? Hàm thưởng "hoàn hảo" cho 1 bước đi của con người là gì?!

There is ongoing interesting research on reward signals. One I'm particularly interested in is called *intrinsic motivation*. Intrinsic motivation allows agent to explore new actions just for sake of it, out of curiosity. Agents that use intrinsic motivation show improved learning performance in environments with sparse rewards, i.e., we get to keep exciting & unique solutions. Point is if you are trying to solve a task that hasn't been modeled or doesn't have a distinct reward function, you will face challenges.

– Hiện đang có những nghiên cứu thú vị về tín hiệu phần thưởng. 1 nghiên cứu mà tôi đặc biệt quan tâm được gọi là *động lực nội tại*. Động lực nội tại cho phép tác nhân khám phá những hành động mới chỉ vì thích, vì tò mò. Các tác nhân sử dụng động lực nội tại cho thấy hiệu suất học tập được cải thiện trong môi trường có ít phần thưởng, i.e., chúng ta có thể duy trì những giải pháp thú vị & độc đáo. Vấn đề là nếu bạn đang cố gắng giải quyết 1 nhiệm vụ chưa được mô hình hóa hoặc không có hàm phần thưởng riêng biệt, bạn sẽ gặp phải nhiều thách thức.

- **1.4. Setting clear 2-way expectations.** Touch on another important point going forward. What to expect? This is very important. 1st, know what to expect from book so there are no surprises later on. I don't want people to think that from this book, they will be able to come up with a trading agent that will make them rich. Sorry, I wouldn't be writing this book if it was that simple. Also expect: people who are looking to learn put in necessary work. Fact: learning will come from combination of me putting in effort to make concepts understandable & you putting in effort to understand them. I did put in effort. But, if you decide to skip a box you didn't think was necessary, we both lose.

  – Đặt kỳ vọng rõ ràng cho cả 2 bên. Nhắc đến 1 điểm quan trọng khác trong tương lai. Mong đợi điều gì? Điều này rất quan trọng. Trước tiên, hãy biết mình mong đợi điều gì từ cuốn sách để không có bất ngờ nào xảy ra sau này. Tôi không muốn mọi người nghĩ rằng từ cuốn sách này, họ sẽ có thể tìm ra 1 đại lý giao dịch giúp họ trở nên giàu có. Xin lỗi, tôi đã không viết cuốn sách này nếu nó đơn giản như vậy. Cũng mong đợi: những người muốn học hỏi sẽ phải bỏ công sức ra. Thực tế: việc học sẽ đến từ sự kết hợp giữa việc tôi nỗ lực để làm cho các khái niệm dễ hiểu & bạn nỗ lực để hiểu chúng. Tôi đã nỗ lực. Nhưng nếu bạn quyết định bỏ qua 1 ô mà bạn không nghĩ là cần thiết, thì cả 2 chúng ta đều thua.

  * **1.4.1. What to expect from book?** Goal for this book: take you, an ML enthusiast, from no prior DRL experience to capable of developing state-of-art DRL algorithms. For this, book is organized into roughly 2 parts. In Chaps. 3–7, learn about agents that can learn from sequential & evaluative feedback, 1st in isolation, & then in interplay. In Chaps. 8–12, dive into core DRL algorithms, methods, & techniques. Chaps. 1–2 are about introductory concepts applicable to DRL in general, & Chap. 13 has concluding remarks.

    – Mong đợi gì từ cuốn sách? Mục tiêu của cuốn sách này: đưa bạn, 1 người đam mê ML, từ chỗ chưa có kinh nghiệm DRL trước đó, đến khả năng phát triển các thuật toán DRL tiên tiến. Để làm được điều này, cuốn sách được chia thành khoảng 2 phần. Trong các Chương 3-7, hãy tìm hiểu về các tác nhân có thể học hỏi từ phản hồi đánh giá tuần tự, đầu tiên là từ sự cô lập, sau đó là từ sự tương tác. Trong các Chương 8-12, hãy đi sâu vào các thuật toán, phương pháp, kỹ thuật DRL cốt lõi. Các Chương 1-2 là về các khái niệm cơ bản áp dụng cho DRL nói chung, & Chương 13 là phần kết luận.

    Goal for 1st part (Chaps. 3–7: understand "tabular" RL. I.e., RL problems that can be exhaustively sampled, problems in which there's no need for neural networks or function approximation of any kind. Chap. 3 is abut sequential aspect of RL & temporal credit assignment problem. Then, we will study, also in isolation, challenge of learning from evaluative feedback & exploration vs. exploitation trade-off in Chap. 4. Last, learn about methods that can deal with these 2 challenges simultaneously. In Chap. 5, study agents that learn to estimate results of fixed behavior. Chap. 6 deals with learning to improve behavior, & Chap. 7 shows you techniques that make RL more effective & efficient.

    – Mục tiêu của phần 1 (Chương 3-7: hiểu RL "dạng bảng". I.e. các vấn đề RL có thể được lấy mẫu 1 cách cạn kiệt, các vấn đề mà không cần mạng nơ-ron hoặc xấp xỉ hàm dưới bất kỳ hình thức nào. Chương 3 nói về khía cạnh tuần tự của RL & vấn đề phân bổ tín dụng tạm thời. Sau đó, chúng ta sẽ nghiên cứu, cũng 1 cách riêng lẻ, thách thức của việc học từ phản hồi đánh giá & sự đánh đổi giữa khám phá & khai thác trong Chương 4. Cuối cùng, tìm hiểu về các phương pháp có thể giải quyết đồng thời 2 thách thức này. Trong Chương 5, nghiên cứu các tác nhân học cách ước tính kết quả của hành vi cố định. Chương 6 đề cập đến việc học để cải thiện hành vi, & Chương 7 chỉ cho bạn các kỹ thuật giúp RL hiệu quả hơn & hiệu suất cao hơn.

    Goal for 2nd part (Chaps. 8–12): grasp details of core DRL algorithms. Dive deep into details; can be sure of that. Learn about many different types of agents from value- & policy-based to actor-critic methods. In Chaps. 8–10, go deep into value-based DRL. In Chap. 11, learn about policy-based DRL & actor-critic, & Chap. 12 is about deterministic policy gradient (DPG) methods, soft actor-critic (SAC) & proximal policy optimization (PPO) methods.

    – Mục tiêu của phần 2 (Chương 8-12): nắm vững chi tiết các thuật toán DRL cốt lõi. Đi sâu vào chi tiết; có thể chắc chắn

về điều đó. Tìm hiểu về nhiều loại tác nhân khác nhau, từ phương pháp dựa trên giá trị & chính sách đến phương pháp tác nhân-phê bình. Trong Chương 8-10, hãy đi sâu vào DRL dựa trên giá trị. Trong Chương 11, hãy tìm hiểu về DRL dựa trên chính sách & tác nhân-phê bình, & Chương 12 là về các phương pháp gradient chính sách xác định (DPG), phương pháp tác nhân-phê bình mềm (SAC) & tối ưu hóa chính sách gần đúng (PPO).

Examples in these chapters are repeated throughout agents of the same type to make comparing & contrasting agents more accessible. Still explore fundamentally different kinds of problems, from small, continuous to image-based state spaces, & from discrete to continuous action spaces. But, book's focus isn't about modeling problems, which is a skill of its own; instead, focus is about solving already modeled environments.

– Các ví dụ trong các chương này được lặp lại xuyên suốt các tác nhân cùng loại để việc so sánh & đối chiếu các tác nhân trở nên dễ tiếp cận hơn. Vẫn khám phá các loại bài toán khác nhau về cơ bản, từ không gian trạng thái nhỏ, liên tục đến không gian trạng thái dựa trên ảnh, & từ không gian hành động rời rạc đến không gian hành động liên tục. Tuy nhiên, trọng tâm của cuốn sách không phải là về việc mô hình hóa các vấn đề, vốn là 1 kỹ năng riêng biệt; thay vào đó, trọng tâm là giải quyết các môi trường đã được mô hình hóa.

Comparison of different algorithmic approaches to DRL. In this book, learn about all these algorithmic approaches to DRL: derivative-free, policy-based, actor-critic, value-based, model-based. In fact, algorithms are focus & not so much problems. Why? Because in DRL, once you know algorithm, can apply that same algorithm to similar problems with hyperparameter tuning. Learning algorithms is where you get most out of your time.

– So sánh các phương pháp tiếp cận thuật toán khác nhau cho DRL. Trong cuốn sách này, bạn sẽ tìm hiểu về tất cả các phương pháp tiếp cận thuật toán cho DRL: không đạo hàm, dựa trên chính sách, dựa trên diễn viên-người phê bình, dựa trên giá trị, dựa trên mô hình. Trên thực tế, thuật toán là trọng tâm & không phải là vấn đề. Tại sao? Bởi vì trong DRL, 1 khi bạn đã biết thuật toán, bạn có thể áp dụng chính thuật toán đó cho các vấn đề tương tự với việc điều chỉnh siêu tham số. Học thuật toán là cách bạn tận dụng tối đa thời gian của mình.

∗ **1.4.2. How to get most out of this book.** There are a few things you need to bring to table to come out grokking DRL. Need to bring a little prior basic knowledge of ML & DL. Need to be comfortable with Python code & simple math. & most importantly, you must be willing to put in work.

– Cách tận dụng tối đa cuốn sách này. Có 1 vài điều bạn cần lưu ý để có thể hiểu rõ DRL. Cần có 1 chút kiến thức cơ bản về ML & DL. Cần thành thạo mã Python & toán học đơn giản. & quan trọng nhất, bạn phải sẵn sàng nỗ lực.

Assume: reader has a solid basic understanding of ML. Should know what Ml is beyond what's covered in this chap; should know how to train simple SL models, perhaps Iris or Titanic datasets; should be familiar with DL concepts e.g. tensors & matrices; & should have trained at least 1 DL model, say a CNN on MNIST dataset.

– Giả sử: người đọc có hiểu biết cơ bản vững chắc về ML. Cần biết ML là gì ngoài những gì được đề cập trong chương này; cần biết cách huấn luyện các mô hình SL đơn giản, có thể là các tập dữ liệu Iris hoặc Titanic; cần quen thuộc với các khái niệm DL, ví dụ như tenxơ & ma trận; cần đã huấn luyện ít nhất 1 mô hình DL, chẳng hạn như CNN trên tập dữ liệu MNIST.

This book is focused on DRL topics, & there's no DL in isolation. There are many useful resources out there that you can leverage. But, again, need a basic understanding; If you have trained a CNN before, then you're fine. Otherwise, highly recommend you follow a couple of DL tutorials before starting 2nd part of book.

– Cuốn sách này tập trung vào các chủ đề DRL, & không có DL riêng biệt. Có rất nhiều tài nguyên hữu ích mà bạn có thể tận dụng. Tuy nhiên, 1 lần nữa, bạn cần có kiến thức cơ bản; nếu bạn đã từng đào tạo CNN trước đây thì không sao. Nếu không, chúng tôi thực sự khuyên bạn nên làm theo 1 vài hướng dẫn DL trước khi bắt đầu phần 2 của sách.

Another assumption: reader is comfortable with Python code. Python is somewhat clear programming language that can be straightforward to understand, & people not familiar with it often get something out of merely reading it. Should be comfortable with it, willing & looking forward to reading code. If don't read code, then miss out on a lot.

– 1 giả định khác: người đọc cảm thấy thoải mái với mã Python. Python là ngôn ngữ lập trình khá rõ ràng, dễ hiểu, & những người chưa quen với nó thường học được điều gì đó chỉ bằng cách đọc nó. Nên cảm thấy thoải mái với nó, sẵn sàng & mong chờ được đọc mã. Nếu không đọc mã, bạn sẽ bỏ lỡ rất nhiều điều.

Likewise, there are many math equations in this book, & that's a good thing. Math is perfect language, & there's nothing that can replace it. However, I'm asking people to be comfortable with math, willing to read, & nothing else. Equations I show are heavily annotated so that people "not into math" can still take advantage of resources.

– Tương tự, cuốn sách này có rất nhiều phương trình toán học, & đó là 1 điều tốt. Toán học là ngôn ngữ hoàn hảo, & không gì có thể thay thế được. Tuy nhiên, tôi yêu cầu mọi người phải thoải mái với toán học, sẵn sàng đọc, & không gì khác. Các phương trình tôi trình bày được chú thích kỹ lưỡng để những người "không rành toán" vẫn có thể tận dụng các nguồn tài liệu.

Finally, assume you are willing to put in work. I.e., really want to learn DRL. If decide to skip math boxes, or Python snippets, or a section, or a page, or chap, or whatever, you will miss out on a lot of relevant information. To get most out of this book, recommend you read entire book front to back. Because of different format, figures & sidebars are part of main narrative in this book. Also, make sure you run book source code & play around & extend code you find most interesting.

– Cuối cùng, hãy giả sử bạn sẵn sàng bỏ công sức. I.e., thực sự muốn học DRL. Nếu quyết định bỏ qua các ô toán học, các đoạn mã Python, 1 phần, 1 trang, 1 chương, hay bất cứ điều gì, bạn sẽ bỏ lỡ rất nhiều thông tin liên quan. Để tận dụng tối đa cuốn sách này, tôi khuyên bạn nên đọc toàn bộ cuốn sách từ đầu đến cuối. Do định dạng khác nhau, các hình ảnh & thanh bên là 1 phần của nội dung chính trong cuốn sách này. Ngoài ra, hãy đảm bảo bạn chạy mã nguồn sách & thử nghiệm & mở rộng mã mà bạn thấy thú vị nhất.

* **DRL development environment.** Along with this book, you are provied with a fully tested environment & code to reproduce my results. I created a Docker image & several Jupyter Notebooks so that you don't have to mess around with installing packages & configuring software, or copying & pasting code. Only prerequisite is Docker. Follow directions at https://github.com/mimoralea/gdrl on running code: pretty straightforward.

  – Môi trường phát triển DRL. Cùng với cuốn sách này, bạn sẽ được cung cấp 1 môi trường đã được kiểm tra đầy đủ & mã để tái tạo kết quả của tôi. Tôi đã tạo 1 ảnh Docker & 1 số Jupyter Notebook để bạn không phải loay hoay cài đặt gói & cấu hình phần mềm, hay sao chép & dán mã. Điều kiện tiên quyết duy nhất là Docker. Làm theo hướng dẫn tại https://github.com/mimoralea/gdrl về cách chạy mã: khá đơn giản.

  Code is written in Python, & make use of NumPy & PyTorch. Chose PyTorch, instead of Keras, or TensorFlow, because found PyTorch to be a "Pythonic" library. Using PyTorch feels natural if have used NumPy, unlike TensorFlow, e.g., which feels like a whole new programming paradigm. My intention is not to start a "PyTorch vs. TensorFlow" debate. But, in my experience from using both libraries, PyTorch is a library much better suited for research & teaching.

  – Mã được viết bằng Python, & sử dụng NumPy & PyTorch. Tôi chọn PyTorch, thay vì Keras, hoặc TensorFlow, vì tôi thấy PyTorch là 1 thư viện "Pythonic". Sử dụng PyTorch cho cảm giác tự nhiên nếu đã từng sử dụng NumPy, không giống như TensorFlow, ví dụ, giống như 1 mô hình lập trình hoàn toàn mới. Ý định của tôi không phải là khơi mào 1 cuộc tranh luận "PyTorch so với TensorFlow". Nhưng, theo kinh nghiệm sử dụng cả 2 thư viện, PyTorch là 1 thư viện phù hợp hơn nhiều cho ngh

  DRL is about algorithms, methods, techniques, tricks, & so on, so it's pointless for us to rewrite a NumPy or a PyTorch library. But, also, in this book, write DRL algorithms from scratch; I'm not teaching you how to use a DRL library, e.g. Keras-RL, or Baselines, or RLlib. I want you to learn DRL, & therefore write DRL code. In years that I have been teaching RL, have noticed those who write RL code are more likely to understand RL. Now, this isn't a book on PyTorch either; there's no separate PyTorch review or anything like that, just PyTorch code that I explain as we move along. If you are somewhat familiar with DL concepts, you will be able to follow along with PyTorch code used in this book. Don't need a separate PyTorch resource before get to this book. Explain everything in detail as move along.

  – DRL là về các thuật toán, phương pháp, kỹ thuật, thủ thuật, & vân vân, nên việc chúng ta viết lại thư viện NumPy hay PyTorch là vô nghĩa. Nhưng, trong cuốn sách này, hãy viết các thuật toán DRL từ đầu; tôi không dạy bạn cách sử dụng thư viện DRL, ví dụ như Keras-RL, hay Baselines hay RLlib. Tôi muốn bạn học DRL, & do đó viết mã DRL. Trong những năm giảng dạy RL, tôi nhận thấy những người viết mã RL có nhiều khả năng hiểu RL hơn. Đây cũng không phải là 1 cuốn sách về PyTorch; không có bài đánh giá PyTorch riêng biệt hay bất cứ thứ gì tương tự, chỉ có mã PyTorch mà tôi giải thích khi chúng ta tiếp tục. Nếu bạn phần nào quen thuộc với các khái niệm DL, bạn sẽ có thể theo dõi mã PyTorch được sử dụng trong cuốn sách này. Không cần 1 tài nguyên PyTorch riêng biệt trước khi đến với cuốn sách này. Hãy giải thích mọi thứ chi tiết khi tiếp tục.

  As for environments we use for training agents, use popular OpenAI Gym package & a few other libraries that I developed for this book. But we are also not going into ins & outs of Gym. Just know: Gym is a library that provides environments for training RL agents. Beyond that, remember our focus is RL algorithms, solutions, not environments, or modeling problems, which, needless to say, are also critical.

  – Về môi trường chúng tôi sử dụng để huấn luyện các tác nhân, hãy sử dụng gói OpenAI Gym phổ biến & 1 vài thư viện khác mà tôi đã phát triển cho cuốn sách này. Tuy nhiên, chúng tôi cũng sẽ không đi sâu vào chi tiết & của Gym. Chỉ cần biết: Gym là 1 thư viện cung cấp môi trường để huấn luyện các tác nhân RL. Ngoài ra, hãy nhớ rằng trọng tâm của chúng tôi là các thuật toán RL, các giải pháp, chứ không phải môi trường, hay các bài toán mô hình hóa, vốn dĩ cũng rất quan trọng.

  Since you should be familiar with DL, presume you know what a graphics processing unit (GPU) is. DRL architectures don't need level of computation commonly seen on DL models. For this reason, use of a GPU, while a good thing, is not required. Conversely, unlike DL models, some DRL agents make heavy use of a central processing unit (CPU) & thread count. If you are planning on investing in a machine, make sure to account for CPU power (well, technically, number of codes, not speed) as well. Certain algorithms massively parallelize processing, & in those case, it's CPU that becomes bottleneck, not GPU. However, code runs fine in container regardless of your CPU or GPU. But, if your hardware is severely limited, recommend checking out cloud platforms. Have seen services, e.g. Google Colab, that offer DL hardware for free.

  – Vì bạn đã quen thuộc với DL, hãy cho rằng bạn biết bộ xử lý đồ họa (GPU) là gì. Kiến trúc DRL không cần mức độ tính toán thường thấy trên các mô hình DL. Vì lý do này, việc sử dụng GPU, mặc dù là 1 điều tốt, nhưng không phải là bắt buộc. Ngược lại, không giống như các mô hình DL, 1 số tác nhân DRL sử dụng nhiều bộ xử lý trung tâm (CPU) & số lượng luồng. Nếu bạn đang có kế hoạch đầu tư vào 1 máy, hãy đảm bảo tính đến cả sức mạnh của CPU (về mặt kỹ thuật là số lượng mã, không phải tốc độ). 1 số thuật toán xử lý song song hàng loạt & trong trường hợp đó, CPU trở thành nút thắt cổ chai, không phải GPU. Tuy nhiên, mã chạy tốt trong vùng chứa bất kể CPU hay GPU của bạn. Tuy nhiên, nếu phần cứng của bạn bị hạn chế nghiêm trọng, hãy khuyên bạn nên kiểm tra các nền tảng đám mây. Đã thấy các dịch vụ, ví dụ: Google Colab, cung cấp phần cứng DL miễn phí.

- **Summary.** DRL is challenging because agents must learn from feedback that is simultaneously sequential, evaluative, & sampled. Learning from sequential feedback forces agent to learn how to balance immediate & long-term goals. Learning from evaluative feedback makes agent learn to balance gathering & utilization of information. Learning from sampled feedback forces agent to generalize from old to new experiences.

  – DRL rất khó khăn vì các tác nhân phải học hỏi từ phản hồi vừa tuần tự, vừa đánh giá, vừa được lấy mẫu. Học từ phản hồi tuần tự buộc tác nhân phải học cách cân bằng giữa mục tiêu trước mắt & dài hạn. Học từ phản hồi đánh giá buộc tác

nhân phải học cách cân bằng giữa việc thu thập & sử dụng thông tin. Học từ phản hồi lấy mẫu buộc tác nhân phải khái quát hóa từ kinh nghiệm cũ sang kinh nghiệm mới.

AI, main field of CS into which RL falls, is a discipline concerned with creating computer programs that display human-like intelligence. This goal is shared across many other disciplines, e.g. control theory & operations research. ML is 1 of most popular & successful approaches to AI. RL is 1 of 3 branches of ML, along with supervised learning, & unsupervised learning. DL, an approach to ML, isn't tied to any specific branch, but its power helps advance entire ML community.

– Trí tuệ nhân tạo (AI), lĩnh vực chính của Khoa học Máy tính (CS) mà Học máy (RL) thuộc về, là 1 ngành học liên quan đến việc tạo ra các chương trình máy tính thể hiện trí thông minh giống con người. Mục tiêu này được chia sẻ trên nhiều ngành học khác, ví dụ như lý thuyết điều khiển & nghiên cứu vận hành. Học máy (ML) là 1 trong những phương pháp tiếp cận AI phổ biến & thành công nhất. Học máy (RL) là 1 trong 3 nhánh của Học máy, cùng với học có giám sát, & học không giám sát. Học máy (DL), 1 phương pháp tiếp cận ML, không bị ràng buộc với bất kỳ nhánh cụ thể nào, nhưng sức mạnh của nó giúp thúc đẩy toàn bộ cộng đồng ML.

DRL is use of multiple layers of powerful function approximators known as neural networks (DL) to solve complex sequential decision-making problems under uncertainty. DRL has performed well in many control problems, but, nevertheless, essential to have in mind: releasing human control for critical decision making shouldn't be taken lightly. Several off core needs in DRL are algorithms with better sample complexity, better-performing exploration strategies, & safe algorithms.

– DRL là việc sử dụng nhiều lớp hàm xấp xỉ mạnh mẽ, được gọi là mạng nơ-ron (DL), để giải quyết các bài toán ra quyết định tuần tự phức tạp trong điều kiện không chắc chắn. DRL đã hoạt động tốt trong nhiều bài toán điều khiển, nhưng điều quan trọng cần lưu ý là: việc giải phóng quyền kiểm soát của con người cho các quyết định quan trọng không nên được xem nhẹ. 1 số nhu cầu ngoài lõi của DRL bao gồm các thuật toán có độ phức tạp mẫu tốt hơn, các chiến lược khám phá hiệu suất cao hơn, & các thuật toán an toàn.

Still, future of DRL is bright, & there are perhaps dangers ahead as technology matures, but more importantly, there's potential in this field, & should feel excited & compelled to bring your best & embark on this journey. Opportunity to be part of a potential change this big happens only every few generations. Should be glad you are living during these times. Now, let's be part of it. By now:

* Understand what DRL is & how it compares with other ML approaches
* Are aware of recent progress in field of DRL, & intuitively understand that it has potential to be applied to a wide variety of problems
* Have a sense as to what to expect from this book, & how to get most out of it

– Tuy nhiên, tương lai của DRL vẫn tươi sáng, & có lẽ vẫn còn những nguy cơ phía trước khi công nghệ phát triển, nhưng quan trọng hơn, lĩnh vực này có tiềm năng, & nên cảm thấy hào hứng & thôi thúc bạn nỗ lực hết mình & bắt tay vào hành trình này. Cơ hội được tham gia vào 1 thay đổi lớn lao như thế này chỉ xảy ra sau vài thế hệ. Nên mừng vì bạn đang sống trong thời đại này. Giờ thì, hãy cùng tham gia nhé.

* Hiểu DRL là gì & so sánh nó với các phương pháp học máy khác
* Nhận thức được những tiến bộ gần đây trong lĩnh vực DRL, & trực giác hiểu rằng nó có tiềm năng ứng dụng cho nhiều vấn đề khác nhau
* Hiểu rõ những gì mong đợi từ cuốn sách này, & cách tận dụng tối đa nó

- **2. Mathematical foundations of RL.** Will learn: about core components of RL. Represent sequential decision-making problem as RL environments using a mathematical framework known as Markov decision processes. Build from scratch environments that RL agents learn to solve in later chaps.

  – Sẽ học: về các thành phần cốt lõi của RL. Biểu diễn bài toán ra quyết định tuần tự dưới dạng môi trường RL bằng cách sử dụng 1 khuôn khổ toán học được gọi là quy trình quyết định Markov. Xây dựng các môi trường từ đầu mà các tác nhân RL sẽ học cách giải quyết trong các chương sau.

> "Mankind's history has been a struggle against a hostile environment. We finally have reached a point where we can begin to dominate our environment . . . As soon as we understand this fact, our mathematical interests necessarily shift in many areas from descriptive analysis to control theory." – Richard Bellman, American applied mathematician, an IEEE medal of honor recipient
>
> – "Lịch sử nhân loại là 1 cuộc đấu tranh chống lại 1 môi trường thù địch. Cuối cùng, chúng ta đã đạt đến điểm mà chúng ta có thể bắt đầu thống trị môi trường của mình . . . Ngay khi chúng ta hiểu được sự thật này, mối quan tâm toán học của chúng ta chắc chắn sẽ chuyển từ phân tích mô tả sang lý thuyết điều khiển trong nhiều lĩnh vực." – Richard Bellman, nhà toán học ứng dụng người Mỹ, người nhận huy chương danh dự của IEEE

Pick up this book & decide to read 1 more chap despite having limited free time. A coach benches their best player for tonight's match ignoring press criticism. A parent invests long hours of hard work & unlimited patience in teaching their child good manners. These are all examples of complex sequential decision-making under uncertainty.

– Hãy cầm cuốn sách này lên & quyết định đọc thêm 1 chương nữa mặc dù thời gian rảnh có hạn. 1 huấn luyện viên cho cầu thủ giỏi nhất của mình ngồi dự bị trong trận đấu tối nay, bất chấp những lời chỉ trích của báo chí. 1 phụ huynh dành nhiều giờ làm việc chăm chỉ & kiên nhẫn vô hạn để dạy con cách cư xử đúng mực. Tất cả đều là những ví dụ về việc ra quyết định tuần tự phức tạp trong điều kiện không chắc chắn.

Want to bring to your attention 3 of words in play in this phrase: complex sequential decision-making under uncertainty. 1st word, *complex*, refers to fact that agents may be learning in environments with vast state & action spaces. In coaching example, even if you discover that your best player needs to rest every so often, perhaps resting them in a match with a specific opponent is better than with other opponents. Learning to generalize accurately is challenging because we learn from sampled feedback.

– Xin lưu ý 3 từ trong cụm từ này: ra quyết định tuần tự phức tạp trong điều kiện không chắc chắn. Từ đầu tiên, *complex*, ám chỉ thực tế là các tác nhân có thể đang học trong môi trường có không gian trạng thái & hành động rộng lớn. Trong ví dụ về huấn luyện, ngay cả khi bạn phát hiện ra rằng cầu thủ giỏi nhất của mình cần nghỉ ngơi thường xuyên, thì việc cho họ nghỉ ngơi trong 1 trận đấu với 1 đối thủ cụ thể có lẽ vẫn tốt hơn so với những đối thủ khác. Học cách khái quát hóa chính xác là 1 thách thức vì chúng ta học hỏi từ phản hồi được lấy mẫu.

2nd word I used is *sequential*, & this one refers to fact: in many problems, there are delayed consequences. In coaching example, again, say coach benched their best player for a seemingly unimportant match midway through season. But, what if action of resting players lowers their morale & performance that only manifests in finals? I.e., what if actual consequences are delayed? Fact: assigning credit to your past decisions is challenging because we learn from sequential feedback.

– Từ thứ hai tôi dùng là *sequence*, & từ này ám chỉ sự thật: trong nhiều vấn đề, có những hậu quả bị trì hoãn. Trong ví dụ về huấn luyện, 1 lần nữa, giả sử huấn luyện viên cho cầu thủ giỏi nhất của mình ngồi dự bị trong 1 trận đấu dường như không quan trọng vào giữa mùa giải. Nhưng, nếu việc cho cầu thủ nghỉ ngơi làm giảm tinh thần & hiệu suất của họ, điều này chỉ thể hiện rõ trong trận chung kết thì sao? I.e., nếu hậu quả thực tế bị trì hoãn thì sao? Sự thật: việc gán công lao cho các quyết định trong quá khứ của bạn là 1 thách thức vì chúng ta học hỏi từ phản hồi tuần tự.

Finally, word *uncertainty* refers to fact: don't know actual inner workings of world to understand how our actions affect it; everything is left to our interpretation. Say coach did bench their best player, but they got injured in next match. Was benching decision reason player got injured because player got out of shape? What if injury becomes a team motivation throughout season, & team ends up winning final? Again, was benching right decision? This uncertainty gives rise to need for exploration. Finding appropriate balance between exploration & exploitation is challenging because we learn from evaluative feedback.

– Cuối cùng, từ *uncertainty* ám chỉ sự thật: không biết hoạt động thực sự bên trong thế giới để hiểu hành động của chúng ta ảnh hưởng đến nó như thế nào; mọi thứ đều phụ thuộc vào cách chúng ta diễn giải. Giả sử huấn luyện viên đã cho cầu thủ giỏi nhất của mình ngồi dự bị, nhưng anh ta lại bị chấn thương trong trận đấu tiếp theo. Liệu quyết định cho cầu thủ ngồi dự bị có phải là lý do khiến cầu thủ bị chấn thương vì mất phong độ? Điều gì sẽ xảy ra nếu chấn thương trở thành động lực cho cả đội trong suốt mùa giải, & đội cuối cùng giành chiến thắng chung cuộc? 1 lần nữa, cho ngồi dự bị có phải là 1 quyết định đúng đắn? Sự không chắc chắn này làm nảy sinh nhu cầu khám phá. Việc tìm kiếm sự cân bằng phù hợp giữa khám phá & khai thác là 1 thách thức bởi vì chúng ta học hỏi từ phản hồi đánh giá.

In this chap, learn to represent these kinds of problems using a mathematical framework known as *Markov decision processes* (MDPs). General framework of MDPs allows us to model virtually any complex sequential decision-making problem under uncertainty in a way that RL agents can interact with & learn to solve solely through experience.

– Trong chương này, chúng ta sẽ học cách biểu diễn những loại bài toán này bằng 1 khuôn khổ toán học được gọi là *Markov Decision Process* (MDP). Khuôn khổ chung của MDP cho phép chúng ta mô hình hóa hầu như mọi vấn đề ra quyết định tuần tự phức tạp trong điều kiện không chắc chắn theo cách mà các tác nhân RL có thể tương tác & học cách giải quyết chỉ thông qua kinh nghiệm.

Dive deep into challenges of learning from sequential feedback in Chap. 3, then into challenges of learning from evaluative feedback in Chap. 4, then into challenges of learning from feedback that's simultaneously sequential & evaluative in Chaps. 5–7, & then Chaps. 8–12 will add complex into mix.

– Đi sâu vào những thách thức của việc học từ phản hồi tuần tự trong Chương 3, sau đó là những thách thức của việc học từ phản hồi đánh giá trong Chương 4, rồi đến những thách thức của việc học từ phản hồi vừa tuần tự & đánh giá trong Chương 5-7, & sau đó là Chương 8-12 sẽ bổ sung thêm sự phức tạp.

- **2.1. Components of RL.** 2 core components in RL are *agent & environment*. Agent is decision maker, & is solution to a problem. Environment is representation of a problem. 1 of fundamental distinctions of RL from other ML approaches: agent & environment interact; agent attempts to influence environment through actions, & environment reacts to agent's actions.

  – Các thành phần của RL. 2 thành phần cốt lõi trong RL là *tác nhân & môi trường*. Tác nhân là người ra quyết định, & là giải pháp cho 1 vấn đề. Môi trường là biểu diễn của 1 vấn đề. 1 trong những điểm khác biệt cơ bản của RL so với các phương pháp tiếp cận ML khác: tác nhân & môi trường tương tác; tác nhân cố gắng tác động đến môi trường thông qua hành động, & môi trường phản ứng với hành động của tác nhân.

  RL-interaction cycle. 1. Agent perceives environment. 2. Agent takes an action. 3. Environment goes through internal state change as a consequence of agent's action. 4. Environment reacts with new observation & a reward.

  Miguel's analogy. Parable of a Chinese farmer. There's an excellent parable that shows how difficult it is to interpret feedback that's simultaneously sequential, evaluative, & sampled. In life, it's challenging to know with certainty what are the long-term consequences of events & our actions. Often, find misfortune responsible for our later good fortune, or our good fortune responsible for our later misforune.

  – **Phép loại suy của Miguel.** Dụ ngôn về 1 người nông dân Trung Quốc. Có 1 dụ ngôn tuyệt vời cho thấy việc diễn giải phản hồi vừa tuần tự, vừa mang tính đánh giá, vừa mang tính lấy mẫu là khó khăn như thế nào. Trong cuộc sống, thật

khó để biết chắc chắn hậu quả lâu dài của các sự kiện & hành động của chúng ta là gì. Thường thì, ta thấy bất hạnh là nguyên nhân dẫn đến may mắn sau này, hoặc may mắn là nguyên nhân dẫn đến bất hạnh sau này.

Even though this story could be interpreted as a lesson that "beauty is in eye of beholder", RL, assume there's a correlation between actions we take & what happens in world. It's just that it's so complicated to understand these relationships, that it's difficult for humans to connect dots with certainty. But, perhaps this is something that computers can help us figure out. Exciting, right?

– Mặc dù câu chuyện này có thể được hiểu là 1 bài học rằng "cái đẹp nằm ở mắt người nhìn", nhưng hãy cứ cho là có mối tương quan giữa hành động của chúng ta & những gì xảy ra trên thế giới. Chỉ là việc hiểu được những mối quan hệ này quá phức tạp, đến nỗi con người khó có thể kết nối các sự kiện 1 cách chắc chắn. Nhưng có lẽ máy tính có thể giúp chúng ta tìm ra điều này. Thật thú vị phải không?

Having in mind that when feedback is simultaneously evaluative, sequential, & sampled, learning is a hard problem. & DRL is a computational approach to learning in these kinds of problems. Welcome to world of DRL.

– Cần lưu ý rằng khi phản hồi vừa mang tính đánh giá, tuần tự, & lấy mẫu, việc học tập sẽ là 1 vấn đề khó khăn. & DRL là 1 phương pháp tính toán để học tập trong những vấn đề như thế này. Chào mừng bạn đến với thế giới DRL.

∗ **2.1.1. Examples of problems, agents, & environments.** Following are abbreviated examples of RL problems, agents, environments, possible actions, & observations:

· Problem: You're training your dog to sit. Agent: Part of your brain that makes decisions. Environment: Your dog, treats, your dog's paws, loud neighbor, & so on. Actions: Talk to your dog. Wait for dog's reaction. Move your hand. Show treat. Give treat. Pet. Observations: Your dog is paying attention to you. Your dog is getting tired. Your dog is going away. Your dog sat on command.

· Problem: Your dog wants treat you have. Agent: Part of your dog's brain that makes decisions. Environment: You, treats, your dog's paws, loud neighbor, & so on. Actions: Stare at owner. Bark. Jump at owner. Try to steal treat. Run. Sit. Observations: Owner keeps talking loud at dog. Owner is showing treat. Owner is hiding treat. Owner gave dog treat.

· Problem: A trading agent investing in stock market. Agent: Executing DRL code in memory & in CPU. Environment: Your internet connection, machine the code is running on, stock prices, geopolitical uncertainty, other investors, day traders, & so on. Actions: Sell n stocks of y company. Buy n stocks of y company. Hold. Observations: Market is going up. Market is going down. There are economic tensions between 2 powerful nations. There's danger of war in continent. A global pandemic is wreaking havoc in entire world.

· Problem: You're driving your car. Agent: Part of your brain that makes decisions. Environment: Make & model of your car, other cars, other drivers, weather, roads, tires, & so on. Actions: Steer by x, accelerate by y. Break by z. Turn headlights on. Defog windows. Play music. Observations: You're approaching your destination. There's a traffic jam on Main Street. Car next to you is driving recklessly. It's starting to rain. There's a police officer driving in front of you.

– Sau đây là các ví dụ tóm tắt về các vấn đề thực tế, tác nhân, môi trường, hành động có thể xảy ra, & quan sát:

· Vấn đề: Bạn đang huấn luyện chó ngồi. Tác nhân: Phần não bộ đưa ra quyết định. Môi trường: Chó của bạn, đồ ăn vặt, bàn chân của chó, người hàng xóm ồn ào, & vân vân. Hành động: Nói chuyện với chó. Chờ phản ứng của chó. Di chuyển tay. Cho chó ăn đồ ăn vặt. Cho chó ăn đồ ăn vặt. Vuốt ve. Quan sát: Chó của bạn đang chú ý đến bạn. Chó của bạn đang mệt. Chó của bạn đang bỏ đi. Chó của bạn ngồi theo lệnh.

· Vấn đề: Chó của bạn muốn đồ ăn vặt mà bạn có. Tác nhân: Phần não bộ đưa ra quyết định của chó. Môi trường: Bạn, đồ ăn vặt, bàn chân của chó, người hàng xóm ồn ào, & vân vân. Hành động: Nhìn chằm chằm vào chủ. Sủa. Nhảy vào chủ. Cố gắng lấy trộm đồ ăn vặt. Chạy. Ngồi. Quan sát: Chủ liên tục nói to với chó. Chủ đang cho chó ăn đồ ăn vặt. Chủ đang giấu đồ ăn vặt. Chủ đã cho chó ăn đồ ăn vặt.

· Vấn đề: 1 đại lý giao dịch đầu tư vào thị trường chứng khoán. Đại lý: Thực thi mã DRL trong bộ nhớ & trong CPU. Môi trường: Kết nối internet của bạn, máy tính đang chạy mã, giá cổ phiếu, bất ổn địa chính trị, các nhà đầu tư khác, nhà giao dịch trong ngày, & vân vân. Hành động: Bán n cổ phiếu của công ty y. Mua n cổ phiếu của công ty y. Giữ. Quan sát: Thị trường đang tăng. Thị trường đang giảm. Có căng thẳng kinh tế giữa 2 quốc gia hùng mạnh. Có nguy cơ chiến tranh trên lục địa. 1 đại dịch toàn cầu đang tàn phá toàn thế giới.

· Vấn đề: Bạn đang lái xe. Đại lý: 1 phần não bộ của bạn đưa ra quyết định. Môi trường: Tạo & mô hình xe của bạn, những chiếc xe khác, những người lái xe khác, thời tiết, đường xá, lốp xe, & vân vân. Hành động: Lái theo x, tăng tốc theo y. Phanh theo z. Bật đèn pha. Làm tan sương cửa sổ. Phát nhạc. Quan sát: Bạn đang đến gần đích. Có 1 vụ tắc đường trên Phố Chính. Chiếc xe bên cạnh bạn đang lái xe ẩu. Trời bắt đầu mưa. Có 1 cảnh sát đang lái xe phía trước bạn.

As you can see, problems can take many forms: from high-level decision-making problems that require long-term thinking & broad general knowledge, e.g. investing in stock market, to low-level control problems, in which geopolitical tensions don't seem to play a direct role, e.g. driving a car.

– Như bạn thấy, vấn đề có thể xuất hiện dưới nhiều hình thức: từ các vấn đề ra quyết định cấp cao đòi hỏi tư duy dài hạn & kiến thức tổng quát rộng, ví dụ như đầu tư vào thị trường chứng khoán, đến các vấn đề kiểm soát cấp thấp, trong đó căng thẳng địa chính trị dường như không đóng vai trò trực tiếp, ví dụ như lái xe ô tô.

Also, can represent a problem from multiple agents' perspectives. In dog training example, in reality, there are 2 agents each interested in a different goal & trying to solve a different problem. Zoom into each of these components independently.

– Ngoài ra, có thể biểu diễn 1 vấn đề từ góc nhìn của nhiều tác nhân. Trong ví dụ huấn luyện chó, thực tế có 2 tác nhân, mỗi tác nhân quan tâm đến 1 mục tiêu khác nhau & cố gắng giải quyết 1 vấn đề khác nhau. Hãy phóng to từng thành phần này 1 cách độc lập.

* **Agent: Decision maker.** As mentioned in Chap. 1, this whole book is about agents, except for this chap, which is about environment. Starting with Chap. 3, dig deep into inner workings of agents, their components, their processes, & techniques to create agents that are effective & efficient.

– Tác nhân: Người ra quyết định. Như đã đề cập trong Chương 1, toàn bộ cuốn sách này nói về các tác nhân, ngoại trừ chương này, nói về môi trường. Bắt đầu từ Chương 3, hãy đi sâu vào hoạt động bên trong của các tác nhân, các thành phần, quy trình, & kỹ thuật của chúng để tạo ra các tác nhân hiệu quả & năng suất.

For now, only important thing for you to know what agents is that they are decision-makers in RL big picture. They have internal components & processes of their own, & that's what makes each of them unique & good at solving specific problems.

– Hiện tại, điều quan trọng duy nhất bạn cần biết về các tác nhân là họ là những người ra quyết định trong bức tranh toàn cảnh của thế giới thực. Họ có các thành phần nội tại & quy trình riêng, & đó là điều khiến mỗi tác nhân trở nên độc đáo & giỏi giải quyết các vấn đề cụ thể.

If we were to zoom in, could see: most agents have a 3-step process: all agents have an interaction component, a way to gather data for learning; all agents evaluate their current behavior; & all agents improve something in their inner components that allows them to improve (or at least attempt to improve) their overall performance.

– Nếu chúng ta phóng to, có thể thấy: hầu hết các tác nhân đều có quy trình 3 bước: tất cả các tác nhân đều có thành phần tương tác, 1 cách để thu thập dữ liệu để học; tất cả các tác nhân đều đánh giá hành vi hiện tại của mình; & tất cả các tác nhân đều cải thiện 1 điều gì đó trong các thành phần bên trong của chúng cho phép chúng cải thiện (hoặc ít nhất là cố gắng cải thiện) hiệu suất tổng thể của chúng.

3 internal steps that every RL agent goes through. 1. All agents evaluate their behavior. 2. RL means, well, agents have to learn something. 3. 1 of coolest things of RL is agents interact with problem.

– 3 các bước nội bộ mà mọi tác nhân thực tế đều trải qua. 1. Tất cả các tác nhân đều đánh giá hành vi của mình. 2. Thực tế ảo có nghĩa là các tác nhân phải học 1 điều gì đó. 3. 1 trong những điều thú vị nhất của thực tế ảo là các tác nhân tương tác với vấn đề.

Continue discussing inner workings of agents starting with next chap. For now, discuss a way to represent environments, how they look, & how we should model them, which is goal of this chap.

– Tiếp tục thảo luận về hoạt động bên trong của các tác nhân, bắt đầu từ chương tiếp theo. Trước tiên, hãy thảo luận về cách thể hiện môi trường, hình dạng của chúng, & cách chúng ta nên mô hình hóa chúng, đó là mục tiêu của chương này.

* **Environment: Everything else.** Most real-world decision-making problems can be expressed as RL environments. A common way to represent decision-making processes in RL is by modeling problem using a mathematical framework known as Markov decision processes (MDPs). In RL, assume all environments have an MDP working under hood. Whether an Atari game, stock market, a self-driving car, your significant other, you name it, every problem has an MDP running under hood (at least in RL world, whether right or wrong).

– Môi trường: Mọi thứ khác. Hầu hết các vấn đề ra quyết định trong thế giới thực đều có thể được diễn đạt dưới dạng môi trường thực tế. 1 cách phổ biến để biểu diễn các quy trình ra quyết định trong thế giới thực là mô hình hóa vấn đề bằng 1 khuôn khổ toán học được gọi là quy trình quyết định Markov (MDP). Trong thế giới thực, giả sử tất cả các môi trường đều có 1 MDP hoạt động ngầm. Cho dù là trò chơi Atari, thị trường chứng khoán, xe tự lái, người yêu của bạn, v.v., mọi vấn đề đều có 1 MDP hoạt động ngầm (ít nhất là trong thế giới thực, dù đúng hay sai).

Environment is represented by a set of variables related to problem. Combination of all possible values this set of variables can take is referred to as *state space*. A state is a specific set of values variables take at any given time.

– Môi trường được biểu diễn bằng 1 tập hợp các biến liên quan đến vấn đề. Tổ hợp tất cả các giá trị khả dĩ mà tập hợp các biến này có thể nhận được được gọi là *không gian trạng thái*. Trạng thái là 1 tập hợp các giá trị cụ thể mà các biến nhận được tại 1 thời điểm nhất định.

Agents may or may not have access to actual environment's state; however, 1 way or another, agents can observe something from environment. Set of variables agent perceives at any given time is called an *observation*.

– Các tác nhân có thể hoặc không thể truy cập vào trạng thái thực tế của môi trường; tuy nhiên, bằng cách này hay cách khác, các tác nhân có thể quan sát 1 điều gì đó từ môi trường. Tập hợp các biến mà tác nhân nhận thức được tại 1 thời điểm nhất định được gọi là 1 quan sát.

Combination of all possible values these variables can take is *observation space*. Know that state & observation are terms used interchangeably in RL community. This is because often agents are allowed to see internal state of environment, but this isn't always case. In this book, use state & observation interchangeably as well. But need to know: there might be a difference between states & observations, even though RL community often uses terms interchangeably.

– Tổ hợp tất cả các giá trị khả dĩ mà các biến này có thể nhận được là *không gian quan sát*. Cần lưu ý rằng trạng thái & quan sát là các thuật ngữ được sử dụng thay thế cho nhau trong cộng đồng RL. Điều này là do các tác nhân thường được phép nhìn thấy trạng thái bên trong của môi trường, nhưng điều này không phải lúc nào cũng đúng. Trong cuốn sách này, chúng tôi cũng sử dụng trạng thái & quan sát thay thế cho nhau. Tuy nhiên, cần lưu ý: có thể có sự khác biệt giữa trạng thái & quan sát, mặc dù cộng đồng RL thường sử dụng các thuật ngữ này thay thế cho nhau.

At every state, environment makes available a set of actions agent can choose from. Often set of actions is same for all states, but this isn't required. Set of all actions in all states is referred to as *action space*.

– Ở mỗi trạng thái, môi trường sẽ cung cấp 1 tập hợp các hành động mà tác nhân có thể lựa chọn. Thông thường, tập

hợp các hành động này giống nhau cho tất cả các trạng thái, nhưng điều này không bắt buộc. Tập hợp tất cả các hành động trong tất cả các trạng thái được gọi là không gian hành động.
Agent attempts to influence environment through these actions. Environment may change states as a response to agent's action. Function that is responsible for this transition is called *transition function.*
– Tác nhân cố gắng tác động đến môi trường thông qua những hành động này. Môi trường có thể thay đổi trạng thái để đáp lại hành động của tác nhân. Hàm chịu trách nhiệm cho quá trình chuyển đổi này được gọi là hàm chuyển tiếp.
After a transition, environment emits a new observation. Environment may also provide a reward signal as a response. Function responsible for this mapping is called *reward function.* Set of transition & reward function is referred to as *model* of environment.
– Sau 1 quá trình chuyển đổi, môi trường phát ra 1 quan sát mới. Môi trường cũng có thể cung cấp tín hiệu thưởng như 1 phản hồi. Hàm chịu trách nhiệm cho việc ánh xạ này được gọi là *hàm thưởng.* Tập hợp các quá trình chuyển đổi & hàm thưởng được gọi là *mô hình* của môi trường.

Example 1 (A concrete example: bandit walk environment)**.** *Make these concepts concrete with our 1st RL environment. I created this very simple environment for this book; called bandit walk (BW). BW is a simple* grid-world *(GW) environment. GWs are a common type of environment for studying RL algorithms that are grids of any size. GWs can have any model (transition & reward functions) you can think of & can make any kind of actions available.*
– Cụ thể hóa những khái niệm này bằng môi trường RL đầu tiên của chúng ta. Tôi đã tạo ra 1 môi trường rất đơn giản cho cuốn sách này; được gọi là bandit walk (BW). BW là 1 môi trường grid-world *(GW) đơn giản. GW là 1 loại môi trường phổ biến để nghiên cứu các thuật toán RL là các lưới có kích thước bất kỳ. GW có thể có bất kỳ mô hình nào (hàm chuyển tiếp & phần thưởng) mà bạn có thể nghĩ đến & có thể thực hiện bất kỳ loại hành động nào.*
But, they all commonly make move actions available to agent: Left, Down, Right, Up (or West, South, East, North, which is more precise because agent has no heading & usually has no visibility of full grid, but cardinal directions can also be more confusing). &, of course, each action corresponds with its logical transition: Left goes left, & Right goes right. Also, they all tend to have a fully observable discrete state & observation spaces (i.e., state equals observation) with integers representing cell id location of agent. A "walk" is a special case of grid-world environments with a single row. In reality, what I call a "walk' is more commonly referred to as a "Corridor". But, in this book, use term "walk" for all grid-world environments with a single row.
– Tuy nhiên, tất cả chúng đều thường cung cấp các hành động di chuyển cho tác nhân: Trái, Xuống, Phải, Lên (hoặc Tây, Nam, Đông, Bắc, chính xác hơn vì tác nhân không có tiêu đề & thường không hiển thị toàn bộ lưới, nhưng các hướng chính cũng có thể gây nhầm lẫn hơn). &, tất nhiên, mỗi hành động tương ứng với quá trình chuyển đổi hợp lý của nó: Trái sang trái, & Phải sang phải. Ngoài ra, tất cả chúng đều có xu hướng có trạng thái rời rạc có thể quan sát đầy đủ & không gian quan sát (i.e., trạng thái bằng quan sát) với các số nguyên biểu thị vị trí id ô của tác nhân. "Đi bộ" là trường hợp đặc biệt của môi trường thế giới lưới có 1 hàng. Trên thực tế, cái mà tôi gọi là "đi bộ" thường được gọi là "Hành lang". Tuy nhiên, trong cuốn sách này, hãy sử dụng thuật ngữ "đi bộ" cho tất cả các môi trường thế giới lưới có 1 hàng.
Bandit walk (BW) is a walk with 3 states, but only 1 non-terminal state. Environments that have a single non-terminal state are called "bandit" environments. "Bandit" here is an analogy to slot machines, which are also known as "1-armed bandits"; they have 1 arm &, if you like gambling, can empty your pockets, same way a bandit would.
– Bandit walk (BW) là 1 kiểu đi bộ có 3 trạng thái, nhưng chỉ có 1 trạng thái không kết thúc. Môi trường chỉ có 1 trạng thái không kết thúc được gọi là môi trường "bandit". "Bandit" ở đây là 1 phép so sánh với máy đánh bạc, còn được gọi là "máy đánh bạc 1 tay"; chúng chỉ có 1 tay &, nếu bạn thích đánh bạc, chúng có thể làm rỗng túi bạn, giống như cách 1 máy đánh bạc làm.
BW environment has just 2 actions available: a Left (action 0) & a Right (action 1) action. BW has a deterministic transition function: a Left action always moves agent to Left, & a Right action always moves agent to right. Reward signal is a $+1$ *when landing on rightmost cell,* 0 *otherwise. Agent starts in middle cell.*
– Môi trường BW chỉ có 2 hành động khả dụng: hành động Trái (hành động 0) & hành động Phải (hành động 1). BW có hàm chuyển tiếp xác định: hành động Trái luôn di chuyển tác nhân sang Trái, & hành động Phải luôn di chuyển tác nhân sang Phải. Tín hiệu thưởng là $a + 1$ *khi hạ cánh ở ô ngoài cùng bên phải,* 0 *nếu không. Tác nhân bắt đầu ở ô giữa.*
Bandit walk (BW) environment. *1. Agent starts in middle of walk. 2. Leftmost state is a hole. 3. Rightmost state is goal, & provides a* $+1$ *reward.*

A graphical representation of the BW environment would look like the following.
Bandit walk graph. 1. State 1 is a starting state. 2. Reward signal. 3. State 2 is a goal terminal state. 4. Transition of Right action is deterministic. 5. Action 1, Right. 6. Action 0, Left. 7. Transition of Left action is deterministic. 8. State 0 is a hole, a bad terminal state.
– Đồ thị di chuyển của Bandit. 1. Trạng thái 1 là trạng thái bắt đầu. 2. Tín hiệu thưởng. 3. Trạng thái 2 là trạng thái kết thúc mục tiêu. 4. Chuyển đổi của hành động sang Phải là xác định. 5. Hành động 1, Phải. 6. Hành động 0, Trái. 7. Chuyển đổi của hành động sang Trái là xác định. 8. Trạng thái 0 là 1 lỗ hổng, trạng thái kết thúc xấu.
Hope this raises several questions, but will find answers throughout this chap. E.g., why do terminal states have actions that transition to themselves: seems wasteful, doesn't? Any other questions? Like, what if environment is stochastic? What exactly is an environment that is "stochastic"?! Keep reading. Can also represent this environment in a table form. State|Action|Next state|Transition probability|Reward signal.
– Hy vọng điều này sẽ gợi ra 1 số câu hỏi, nhưng sẽ tìm thấy câu trả lời trong suốt chương này. Ví dụ, tại sao các trạng thái cuối cùng lại có các hành động chuyển tiếp sang chính nó: có vẻ lãng phí, phải không? Còn câu hỏi nào khác không? Ví dụ, nếu môi trường là ngẫu nhiên thì sao? Chính xác thì môi trường "ngẫu nhiên" là gì?! Đọc tiếp. Cũng có thể biểu

diễn môi trường này dưới dạng bảng. Trạng thái | Hành động | Trạng thái tiếp theo | Xác suất chuyển tiếp | Tín hiệu phần thưởng.

Example 2 (A concrete example: bandit slippery walk environment)**.** *So how about we make this environment stochastic? Say surface of walk is slippery & each action has a 20% chance of sending agent backwards. I call this environment bandit slipper walk (BSW). BSW is still a 1-row-grid world, a walk, a corridor, with only Left & Right actions available. Again, 3 states & 2 actions. Reward is same as before, +1 when landing at rightmost state (except when coming from rightmost state – from itself), & 0 otherwise.*

– Vậy thì chúng ta làm sao để biến môi trường này thành ngẫu nhiên? Giả sử bề mặt đường đi trơn trượt & mỗi hành động có 20% khả năng khiến tác nhân bị lùi lại. Tôi gọi môi trường này là "đi bộ kiểu bandit slipper walk" (BSW). BSW vẫn là 1 thế giới lưới 1 hàng, 1 đường đi, 1 hành lang, chỉ có các hành động Trái & Phải khả dụng. 1 lần nữa, 3 trạng thái & 2 hành động. Phần thưởng vẫn như trước, +1 khi hạ cánh ở trạng thái ngoài cùng bên phải (trừ khi đến từ trạng thái ngoài cùng bên phải – từ chính nó), & 0 nếu không.

However, transition function is different: 80% of time agent moves to intended cell, & 20% of time in opposite direction. A depiction of this environment would look as follows.

– Tuy nhiên, hàm chuyển tiếp lại khác: 80% thời gian, tác nhân di chuyển đến ô mong muốn & 20% thời gian theo hướng ngược lại. Mô tả môi trường này sẽ như sau.

Bandit slippery walk (BSW) environment. *1. Agent starts in middle of walk. 2. Rightmost state is goal, & provides a +1 reward. 3. Leftmost state is a hole. Identical to BW environment.*

– Môi trường Bandit Slipper Walk (BSW). *1. Đặc vụ bắt đầu ở giữa chặng đường. 2. Trạng thái ngoài cùng bên phải là mục tiêu, & cung cấp phần thưởng +1. 3. Trạng thái ngoài cùng bên trái là 1 cái hố. Giống hệt với môi trường BW.*

How do we know: action effects are stochastic? How do we represent "slippery" part of this problem? Graphical & table representations can help us with that. A graphical representation of the BSW environment would look like the following.

– Làm sao chúng ta biết: các hiệu ứng hành động là ngẫu nhiên? Làm thế nào để biểu diễn phần "trơn trượt" của bài toán này? Biểu diễn đồ họa & bảng có thể giúp chúng ta làm điều đó. Biểu diễn đồ họa của môi trường BSW sẽ trông như sau.

Bandit slippery walk graph. *1. Same as before: a hole, starting, & goal states. 2. But transition function is different! With an 80% chance, we move forward, & with a 20% chance, we move backward! See how transition function is different now? BSW environment has a stochastic transition function. Represent this environment in a table form as well.*

– Đồ thị bước đi trơn trượt Bandit. *1. Giống như trước: 1 lỗ, trạng thái bắt đầu, & mục tiêu. 2. Nhưng hàm chuyển đổi thì khác! Với xác suất 80%, chúng ta tiến về phía trước, & với xác suất 20%, chúng ta lùi về phía sau! Bây giờ bạn đã thấy hàm chuyển đổi khác biệt như thế nào chưa? Môi trường BSW có 1 hàm chuyển đổi ngẫu nhiên. Hãy biểu diễn môi trường này dưới dạng bảng.*

& we don't have to limit ourselves to thinking about environments with discrete state & action spaces or even walks (corridors) or bandits (which we discuss in-depth in next chap) or grid worlds. Representing environments as MDPs is a surprisingly powerful & straightforward approach to modeling complex sequential decision-making problems under uncertainty.

– & chúng ta không cần phải giới hạn bản thân trong việc suy nghĩ về các môi trường với không gian hành động & trạng thái rời rạc, hay thậm chí là các hành lang (hành lang) hay các khối (mà chúng ta sẽ thảo luận sâu hơn trong chương tiếp theo) hay các thế giới lưới. Việc biểu diễn môi trường dưới dạng MDP là 1 cách tiếp cận & trực quan đáng ngạc nhiên để mô hình hóa các vấn đề ra quyết định tuần tự phức tạp trong điều kiện không chắc chắn.

Here are a few more examples of environments that are powered by underlying MDPs.

1. **Hotter, colder.** Guess a randomly selected number using hints.
2. **Cart pole.** Balance a pole in a cart.
3. **Lunar lander.** Navigate a lander to its landing pad.
4. **Pong.** Bounce ball past opponent, & avoid letting ball pass you.
5. **Humanoid.** Make robot run as fast as possible & not fall.

Notice I didn't add transition function to this table. That's because, while you can look at code implementing dynamics for certain environments, other implementations are not easily accessible. E.g., transition function of cart pole environment is a small Python file defining mass of cart & pole & implementing basic physics equations, while dynamics of Atari games, e.g. Pong, are hidden inside an Atari emulator & corresponding game-specific ROM file.

– Dưới đây là 1 vài ví dụ khác về các môi trường được hỗ trợ bởi các MDP cơ bản.

1. **Nóng hơn, lạnh hơn.** Đoán 1 số được chọn ngẫu nhiên bằng cách sử dụng gợi ý.
2. **Cột xe đẩy.** Giữ thăng bằng 1 cột trong xe đẩy.
3. **Tàu đổ bộ Mặt Trăng.** Điều hướng tàu đổ bộ đến bệ hạ cánh.
4. **Pong.** Ném bóng qua đối thủ, & tránh để bóng vượt qua bạn.
5. **Hình người.** Làm cho robot chạy nhanh nhất có thể & không bị ngã.

Lưu ý rằng tôi đã không thêm hàm chuyển đổi vào bảng này. Đó là bởi vì, mặc dù bạn có thể xem mã triển khai động lực học cho 1 số môi trường nhất định, nhưng các triển khai khác không dễ truy cập. Ví dụ: hàm chuyển đổi của môi trường cột xe đẩy là 1 tệp Python nhỏ xác định khối lượng của xe đẩy & cột & triển khai các phương trình vật lý cơ bản, trong khi động lực học của các trò chơi Atari, ví dụ: Pong được ẩn bên trong trình giả lập Atari & tệp ROM tương ứng dành riêng cho trò chơi.

Notice: what we are trying to represent here is fact: environment "reacts" to agent's actions in some way, perhaps even by ignoring agent's actions. But at end of day, there is an internal process that is uncertain (except in this & next chap). To

represent ability to interact with an environment in an MDP, need states, observations, actions, a transition, & a reward function.

– Lưu ý: điều chúng ta đang cố gắng thể hiện ở đây là sự thật: môi trường "phản ứng" với hành động của tác nhân theo 1 cách nào đó, thậm chí có thể bằng cách bỏ qua hành động của tác nhân. Nhưng cuối cùng, có 1 quy trình nội tại chưa chắc chắn (trừ chương này & chương tiếp theo). Để thể hiện khả năng tương tác với môi trường trong MDP, cần có các trạng thái, quan sát, hành động, quá trình chuyển đổi, & hàm thưởng.

Process environment goes through as a consequence of agent's actions. 1. Environment receives action selected by agent. 2. Depending on current environment state, & agent's chosen action. 3. Environment will transition to a new internal state. 4. New state & reward are passed through a filter: some problem don"t let true state of environment be perceived by agent! 5. Finally, reaction is passed back to agent.

– Môi trường tiến trình được xử lý như 1 hệ quả từ hành động của tác nhân. 1. Môi trường nhận hành động do tác nhân lựa chọn. 2. Tùy thuộc vào trạng thái môi trường hiện tại, & hành động do tác nhân lựa chọn. 3. Môi trường sẽ chuyển sang trạng thái nội bộ mới. 4. Trạng thái & phần thưởng mới được chuyển qua bộ lọc: 1 số vấn đề không cho phép tác nhân nhận biết trạng thái thực sự của môi trường! 5. Cuối cùng, phản ứng được chuyển lại cho tác nhân.

* **Agent-environment interaction cycle.** Environment commonly has a well-defined task. Goal of this task is defined through reward signal. Reward signal can be dense, sparse, or anything in between. When design environments, reward signals are way to train your agent way you want. More dense, more supervision agent will have, & faster agent will learn, but more bias you will inject into your agent, & less likely agent will come up with unexpected behaviors. More sparse, less supervision, & therefore, higher chance of new, emerging behaviors, but the longer it will take agent to learn.

– Chu kỳ tương tác giữa tác nhân & môi trường. Môi trường thường có 1 nhiệm vụ được xác định rõ ràng. Mục tiêu của nhiệm vụ này được xác định thông qua tín hiệu thưởng. Tín hiệu thưởng có thể dày đặc, thưa thớt, hoặc bất kỳ mức độ nào ở giữa. Khi thiết kế môi trường, tín hiệu thưởng là cách để huấn luyện tác nhân theo cách bạn muốn. Càng dày đặc, tác nhân sẽ càng được giám sát nhiều hơn, & tác nhân sẽ học nhanh hơn, nhưng bạn sẽ tiêm nhiễm nhiều thiên kiến hơn vào tác nhân, & tác nhân ít có khả năng nảy sinh các hành vi bất ngờ. Càng thưa thớt, càng ít giám sát, & do đó, khả năng xuất hiện các hành vi mới, xuất hiện cao hơn, nhưng thời gian học của tác nhân sẽ lâu hơn.

Interactions between agent & environment go on for several cycles. Each cycle is called a *time step*. A time step is a unit of time, which can be a millisecond, a second, 1.2563 seconds, a minute, a day, or any other period of time.

– Tương tác giữa tác nhân & môi trường diễn ra trong nhiều chu kỳ. Mỗi chu kỳ được gọi là 1 *bước thời gian*. 1 bước thời gian là 1 đơn vị thời gian, có thể là 1 mili giây, 1 giây, 1,2563 giây, 1 phút, 1 ngày hoặc bất kỳ khoảng thời gian nào khác.

At each time step, agent observes environment, takes action, & receives a new observation & reward. Notice: even though rewards can be negative values, they are still called rewards in RL world. Set of observation (or state), action, reward, & new observation (or new state) is called an *experience tuple*.

– Tại mỗi bước thời gian, tác nhân quan sát môi trường, thực hiện hành động, & nhận được 1 quan sát mới & phần thưởng. Lưu ý: mặc dù phần thưởng có thể là giá trị âm, chúng vẫn được gọi là phần thưởng trong thế giới thực. Tập hợp quan sát (hoặc trạng thái), hành động, phần thưởng, & quan sát mới (hoặc trạng thái mới) được gọi là *bộ kinh nghiệm*.

Task agent is trying to solve may or may not have a natural ending. Tasks that have a natural ending, e.g. a game, are called *episodic tasks*. Tasks that don't, e.g. learning forward motion, are called *continuing tasks*. Sequence of time steps from beginning to end of an episode task is called an *episode*. Agents may make several time steps & episodes to learn to solve a task. Sum of rewards collected in a single episode is called a *return*. Agents are often designed to maximize return. A time step limit is often added to continuing tasks, so they become episodic tasks, & agents can maximize return.

– Nhiệm vụ mà tác nhân đang cố gắng giải quyết có thể có hoặc không có kết thúc tự nhiên. Các nhiệm vụ có kết thúc tự nhiên, ví dụ: trò chơi, được gọi là *nhiệm vụ theo từng tập*. Các nhiệm vụ không có, ví dụ: học chuyển động về phía trước, được gọi là *nhiệm vụ liên tục*. Chuỗi các bước thời gian từ đầu đến cuối của 1 nhiệm vụ theo từng tập được gọi là *tập*. Các tác nhân có thể thực hiện nhiều bước thời gian & tập để học cách giải quyết 1 nhiệm vụ. Tổng phần thưởng thu thập được trong 1 tập duy nhất được gọi là *lợi nhuận*. Các tác nhân thường được thiết kế để tối đa hóa lợi nhuận. Giới hạn bước thời gian thường được thêm vào các nhiệm vụ liên tục, vì vậy chúng trở thành nhiệm vụ theo từng tập, & các tác nhân có thể tối đa hóa lợi nhuận.

Every experience tuple has an opportunity for learning & improving performance. Agent may have 1 or more components to aid learning. Agent may be designed to learn mappings from observations to actions called policies. Agent may be designed to learn mappings from observations to new observations &/or rewards called models. Agent may be designed to learn mappings from observations (& possibly actions) to rewards-to-go estimates (A slice of return) called *value functions*.

– Mỗi bộ dữ liệu trải nghiệm đều có cơ hội học hỏi & cải thiện hiệu suất. Tác nhân có thể có 1 hoặc nhiều thành phần hỗ trợ việc học hỏi. Tác nhân có thể được thiết kế để học các ánh xạ từ quan sát đến hành động được gọi là chính sách. Tác nhân có thể được thiết kế để học các ánh xạ từ quan sát đến quan sát mới &/hoặc phần thưởng được gọi là mô hình. Tác nhân có thể được thiết kế để học các ánh xạ từ quan sát (& có thể là hành động) đến các ước tính phần thưởng (một phần lợi nhuận) được gọi là hàm giá trị.

For rest of this chap, put aside agent & interactions, & will examine environment & inner MDP in depth. In Chap. 3, pick back up agent, but there will be no interactions because agent won't need them as it will have access to MDPs. In Chap. 4, will remove agent's access to MDPs & add interactions back into equation, but it will be in single-state environments (bandits). Chap. 5 is about learning to estimate returns in multi-state environments when agents have no access to MDPs. Chaps. 6–7 are about optimizing behavior, which is full RL problem. Chaps. 5–7 are about agents learning in environments where there's no need for function approximation. After that, rest of book is all about agents that use neural networks for learning.

– Trong phần còn lại của chương này, hãy gác lại các tương tác của tác nhân &, & sẽ xem xét sâu về môi trường & MDP bên trong. Trong Chương 3, hãy chọn lại tác nhân, nhưng sẽ không có tương tác nào vì tác nhân sẽ không cần chúng vì nó sẽ có quyền truy cập vào MDP. Trong Chương 4, sẽ loại bỏ quyền truy cập của tác nhân vào MDP & thêm tương tác trở lại phương trình, nhưng sẽ ở trong môi trường 1 trạng thái (kẻ cướp). Chương 5 nói về việc học cách ước tính lợi nhuận trong môi trường nhiều trạng thái khi các tác nhân không có quyền truy cập vào MDP. Chương 6-7 nói về việc tối ưu hóa hành vi, đây là vấn đề RL đầy đủ. Chương 5-7 nói về việc các tác nhân học trong môi trường không cần xấp xỉ hàm. Sau đó, phần còn lại của cuốn sách sẽ nói về các tác nhân sử dụng mạng nơ-ron để học.

- **2.2. MDPs: Engine of Environment.** Build MDPs for a few environments as learn about components that make them up. Create Python dictionaries representing MDPs from descriptions of problems. In next chap, study algorithms for planning on MDPs. These methods can devise solutions to MDPs & will allow us to find optimal solutions to all problems in this chap.

  – MDP: Động cơ của Môi trường. Xây dựng MDP cho 1 số môi trường bằng cách tìm hiểu về các thành phần cấu thành chúng. Tạo các từ điển Python biểu diễn MDP từ mô tả bài toán. Trong chương tiếp theo, hãy nghiên cứu các thuật toán lập kế hoạch cho MDP. Các phương pháp này có thể đưa ra giải pháp cho MDP & sẽ cho phép chúng ta tìm ra giải pháp tối ưu cho tất cả các bài toán trong chương này.

  Ability to build environments yourself is an important skill to have. However, often find environments for which somebody else has already created MDP. Also, dynamics of environments are often hidden behind a simulation engine & are too complex to examine in detail; certain dynamics are even inaccessible & hidden behind real world. In reality, RL agents don't need to know precise MDP of a problem to learn robust behaviors, but knowing about MDPs, in general, is crucial for you because agents are commonly designed with assumption: an MDP, even if inaccessible, is running under hood.

  – Khả năng tự xây dựng môi trường là 1 kỹ năng quan trọng. Tuy nhiên, thường gặp phải những môi trường mà người khác đã tạo MDP. Hơn nữa, động lực của môi trường thường bị ẩn sau 1 công cụ mô phỏng & quá phức tạp để xem xét chi tiết; 1 số động lực thậm chí không thể tiếp cận & ẩn sau thế giới thực. Trên thực tế, các tác nhân RL không cần biết chính xác MDP của 1 vấn đề để học các hành vi mạnh mẽ, nhưng việc hiểu biết về MDP nói chung là rất quan trọng đối với bạn vì các tác nhân thường được thiết kế với giả định: 1 MDP, ngay cả khi không thể tiếp cận, vẫn đang chạy ngầm.

  Example 3 (Frozen lake environment.)**.** *This is another, more challenging problem for which we will build an MDP in this chap. This environment is called frozen lake (FL). FL is a simple grid-world (GW) environment. It also has discrete state & action spaces. However, this time, 4 actions are available: move Left, Down, Right, or Up. Task in FL environment is similar to task in BW & BSW environments: to go from a start location to a goal location while avoiding falling into holes. Challenges is similar to BSW, in that surface of FL environment is slippery, it is a frozen lake after all. But environment itself is larger. Look at a depiction of FL.*

  – Đây là 1 bài toán khác, khó hơn mà chúng ta sẽ xây dựng 1 MDP trong chương này. Môi trường này được gọi là hồ đóng băng (FL). FL là 1 môi trường thế giới lưới (GW) đơn giản. Nó cũng có các không gian trạng thái & hành động rời rạc. Tuy nhiên, lần này, có 4 hành động: di chuyển sang Trái, Xuống, Phải hoặc Lên. Nhiệm vụ trong môi trường FL tương tự như nhiệm vụ trong môi trường BW & BSW: đi từ vị trí bắt đầu đến vị trí đích trong khi tránh rơi xuống hố. Thử thách tương tự như BSW, ở chỗ bề mặt của môi trường FL trơn trượt, dù sao thì nó cũng là 1 hồ đóng băng. Nhưng bản thân môi trường thì lớn hơn. Hãy xem hình minh họa của FL.

  Frozen lake (FL) environment. *1. Agent starts each trial here. 2. Note: slippery, frozen surface may send agent to unintended places. 3. Agent gets a +1 when it arrives here. 4. But, these are holes that, if agent falls into, will end episode right away.*

  – Môi trường hồ đóng băng (FL). *1. Đặc vụ bắt đầu mỗi thử thách tại đây. 2. Lưu ý: bề mặt trơn trượt, đóng băng có thể khiến đặc vụ đến những nơi không mong muốn. 3. Đặc vụ nhận được +1 khi đến đây. 4. Tuy nhiên, đây là những cái hố mà nếu đặc vụ rơi vào, tập phim sẽ kết thúc ngay lập tức.*

  FL is a 4×4 *grid (it has 16 cells, ids 0–15). Agent shows up in START cell every new episode. Reaching GOAL cell gives a* $+1$ *reward; anything else is a 0. Because surface are slippery, agent moves only* $\frac{1}{3}$ *of time as intended. Other* $\frac{2}{3}$ *are split evenly in orthogonal directions. E.g., if agent chooses to move down, there's a 33.3% chance it moves down, 33.3% chance it moves left, & 33.3% chance it moves right. There's a fence around lake, so if agent tries to move out of grid world, it will bounce back to cell from which it tried to move. There are 4 holes in lake. If agent falls into 1 of these holes, it's game over. Ready to start building a representation of these dynamics? Need a Python dictionary representing MDP as described here. Start building MDP.*

  – FL là lưới 4×4 *(có 16 ô, id từ 0 đến 15). Đặc vụ xuất hiện trong ô BẮT ĐẦU mỗi tập mới. Đạt đến ô MỤC TIÊU sẽ nhận được phần thưởng* $+1$*; mọi thứ khác đều là 0. Vì bề mặt trơn trượt, đặc vụ chỉ di chuyển* $\frac{1}{3}$ *thời gian theo dự định. Các* $\frac{2}{3}$ *khác được chia đều theo các hướng trực giao. Ví dụ: nếu đặc vụ chọn di chuyển xuống, có 33,3% khả năng nó di chuyển xuống, 33,3% khả năng nó di chuyển sang trái, & 33,3% khả năng nó di chuyển sang phải. Có 1 hàng rào xung quanh hồ, vì vậy nếu đặc vụ cố gắng di chuyển ra khỏi thế giới lưới, nó sẽ nảy trở lại ô mà nó đã cố gắng di chuyển. Có 4 lỗ trong hồ. Nếu đặc vụ rơi vào 1 trong những lỗ này, trò chơi kết thúc. Sẵn sàng bắt đầu xây dựng biểu diễn của các động lực này không? Cần 1 từ điển Python biểu diễn MDP như được mô tả tại đây. Bắt đầu xây dựng MDP.*

- **States: Specific configurations of environment.** A *state* is a unique & self-contained configuration of problem. Set of all possible states, *state space*, is defined as set S. State space can be finite or infinite. But notice: state space is different than set of variables that compose a single state. This other set must always be finite & of constant size from state to state. In end, state space is a set of sets. Inner set must be of equal size & finite, as it contains number of variables representing states, but outer set can be infinite depending on types of elements of inner sets.

– Trạng thái: Cấu hình môi trường cụ thể. *state* là 1 cấu hình bài toán độc lập & duy nhất. Tập hợp tất cả các trạng thái khả dĩ, *state space*, được định nghĩa là tập S. Không gian trạng thái có thể hữu hạn hoặc vô hạn. Tuy nhiên, lưu ý: không gian trạng thái khác với tập hợp các biến tạo nên 1 trạng thái duy nhất. Tập hợp này phải luôn hữu hạn & với kích thước không đổi từ trạng thái này sang trạng thái khác. Cuối cùng, không gian trạng thái là 1 tập hợp các tập hợp. Tập hợp bên trong phải có kích thước bằng & hữu hạn, vì nó chứa số lượng biến biểu diễn các trạng thái, nhưng tập hợp bên ngoài có thể vô hạn tùy thuộc vào loại phần tử của các tập hợp bên trong.

State space: A set of sets. 1. Inner set (number of variables that compose states) must be finite. Size of inner set must be a positive integer. 2. But outer set may be infinite: e.g., if any of inner sets' elements is continuous.

– Không gian trạng thái: 1 tập hợp các tập hợp. 1. Tập hợp bên trong (số biến tạo nên các trạng thái) phải hữu hạn. Kích thước của tập hợp bên trong phải là 1 số nguyên dương. 2. Nhưng tập hợp bên ngoài có thể vô hạn: ví dụ, nếu bất kỳ phần tử nào của tập hợp bên trong là liên tục.

For BW, BSW, & FL environments, state is composed of a single variable containing id of cell where agent is at any given time. Agent's location cell is is a discrete variable. But state variables can be of any kind, & set of variables can be > 1. Could have Euclidean distance that would be a continuous variable & an infinite state space; e.g., 2.124, 2.12456, 5.1, 5.1239458, & so on. Could also have multiple variables defining state, e.g., number of cells away from goal in x- & y-axis. That would be 2 variables representing a single state. Both variables would be a discrete, therefore, state space finite. However, could also have variables of mixed types; e.g., one could be discrete, another continuous, another Boolean.

– Đối với môi trường BW, BSW, & FL, trạng thái bao gồm 1 biến duy nhất chứa id của ô nơi tác nhân ở tại bất kỳ thời điểm nào. Ô vị trí của tác nhân là 1 biến rời rạc. Nhưng các biến trạng thái có thể là bất kỳ loại nào, & tập hợp các biến có thể > 1. Có thể có khoảng cách Euclidean sẽ là 1 biến liên tục & 1 không gian trạng thái vô hạn; ví dụ: 2,124, 2,12456, 5,1, 5,1239458, & vân vân. Cũng có thể có nhiều biến xác định trạng thái, ví dụ: số ô cách mục tiêu theo trục x- & y. Đó sẽ là 2 biến biểu diễn 1 trạng thái duy nhất. Cả hai biến sẽ là 1 biến rời rạc, do đó, không gian trạng thái hữu hạn. Tuy nhiên, cũng có thể có các biến có kiểu hỗn hợp; ví dụ: 1 biến có thể là rời rạc, 1 biến khác là liên tục, 1 biến khác là Boolean.

With this state representation for BW, BSW, & FL environments, size of state space is 3, 3, & 16, resp. Given we have 3, 3, or 16 cells, agent can be at any given time, then have 3, 3, & 16 possible states in state space. Can set ids of each cell starting from 0, going left to right, top to bottom.

– Với biểu diễn trạng thái này cho các môi trường BW, BSW, & FL, kích thước của không gian trạng thái tương ứng là 3, 3, & 16. Giả sử chúng ta có 3, 3 hoặc 16 ô, tại bất kỳ thời điểm nào, tác nhân có thể có 3, 3, & 16 trạng thái khả dĩ trong không gian trạng thái. Có thể thiết lập id của mỗi ô bắt đầu từ 0, từ trái sang phải, từ trên xuống dưới.

In FL, set ids from 0 to 15, left to right, top to bottom. Could set ids in any other way: in a random order, or group cells by proximity, or whatever. It's up to you; as long as you keep them consistent throughout training, it will work. However, this representation is adequate, & it works well, so it's what we will use.

– Trong FL, hãy đặt id từ 0 đến 15, từ trái sang phải, từ trên xuống dưới. Bạn có thể đặt id theo bất kỳ cách nào khác: theo thứ tự ngẫu nhiên, hoặc nhóm các ô theo khoảng cách, hoặc bất cứ cách nào khác. Tùy bạn; miễn là bạn giữ chúng nhất quán trong suốt quá trình huấn luyện, thì nó sẽ hoạt động. Tuy nhiên, cách biểu diễn này là đủ, & nó hoạt động tốt, vì vậy chúng ta sẽ sử dụng nó.

States in FL contain a single variable indicating id of cell in which agent is at any given time step. – Các trạng thái trong FL chứa 1 biến duy nhất chỉ ra ID của ô mà tác nhân đang ở tại bất kỳ bước thời gian nào.

In case of MDPs, states are fully observable: can see internal state of environment at each time step, i.e., observations & states are same. *Partially observable Markov decision processes* (POMDPs) is a more general framework for modeling environments in which observations, which still depend on internal state of environment, are only things agent can see instead of state. Notice: for BW, BSW, & FL environments, we are creating an MDP, so agent will be able to observe internal state of environment.

– Trong trường hợp MDP, các trạng thái có thể quan sát được hoàn toàn: có thể thấy trạng thái nội tại của môi trường tại mỗi bước thời gian, i.e., các quan sát & trạng thái là như nhau. *Partially observable Markov Decision Process* (POMDP) là 1 khuôn khổ tổng quát hơn để mô hình hóa các môi trường, trong đó các quan sát, vẫn phụ thuộc vào trạng thái nội tại của môi trường, chỉ là những thứ mà tác nhân có thể thấy thay vì trạng thái. Lưu ý: đối với các môi trường BW, BSW, & FL, chúng tôi đang tạo 1 MDP, do đó tác nhân sẽ có thể quan sát trạng thái nội tại của môi trường.

States must contain all variables necessary to make them independent of all other states. In FL environment, only need to know current state of agent to tell its next possible states. I.e., you don't need history of states visited by agent for anything. You know that from state 2 agent can only transition to states 1, 3, 6, or 2, & this is true regardless of whether agent's previous state was 1, 3, 6, or 2.

– Trạng thái phải chứa tất cả các biến cần thiết để chúng độc lập với tất cả các trạng thái khác. Trong môi trường FL, chỉ cần biết trạng thái hiện tại của tác nhân là có thể biết các trạng thái tiếp theo của nó. I.e., bạn không cần lịch sử các trạng thái mà tác nhân đã ghé thăm. Bạn biết rằng từ trạng thái 2, tác nhân chỉ có thể chuyển sang trạng thái 1, 3, 6 hoặc 2, & điều này đúng bất kể trạng thái trước đó của tác nhân là 1, 3, 6 hay 2.

Probability of next state, given current state & action, is independent of history of interactions. This memoryless property of MDPs is known as *Markov property*: probability of moving from 1 state s to another state s on 2 separate occasions, given same action a, is same regardless of all previous states or actions encountered before that point.

– Xác suất của trạng thái tiếp theo, với trạng thái hiện tại & hành động, không phụ thuộc vào lịch sử tương tác. Tính chất không nhớ này của MDP được gọi là *Markov property*: xác suất di chuyển từ trạng thái s này sang trạng thái s khác trong

2 lần riêng biệt, với cùng 1 hành động a, là như nhau bất kể tất cả các trạng thái hoặc hành động trước đó đã gặp phải trước điểm đó.

Markov property. Probability of next state given current state & current action $P(S_{t+1}|S_t, A_t)$ will be same $P(S_{t+1}|S_t, A_t, S_{t-1}, A$ as if you give it entire history of interactions.

– **Thuộc tính Markov.** Xác suất của trạng thái tiếp theo khi biết trạng thái hiện tại & hành động hiện tại $P(S_{t+1}|S_t, A_t)$ sẽ giống $P(S_{t+1}|S_t, A_t, S_{t-1}, A_{t-1}, \ldots)$ như thể bạn cung cấp cho nó toàn bộ lịch sử tương tác.

But why do you care about this? Well, in environments we have explored so far it is not that obvious, & it is not that important. But because most RL & DRL agents are designed to take advantage of Markov assumption, you must make sure you feed your agent necessary variables to make it hold as tightly as possible (completely keeping Markov assumption is impractical, perhaps impossible).

– Nhưng tại sao bạn lại quan tâm đến điều này? Trong các môi trường chúng ta đã khám phá cho đến nay, điều này không quá rõ ràng, & nó cũng không quá quan trọng. Nhưng vì hầu hết các tác nhân RL & DRL được thiết kế để tận dụng giả định Markov, bạn phải đảm bảo cung cấp cho tác nhân các biến cần thiết để nó bám chặt nhất có thể (việc giữ nguyên hoàn toàn giả định Markov là không thực tế, thậm chí là không thể).

E.g., if you are designing an agent to learn to land a spacecraft, agent must receive all variables that indicate velocities along with its locations. Locations alone are not sufficient to land a spacecraft safely, & because you must assume agent is memoryless, need to feed agent more information than just its x, y, z coordinates away from landing pad.

– Ví dụ, nếu bạn đang thiết kế 1 tác nhân để học cách hạ cánh tàu vũ trụ, tác nhân đó phải nhận được tất cả các biến số biểu thị vận tốc cùng với vị trí của nó. Chỉ riêng vị trí là không đủ để hạ cánh tàu vũ trụ an toàn, & vì bạn phải giả định rằng tác nhân không có trí nhớ, cần cung cấp cho tác nhân nhiều thông tin hơn là chỉ các tọa độ x, y, z của nó khi hạ cánh.

But, you probably know that acceleration is to velocity what velocity is to position: derivative. Probably also know that you can keep taking derivatives beyond acceleration. To make MDP completely Markovian, how deep do you have to go? This is more of an art than a science: more variables you add, longer it takes to train an agent, but fewer variables, higher chance information fed to agent is not sufficient, & harder it is to learn anything useful. For spacecraft example, often locations & velocities are adequate, & for grid-world environments, only state id location of agent is sufficient.

– Nhưng, có lẽ bạn đã biết rằng gia tốc so với vận tốc cũng giống như vận tốc so với vị trí: đạo hàm. Có lẽ bạn cũng biết rằng bạn có thể tiếp tục lấy đạo hàm vượt quá gia tốc. Để biến MDP hoàn toàn theo mô hình Markovian, bạn phải đi sâu đến mức nào? Đây là 1 nghệ thuật hơn là khoa học: bạn thêm nhiều biến, thời gian đào tạo tác nhân sẽ lâu hơn, nhưng ít biến hơn, thông tin có xác suất cao hơn được cung cấp cho tác nhân là không đủ, & khó học được bất cứ điều gì hữu ích. Ví dụ như đối với tàu vũ trụ, vị trí & vận tốc thường là đủ, & đối với môi trường thế giới lưới, chỉ cần ID trạng thái vị trí của tác nhân là đủ.

Set of all states in MDP is denoted S^+. There is a subset of S^+ called set of *starting* or *initial states*, denoted S^i. To begin interacting with an MDP, draw a state from S^i from a probability distribution. This distribution can be anything, but it must be fixed throughout training: i.e., probabilities must be same from 1st to last episode of training & for agent evaluation.

– Tập hợp tất cả các trạng thái trong MDP được ký hiệu là S^+. Có 1 tập con của S^+ được gọi là tập hợp *là trạng thái bắt đầu* hoặc *là trạng thái ban đầu*, ký hiệu là S^i. Để bắt đầu tương tác với MDP, hãy vẽ 1 trạng thái từ S^i từ 1 phân phối xác suất. Phân phối này có thể là bất kỳ giá trị nào, nhưng nó phải được cố định trong suốt quá trình huấn luyện: i.e., xác suất phải giống nhau từ tập huấn luyện đầu tiên đến tập huấn luyện cuối cùng & để đánh giá tác nhân.

There's a unique state called *absorbing* or *terminal state*, & set of all non-terminal states is denoted S. Now, while it's common practice to create a single terminal state (a sink state) to which all terminal transitions go, this is not always implemented this way. What you see more often is multiple terminal states, & that is okay. It doesn't really matter under hood if you make all terminal states behave as expected.

– Có 1 trạng thái duy nhất được gọi là *hấp thụ* hoặc *trạng thái kết thúc*, & tập hợp tất cả các trạng thái không kết thúc được ký hiệu là S. Mặc dù thông lệ chung là tạo 1 trạng thái kết thúc duy nhất (trạng thái chìm) mà tất cả các chuyển tiếp kết thúc đều hướng đến, nhưng điều này không phải lúc nào cũng được triển khai theo cách này. Điều bạn thường thấy là nhiều trạng thái kết thúc, & điều đó không sao cả. Việc bạn làm cho tất cả các trạng thái kết thúc hoạt động như mong đợi không thực sự quan trọng.

As expected? Yes. A terminal state is a special state: it must have all available actions transitioning, with probability 1, to itself, & these transitions must provide no reward. Note: I'm referring to transitions from terminal state, not to terminal state.

– Như mong đợi? Đúng vậy. Trạng thái kết thúc là 1 trạng thái đặc biệt: nó phải có tất cả các hành động khả dụng chuyển đổi sang chính nó, với xác suất 1, & những chuyển đổi này không được mang lại bất kỳ phần thưởng nào. Lưu ý: Tôi đang nói đến các chuyển đổi từ trạng thái kết thúc, chứ không phải đến trạng thái kết thúc.

It's very commonly case that end of an episode provides a nonzero reward. E.g., in a chess game you win, you lose, or you draw. A logical reward signal would be +1, -1, & 0, resp. But it is a compatibility convention that allows for all algorithms to converge to same solution to make all actions available in a terminal state transition from that terminal state to itself with probability 1 & reward 0. Otherwise, run risk of infinite sums & algorithms that may not work altogether. Remember how BW & BSW environments had these terminal states?

– Rất thường xảy ra trường hợp phần cuối của 1 tập phim cung cấp phần thưởng khác không. Ví dụ, trong 1 ván cờ vua, bạn thắng, thua hoặc hòa. 1 tín hiệu phần thưởng logic sẽ là +1, -1, & 0, tương ứng. Nhưng đó là 1 quy ước tương thích cho phép tất cả các thuật toán hội tụ về cùng 1 giải pháp để làm cho tất cả các hành động có thể thực hiện trong trạng

thái kết thúc chuyển từ trạng thái kết thúc đó sang chính nó với xác suất 1 & phần thưởng 0. Nếu không, sẽ có nguy cơ tạo ra tổng vô hạn & các thuật toán có thể không hoạt động hoàn toàn. Bạn còn nhớ môi trường BW & BSW đã có những trạng thái kết thúc này như thế nào không?

In FL environment, e.g., there's only 1 starting state (which is state 0) & 5 terminal states (or 5 states that transition to a single terminal state, whichever you prefer). For clarity, use convention of multiple terminal states (5, 7, 11, 12, & 15) for illustrations & code; again, each terminal state is a separate terminal state.

– Ví dụ, trong môi trường FL, chỉ có 1 trạng thái bắt đầu (là trạng thái 0) & 5 trạng thái kết thúc (hoặc 5 trạng thái chuyển tiếp thành 1 trạng thái kết thúc duy nhất, tùy theo bạn thích). Để rõ ràng hơn, hãy sử dụng quy ước về nhiều trạng thái kết thúc (5, 7, 11, 12, & 15) để minh họa & mã; 1 lần nữa, mỗi trạng thái kết thúc là 1 trạng thái kết thúc riêng biệt.

States in frozen lake environment. There is 1 initial state & 5 terminal states.

- **Actions: A mechanism to influence environment.** MDPs make available a set of actions A that depends on state. I.e., there might be actions that aren't allowed in a state – in fact, A is a function that takes a state as an argument, i.e., $A(s)$. This function returns set of available actions for state s. If needed, can define this set to be constant across state space; i.e., all actions are available at every state. Can also set all transitions from a state-action pair to 0 if you want to deny an action in a given state. Could also set all transitions from state s & action a to same state s to denote action a as a no-intervene or no-op action.

  – Hành động: 1 cơ chế tác động đến môi trường. MDP cung cấp 1 tập hợp các hành động A phụ thuộc vào trạng thái. I.e., có thể có những hành động không được phép trong 1 trạng thái – thực tế, A là 1 hàm lấy 1 trạng thái làm đối số, i.e., $A(s)$. Hàm này trả về tập hợp các hành động khả dụng cho trạng thái s. Nếu cần, có thể định nghĩa tập hợp này là hằng số trên không gian trạng thái; i.e., tất cả các hành động đều khả dụng ở mọi trạng thái. Cũng có thể đặt tất cả các chuyển đổi từ 1 cặp trạng thái-hành động thành 0 nếu bạn muốn từ chối 1 hành động trong 1 trạng thái nhất định. Cũng có thể đặt tất cả các chuyển đổi từ trạng thái s & hành động a thành cùng 1 trạng thái s để biểu thị hành động a là hành động không can thiệp hoặc không thao tác.

  Just as with state, action space may be finite or infinite, & set of variables of a single action may contain > 1 element & must be finite. However, unlike number of state variables, number of variables that compose an action may not be constant. Actions available in a state may change depending on that state. For simplicity, most environments are designed with same number of actions in all states.

  – Cũng giống như trạng thái, không gian hành động có thể hữu hạn hoặc vô hạn, & tập hợp các biến của 1 hành động đơn lẻ có thể chứa > 1 phần tử & phải hữu hạn. Tuy nhiên, không giống như số lượng biến trạng thái, số lượng biến tạo nên 1 hành động có thể không cố định. Các hành động khả dụng trong 1 trạng thái có thể thay đổi tùy thuộc vào trạng thái đó. Để đơn giản, hầu hết các môi trường đều được thiết kế với số lượng hành động như nhau ở tất cả các trạng thái.

  Environment makes set of all available actions known in advance. Agents can select actions either deterministically or stochastically. This is different than saying environment reacts deterministically or stochastically to agents' actions. Both are true statements, but I am referring here to fact: agents can either select actions from a lookup table or from per-state probability distributions.

  – Môi trường làm cho tập hợp tất cả các hành động khả dụng được biết trước. Các tác nhân có thể chọn hành động theo cách xác định hoặc ngẫu nhiên. Điều này khác với việc nói rằng môi trường phản ứng theo cách xác định hoặc ngẫu nhiên với hành động của tác nhân. Cả hai đều đúng, nhưng ở đây tôi đang nói đến thực tế: các tác nhân có thể chọn hành động từ bảng tra cứu hoặc từ phân phối xác suất theo trạng thái.

  In BW, BSW, & FL environments, actions are singletons representing direction agent will attempt to move. In FL, there are 4 available actions in all states: Up, Down, Right, or Left. There is 1 variable per action, & size of action space is 4.

  – Trong môi trường BW, BSW, & FL, các hành động là các singleton đại diện cho hướng mà tác nhân sẽ cố gắng di chuyển. Trong FL, có 4 hành động khả dụng ở tất cả các trạng thái: Lên, Xuống, Phải hoặc Trái. Mỗi hành động có 1 biến, & kích thước không gian hành động là 4.

  Frozen lake environment has 4 simple move actions. 1. Actions: Up, Down, Left, Right. 2. From now on, draw terminal states without actions for simplicity. 3. But have in mind: terminal states are defined as states with all actions with deterministic transitions to themselves.

  – Môi trường hồ đóng băng có 4 hành động di chuyển đơn giản. 1. Hành động: Lên, Xuống, Trái, Phải. 2. Từ bây giờ, hãy vẽ các trạng thái kết thúc mà không có hành động để đơn giản hơn. 3. Nhưng hãy nhớ: các trạng thái kết thúc được định nghĩa là các trạng thái có tất cả các hành động với các chuyển đổi xác định cho chính chúng.

- **Transition function: Consequences of agent actions.** Way environment changes as a response to actions is referred to as *state-transition probabilities*, or more simply, *transition function*, & is deoted by $T(s, a, a')$. Transition function T maps a transition tuple s, a, s' to a probability; i.e., you pass in a state s, an action a, & a next state s', & it will return corresponding probability of transition from state s to state s' when taking action a. Could also represent it as $T(s, a)$ & return a dictionary with next states for its keys & probabilities for its values.

  – Hàm chuyển tiếp: Hậu quả của hành động tác nhân. Cách môi trường thay đổi như 1 phản ứng với các hành động được gọi là *xác suất chuyển tiếp trạng thái*, hay đơn giản hơn là *hàm chuyển tiếp*, & được ký hiệu là $T(s, a, a')$. Hàm chuyển tiếp T ánh xạ 1 bộ chuyển tiếp s, a, s' thành 1 xác suất; i.e., bạn truyền vào trạng thái s, 1 hành động a, & trạng thái tiếp theo s', & nó sẽ trả về xác suất chuyển tiếp tương ứng từ trạng thái s sang trạng thái s' khi thực hiện hành động a. Cũng có thể biểu diễn nó là $T(s, a)$ & trả về 1 từ điển với các trạng thái tiếp theo cho các khóa của nó & xác suất cho các giá trị của nó.s

Notice T also describes a probability distribution $p(\cdot|s,a)$ determining how system will evolve in an interaction cycle from selecting action a in state s. When integrating over next state s', as any probability distribution, sum of these probabilities must equal 1.

– Lưu ý T cũng mô tả 1 phân phối xác suất $p(\cdot|s,a)$ xác định cách hệ thống sẽ tiến hóa trong 1 chu kỳ tương tác từ việc chọn hành động a ở trạng thái s. Khi tích phân trên trạng thái tiếp theo s', như bất kỳ phân phối xác suất nào, tổng của các xác suất này phải bằng 1.

Transition function. Transition function is defined as probability of transitioning to state s' at time step t given action a was selected on state s in previous time step $t-1$: $p(s'|s,a) = P(S_t = s'|S_{t-1} = s, A_{t-1} = a)$. Given these are probabilities, expect sum of probabilities across all possible next states to sum to 1: $\sum_{s' \in S} p(s'|s,a) = 1$, $\forall s \in S$, $\forall a \in A(s)$. That is true for all states s in set of states s, & all actions a in set of actions available in state s.

– **Hàm chuyển tiếp.** Hàm chuyển tiếp được định nghĩa là xác suất chuyển sang trạng thái s' tại bước thời gian t với hành động a được chọn trên trạng thái s tại bước thời gian trước đó $t-1$: $p(s'|s,a) = P(S_t = s'|S_{t-1} = s, A_{t-1} = a)$. Với các xác suất này, kỳ vọng tổng xác suất trên tất cả các trạng thái tiếp theo có thể bằng 1: $\sum_{s' \in S} p(s'|s,a) = 1$, $\forall s \in S$, $\forall a \in A(s)$. Điều đó đúng với mọi trạng thái s trong tập các trạng thái s, & mọi hành động a trong tập các hành động khả dụng trong trạng thái s.

BW environment was deterministic; i.e., probability of next state s' given current state s & action a was always 1. There was always a single possible next state s'. BSW & FL environments are stochastic, i.e., probability of next state s' given current state s & action a is < 1. There are > 1 possible next state s's.

– Môi trường BW mang tính xác định; i.e., xác suất trạng thái tiếp theo s' với trạng thái hiện tại s & hành động a luôn bằng 1. Luôn có 1 trạng thái tiếp theo s' khả thi. Môi trường BSW & FL mang tính ngẫu nhiên, i.e., xác suất trạng thái tiếp theo s' với trạng thái hiện tại s & hành động a là < 1. Có > 1 trạng thái tiếp theo s' khả thi.

1 key assumption of many RL (& DRL) algorithm: this distribution is stationary. I.e., while there may be highly stochastic transitions, probability distribution may not change during training or evaluation. Just as with Markov assumption, stationarity assumption is often relaxed to an extent. However, important for most agents to interact with environments that at least appear to be stationary.

– 1 giả định chính của nhiều thuật toán RL (& DRL): phân phối này là dừng. Nghĩa là, mặc dù có thể có các chuyển đổi ngẫu nhiên cao, phân phối xác suất có thể không thay đổi trong quá trình huấn luyện hoặc đánh giá. Cũng giống như giả định Markov, giả định tính dừng thường được nới lỏng ở 1 mức độ nào đó. Tuy nhiên, điều quan trọng đối với hầu hết các tác nhân là tương tác với các môi trường ít nhất có vẻ dừng.

In FL environment, know: there is a 33.3% chance we will transition to intended cell (state) & a 66.6% chance we will transition to orthogonal directions. There is also a chance we will bounce back to state we are coming from if it is next to wall.

– Trong môi trường FL, hãy biết rằng: có 33,3% khả năng chúng ta sẽ chuyển sang ô (trạng thái) mong muốn & 66,6% khả năng chúng ta sẽ chuyển sang các hướng trực giao. Cũng có khả năng chúng ta sẽ bật trở lại trạng thái ban đầu nếu nó nằm cạnh tường.

For simplicity & clarity, have added to following image only transition functions for all actions of states 0, 2, 5, 7, 11, 12, 13, & 15 of FL environment. This subset of states allows for illustration of all possible transitions without too much clutter.

– Để đơn giản & rõ ràng hơn, chúng tôi đã thêm vào hình ảnh sau các hàm chuyển tiếp chỉ dành cho tất cả các hành động ở trạng thái 0, 2, 5, 7, 11, 12, 13, & 15 của môi trường FL. Tập hợp con các trạng thái này cho phép minh họa tất cả các chuyển tiếp có thể xảy ra mà không quá rườm rà.

Transition function of frozen lake environment. 1. Without probabilities for clarity. 2. Notice corner states are special. You bounce back from horizontal & vertical walls. 3. Remember: terminal states have all transitions from all actions looping back to themselves with probability 1. 5. This environment is highly stochastic!

– Hàm chuyển tiếp của môi trường hồ đóng băng. 1. Không có xác suất để làm rõ. 2. Lưu ý rằng các trạng thái góc là đặc biệt. Bạn bật lại từ các bức tường ngang & dọc. 3. Hãy nhớ: các trạng thái cuối cùng có tất cả các chuyển tiếp từ mọi hành động lặp lại với chính chúng với xác suất 1. 5. Môi trường này rất ngẫu nhiên!

It might still be a bit confusing, but look at it this way: for consistency, each action in non-terminal states has 3 separate transitions (certain actions in corner states could be represented with only 2, but again, let me be consistent): 1 to intended cell & 2 to cells in orthogonal directions.

– Có thể vẫn còn hơi khó hiểu, nhưng hãy xem xét theo cách này: để thống nhất, mỗi hành động ở trạng thái không kết thúc có 3 chuyển tiếp riêng biệt (một số hành động ở trạng thái góc có thể chỉ được biểu diễn bằng 2, nhưng 1 lần nữa, hãy để tôi thống nhất): 1 đến ô dự định & 2 đến các ô theo hướng trực giao.

- **Reward signal: Carrots & sticks.** Reward function R maps a transition tuple s, a, s' to a scalar. Reward function gives a numeric signal of goodness to transitions. When signal is positive, can think of reward as an income or a reward. most problems have at least 1 positive signal – winning a chess match or reaching desired destination, e.g. But, rewards can also be negative, & we can see these as cost, punishment, or penalty. In robotics, adding a time step cost is a common practice because usually want to reach a goal, but within a number of time steps. 1 thing to clarify: whether positive or negative, scalar coming out of reward function is always referred to as *reward*. RL folks are happy folks.

  – Tín hiệu thưởng: Cà rốt & gậy. Hàm thưởng R ánh xạ 1 bộ chuyển đổi s, a, s' thành 1 số vô hướng. Hàm thưởng cung cấp 1 tín hiệu số về độ tốt của các chuyển đổi. Khi tín hiệu là dương, có thể coi phần thưởng là thu nhập hoặc phần thưởng. hầu hết các bài toán đều có ít nhất 1 tín hiệu dương – thắng 1 ván cờ hoặc đạt đến đích mong muốn, ví dụ. Nhưng phần

thưởng cũng có thể là âm, & chúng ta có thể coi chúng là chi phí, hình phạt hoặc hình phạt. Trong robot học, việc thêm chi phí bước thời gian là 1 thực hành phổ biến vì thường muốn đạt được mục tiêu, nhưng trong 1 số bước thời gian. 1 điều cần làm rõ: dù dương hay âm, số vô hướng phát ra từ hàm thưởng luôn được gọi là *reward*. Những người RL là những người hạnh phúc.

Also important to highlight: while reward function can be represented as $R(s, a, s')$, which is explicit, could also use $R(s, a)$, or even $R(s)$, depending on our needs. Sometimes rewarding agent based on state is what we need; sometime it makes more sense to use action & state. However, most explicit way to represent reward function: use a state, action, & next state triplet. With that, can compute marginalization over next states in $R(s, a, s')$ to obtain $R(s, a)$, & marginalization over actions in $R(s, a)$ to get $R(s)$. But, once we are in $R(s)$ we can't recover $R(s, a)$ or $R(s, a, s')$, & once we are in $R(s, a)$ we can't recover $R(s, a, s')$.

– Cũng quan trọng cần nhấn mạnh: trong khi hàm phần thưởng có thể được biểu diễn là $R(s, a, s')$, là rõ ràng, cũng có thể sử dụng $R(s, a)$, hoặc thậm chí $R(s)$, tùy thuộc vào nhu cầu của chúng ta. Đôi khi, tác nhân thưởng dựa trên trạng thái là những gì chúng ta cần; đôi khi, sử dụng hành động & trạng thái hợp lý hơn. Tuy nhiên, cách rõ ràng nhất để biểu diễn hàm phần thưởng: sử dụng bộ ba trạng thái, hành động, & trạng thái tiếp theo. Với điều đó, có thể tính toán biên độ trên các trạng thái tiếp theo trong $R(s, a, s')$ để thu được $R(s, a)$, & biên độ trên các hành động trong $R(s, a)$ để thu được $R(s)$. Nhưng, 1 khi chúng ta ở trong $R(s)$, chúng ta không thể khôi phục $R(s, a)$ hoặc $R(s, a, s')$, & 1 khi chúng ta ở trong $R(s, a)$, chúng ta không thể khôi phục $R(s, a, s')$.

Reward function. 1. Reward function can be defined as follows. $r(s, a) = \mathbb{E}[R_t | S_{t-1} = s, A_{t-1} = a]$. 2. It can be defined as a function that takes in a state-action pair. 3. It is expectation of reward at time step t, given state-action pair in previous time step. 4. But it can also be defined as a function that takes a full transition tuple s, a, s':

$$r(s, a, s') = \mathbb{E}[R_t | S_{t-1} = s, A_{t-1} = a, S_t = s'].$$

5. & it is also defined as expectation, but now given that transition tuple. 6. Reward at time step t comes from a set of all rewards $\mathcal{R}$, which is a subset of all real numbers: $R_t \in \mathcal{R} \subset \mathbb{R}$.

– **Hàm thưởng.** 1. Hàm thưởng có thể được định nghĩa như sau. $r(s, a) = \mathbb{E}[R_t | S_{t-1} = s, A_{t-1} = a]$. 2. Nó có thể được định nghĩa là 1 hàm nhận vào 1 cặp trạng thái-hành động. 3. Nó là kỳ vọng về phần thưởng tại bước thời gian t, cho cặp trạng thái-hành động ở bước thời gian trước đó. 4. Nhưng nó cũng có thể được định nghĩa là 1 hàm nhận vào 1 bộ chuyển tiếp đầy đủ s, a, s':

$$r(s, a, s') = \mathbb{E}[R_t | S_{t-1} = s, A_{t-1} = a, S_t = s'].$$

5. & nó cũng được định nghĩa là kỳ vọng, nhưng bây giờ với bộ chuyển tiếp đó. 6. Phần thưởng tại bước thời gian t đến từ 1 tập hợp tất cả các phần thưởng $\mathcal{R}$, là 1 tập hợp con của tất cả các số thực: $R_t \in \mathcal{R} \subset \mathbb{R}$.

In FL environment, reward function is +1 for landing in state 15, 0 otherwise. Again, for clarity to following image, have only added reward signal to transitions that give a nonzero reward, landing on final state (state 15).

– Trong môi trường FL, hàm thưởng là +1 khi hạ cánh ở trạng thái 15, ngược lại là 0. 1 lần nữa, để hình ảnh sau rõ ràng hơn, chúng tôi chỉ thêm tín hiệu thưởng vào các chuyển đổi mang lại phần thưởng khác không, hạ cánh ở trạng thái cuối cùng (trạng thái 15).

There are only 3 ways to land on 15.

1. Selecting Right action in state 14 will transition agent with 33.3% chance there (33.3% to state 10 & 0.33% back to 14). But
2. Selecting Up &
3. Down action from state 14 will unintentionally also transition agent there with 33.3% probability for each action.

See difference between actions & transitions? Interesting to see how stochasticity complicates things, right? Reward signal for states with nonzero reward transitions.

– Chỉ có 3 cách để hạ cánh ở trạng thái 15.

1. Chọn hành động Đúng ở trạng thái 14 sẽ chuyển tác nhân với xác suất 33,3% ở đó (33,3% sang trạng thái 10 & 0,33% trở lại trạng thái 14). Nhưng
2. Chọn hành động Lên &
3. Xuống từ trạng thái 14 cũng vô tình chuyển tác nhân ở đó với xác suất 33,3% cho mỗi hành động.

Bạn thấy sự khác biệt giữa hành động & chuyển tiếp chứ? Thật thú vị khi thấy tính ngẫu nhiên làm mọi thứ phức tạp như thế nào, phải không? Tín hiệu thưởng cho các trạng thái có chuyển tiếp thưởng khác không.

Expanding transition & reward functions innto a table form is also useful. Following is format I recommend for most problems. Notice: being explicit, & several of these transitions could be grouped & refactored (e.g., corner cells).

– Việc mở rộng các hàm chuyển tiếp & thành dạng bảng cũng hữu ích. Sau đây là định dạng tôi khuyên dùng cho hầu hết các bài toán. Lưu ý: vì rõ ràng, & 1 số chuyển tiếp này có thể được nhóm lại & cấu trúc lại (ví dụ: ô góc).

- **Horizon: Time changes what's optimal.** Can represent time in MDPs as well. A *time step*, also referred to as epoch, cycle, iteration, or even interaction, is a global clock syncing all parties & discretizing time. Having a clock gives rise to a couple of possible types of tasks. An *episodic* task is a task in which there is a finite number of time steps, either because clock stops or because agent reaches a terminal state. There are also continuing tasks, which are tasks that go on forever;

there are no terminal states, so there is an infinite number of time steps. In this type of task, agent must be stopped manually.

– Horizon: Thời gian thay đổi những gì tối ưu. Cũng có thể biểu diễn thời gian trong MDP. 1 *bước thời gian*, còn được gọi là kỷ nguyên, chu kỳ, lặp lại, hoặc thậm chí tương tác, là 1 đồng hồ toàn cục đồng bộ hóa tất cả các bên & rời rạc hóa thời gian. Việc có 1 đồng hồ sẽ tạo ra 1 vài loại tác vụ khả thi. 1 tác vụ *theo từng giai đoạn* là 1 tác vụ trong đó có 1 số bước thời gian hữu hạn, hoặc do đồng hồ dừng hoặc do tác nhân đạt đến trạng thái kết thúc. Cũng có các tác vụ liên tục, là các tác vụ diễn ra mãi mãi; không có trạng thái kết thúc, vì vậy có vô số bước thời gian. Trong loại tác vụ này, tác nhân phải được dừng thủ công.

Episodic & continuing tasks can also be defined from agent's perspective. Call it *planning horizon*. On 1 hand, a *finite horizon* is a planning horizon in which agent knows task will terminate in a finite number of time steps: if forced agent to complete frozen lake environment in 15 steps, e.g. A special case of this kind of planning horizon is called a *greedy horizon*, of which planning horizon is one. BW & BSW have both a greedy planning horizon: episode terminates immediately after 1 interaction. In fact, all bandit environments have greedy horizons.

– Các tác vụ theo từng tập & liên tục cũng có thể được định nghĩa từ góc nhìn của tác nhân. Gọi nó là *planning horizon*. Mặt khác, *finite horizon* là 1 chân trời lập kế hoạch mà tác nhân biết rằng tác vụ sẽ kết thúc sau 1 số bước thời gian hữu hạn: nếu tác nhân bị buộc phải hoàn thành môi trường hồ đóng băng trong 15 bước, ví dụ: nếu 1 tác nhân bị ép phải hoàn thành môi trường hồ đóng băng trong 15 bước, ví dụ: nếu 1 tác nhân bị ép phải hoàn thành môi trường hồ đóng băng trong 15 bước, thì chân trời lập kế hoạch là 1 trong số đó. BW & BSW đều có chân trời lập kế hoạch tham lam: tập kết thúc ngay sau 1 tương tác. Trên thực tế, tất cả các môi trường bandit đều có chân trời tham lam.

On other hand, an *infinite horizon* when agent doesn't have a predetermined time step limit, so agent plans for an infinite number of time steps. Such a task may still be episodic & therefore terminate, but from perspective of agent, its planning horizon is infinite. Refer to this type of infinite planning horizon tasks as an *indefinite horizon* task. Agent plans for infinite, but interactions may be stopped at any time by environment.

– Mặt khác, 1 *infinite horizon* khi tác nhân không có giới hạn bước thời gian định trước, do đó tác nhân lập kế hoạch cho 1 số bước thời gian vô hạn. 1 tác vụ như vậy vẫn có thể là từng đợt & do đó kết thúc, nhưng từ góc nhìn của tác nhân, tầm nhìn lập kế hoạch của nó là vô hạn. Hãy gọi loại tác vụ tầm nhìn lập kế hoạch vô hạn này là 1 tác vụ *indefinite horizon*. Tác nhân lập kế hoạch cho vô hạn, nhưng tương tác có thể bị dừng bất cứ lúc nào bởi môi trường.

For tasks in which there's a high chance agent gets stuck in a loop & never terminates, it is common practice to add an artificial terminal state based on time step: a hard time step limit using transition function. These cases require special handling of time step limit terminal state. Environment for Chaps. 8–10, cart pole environment, has this kind of artificial terminal step, & will learn to handle these special cases in those chaps.

– Đối với các tác vụ có khả năng cao tác nhân bị kẹt trong vòng lặp & không bao giờ kết thúc, thông lệ thường thấy là thêm trạng thái kết thúc nhân tạo dựa trên bước thời gian: giới hạn bước thời gian khó bằng hàm chuyển tiếp. Những trường hợp này yêu cầu xử lý đặc biệt trạng thái kết thúc giới hạn bước thời gian. Môi trường cho Chương 8-10, môi trường cột xe đẩy, có loại bước kết thúc nhân tạo này, & sẽ học cách xử lý các trường hợp đặc biệt này trong các chương đó.

BW, BSW, & FL environments are episodic tasks, because there are terminal states; there are a clear goal & failure states. FL is an indefinite planning horizon; agent plans for infinite number of steps, but interactions may stop at any time. Won't add a time step limit to FL environment because there's a high chance agent will terminate naturally; environment is highly stochastic. This kind of task is most common in RL.

– Môi trường BW, BSW, & FL là các tác vụ theo từng giai đoạn, vì có các trạng thái kết thúc; có mục tiêu rõ ràng & trạng thái thất bại. FL là 1 đường chân trời lập kế hoạch không xác định; tác nhân lập kế hoạch cho vô số bước, nhưng các tương tác có thể dừng lại bất cứ lúc nào. Sẽ không thêm giới hạn bước thời gian vào môi trường FL vì khả năng tác nhân tự động kết thúc là rất cao; môi trường có tính ngẫu nhiên cao. Loại tác vụ này phổ biến nhất trong RL.

Refer to sequence of consecutive time steps from beginning to end of an episodic task as an *episode, trial, period*, or *stage*. In indefinite planning horizons, an episode is a collection containing all interactions between an initial & a terminal state.

– Tham khảo chuỗi các bước thời gian liên tiếp từ đầu đến cuối của 1 nhiệm vụ theo từng giai đoạn như 1 *episode, trial, period* hoặc *stage*. Trong các chân trời lập kế hoạch không xác định, 1 giai đoạn là 1 tập hợp chứa tất cả các tương tác giữa trạng thái ban đầu & trạng thái kết thúc.

- **Discount: future is uncertain, value it less.** Because of possibility of infinite sequences of time steps in infinite horizon tasks, need a way to discount value of rewards over time; i.e., need a way to tell agent that getting +1's is better sooner than later. Commonly use a positive real value < 1 to exponentially discount value of future rewards. Further into future we receive rewards, less valuable it is in present.

  – Giảm giá: tương lai không chắc chắn, hãy giảm giá trị của nó. Do khả năng có vô số chuỗi bước thời gian trong các nhiệm vụ có chân trời vô hạn, cần 1 cách để giảm giá trị của phần thưởng theo thời gian; i.e., cần 1 cách để cho tác nhân biết rằng nhận được +1 sớm hơn sẽ tốt hơn muộn. Thông thường, sử dụng giá trị thực dương < 1 để giảm giá trị theo cấp số nhân của phần thưởng trong tương lai. Càng về sau, chúng ta càng nhận được phần thưởng, nhưng giá trị của nó ở hiện tại lại kém hơn.

  This number is called *discount factor*, or *gamma*. Discount factor adjusts importance of rewards over time. The later we receive rewards, the less attractive they are to present calculations. Another important reason why discount factor is commonly used: reduce variance of return estimates. Given: future is uncertain, & further we look into future, more stochasticity we accumulate & more variance our value estimates will have, discount factor helps reduce degree to which future rewards affect our value function estimates, which stabilizes learning for most agents.

– Con số này được gọi là *hệ số chiết khấu*, hay *gamma*. Hệ số chiết khấu điều chỉnh tầm quan trọng của phần thưởng theo thời gian. Phần thưởng càng nhận được muộn thì chúng càng kém hấp dẫn khi trình bày các phép tính. 1 lý do quan trọng khác khiến hệ số chiết khấu được sử dụng phổ biến: giảm phương sai của ước tính lợi nhuận. Giả sử: tương lai không chắc chắn, & khi chúng ta càng nhìn vào tương lai, chúng ta càng tích lũy nhiều ngẫu nhiên & ước tính giá trị của chúng ta sẽ có nhiều phương sai hơn, hệ số chiết khấu giúp giảm mức độ ảnh hưởng của phần thưởng trong tương lai đến ước tính hàm giá trị của chúng ta, giúp ổn định quá trình học tập cho hầu hết các tác nhân.

Effect of discount factor & time on value of rewards. 1. Discount factor will exponentially decay value of later rewards. 2. Value of a +1 reward at time step 0 isn't same as value of a +1 reward at time step 1000.

– Ảnh hưởng của hệ số chiết khấu & thời gian lên giá trị phần thưởng. 1. Hệ số chiết khấu sẽ làm giảm giá trị của phần thưởng sau theo cấp số nhân. 2. Giá trị của phần thưởng +1 tại bước thời gian 0 không giống với giá trị của phần thưởng +1 tại bước thời gian 1000.

Interestingly, gamma is part of MDP definition: problem, & not agent. However, often you will find no guidance for proper value of gamma to use for a given environment. Again, this is because gamma is also used as a hyperparameter for reducing variance, & therefore left for agent to tune.

– Điều thú vị là gamma là 1 phần của định nghĩa MDP: vấn đề, & chứ không phải tác nhân. Tuy nhiên, thường bạn sẽ không tìm thấy hướng dẫn nào về giá trị gamma phù hợp để sử dụng cho 1 môi trường nhất định. 1 lần nữa, điều này là do gamma cũng được sử dụng như 1 siêu tham số để giảm phương sai, & do đó được để lại cho tác nhân điều chỉnh.

Can also use gamma as a way to give sense of "urgency" to agent. To wrap your head around that, imagine: I tell you I will give you $1000 once you finish reading this book, but I will discount (gamma) that reward by 0.5 daily, i.e., every day I cut value that I pay in half. You will probably finish reading this book today. If I say gamma is 1, then it doesn't matter when you finish it, you still get full amount.

– Cũng có thể sử dụng gamma như 1 cách để tạo cảm giác "khẩn cấp" cho người đại diện. Để hiểu rõ hơn, hãy tưởng tượng: Tôi nói với bạn rằng tôi sẽ thưởng cho bạn 1000 đô la sau khi bạn đọc xong cuốn sách này, nhưng tôi sẽ giảm (gamma) phần thưởng đó 0,5 đô la mỗi ngày, i.e., mỗi ngày tôi sẽ giảm 1 nửa giá trị mà tôi trả. Có lẽ bạn sẽ đọc xong cuốn sách này trong hôm nay. Nếu tôi nói gamma là 1, thì dù bạn đọc xong khi nào cũng không quan trọng, bạn vẫn nhận được toàn bộ số tiền.

For BW & BSW environments, a gamma of 1 is appropriate; for FL environment, however, we will use a gamma of 0.99, a commonly used value.

– Đối với môi trường BW & BSW, gamma bằng 1 là phù hợp; tuy nhiên, đối với môi trường FL, chúng ta sẽ sử dụng gamma bằng 0,99, 1 giá trị thường được sử dụng.

Discount factor (gamma). 1. Sum of all rewards obtained during course of an episode is referred to as return $G_t = \sum_{i=t+1}^{T} R_i = R_{t+1} + R_{t+2} + \cdots + R_T$. 2. But we can also use discount factor this way & obtain discounted return. Discounted return will downweight rewards that occur later during episode.

$$G = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T.$$

3. Can simplify equation & have a more general equation, e.g. this one. $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$. 4. Finally, take a look at this interesting recursive definition. $G_t = R_{t+1} + \gamma G_{t+1}$. In next chap, spend time exploiting this form.

– **Hệ số chiết khấu (gamma).** 1. Tổng tất cả các phần thưởng nhận được trong suốt 1 tập phim được gọi là lợi nhuận $G_t = \sum_{i=t+1}^{T} R_i = R_{t+1} + R_{t+2} + \cdots + R_T$. 2. Nhưng chúng ta cũng có thể sử dụng hệ số chiết khấu theo cách này & thu được lợi nhuận chiết khấu. Lợi nhuận chiết khấu sẽ giảm trọng số phần thưởng xuất hiện sau đó trong tập phim.

$$G = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T.$$

3. Có thể đơn giản hóa phương trình & có 1 phương trình tổng quát hơn, ví dụ như phương trình này. $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$. 4. Cuối cùng, hãy xem định nghĩa đệ quy thú vị này. $G_t = R_{t+1} + \gamma G_{t+1}$. Trong chương tiếp theo, hãy dành thời gian khai thác dạng này.

- **Extensions to MDPs.** There are many extensions to MDP framework. They allow us to target slightly different types of RL problems. Following list isn't comprehensive, but it should give you an idea of how large field is. Know that acronym MDPs is often used to refer to all types of MDPs. Currently looking only at tip of iceberg:
  - Partially observable Markov decision process (POMDP): When agent cannot fully observe environment state
  - Factored Markov decision process (FMDP): Allows representation of transition & reward function more compactly so that we can represent large MDPs
  - Continuous [Time|Action|State] Markov decision process: When either time, action, state or any combination of them are continuous
  - Relational Markov decision process (RMDP): Allows combination of probabilistic & relational knowledge
  - Semi-Markov decision process (SMDP): Allows inclusion of abstract actions that can take multiple time steps to complete
  - Multi-agent Markov decision process (MMDP): Allows inclusion of multiple agents in same environment
  - Decentralized Markov decision process (Dec-MDP): Allows for multiple agents to collaborate & maximize a common reward

– Mở rộng cho MDP. Có rất nhiều phần mở rộng cho khung MDP. Chúng cho phép chúng ta nhắm đến các loại bài toán RL khác nhau đôi chút. Danh sách sau đây không đầy đủ, nhưng cũng đủ để bạn hình dung được trường dữ liệu lớn như thế nào. Cần lưu ý rằng từ viết tắt MDP thường được dùng để chỉ tất cả các loại MDP. Hiện tại chỉ nhìn vào phần nổi của tảng băng chìm:

* Quy trình quyết định Markov quan sát 1 phần (POMDP): Khi tác nhân không thể quan sát đầy đủ trạng thái môi trường
* Quy trình quyết định Markov phân tích nhân tử (FMDP): Cho phép biểu diễn hàm chuyển tiếp & phần thưởng 1 cách cô đọng hơn để có thể biểu diễn các MDP lớn
* Quy trình quyết định Markov liên tục [Thời gian | Hành động | Trạng thái]: Khi thời gian, hành động, trạng thái hoặc bất kỳ sự kết hợp nào của chúng là liên tục
* Quy trình quyết định Markov quan hệ (RMDP): Cho phép kết hợp kiến thức xác suất & quan hệ
* Quy trình quyết định bán Markov (SMDP): Cho phép bao gồm các hành động trừu tượng có thể mất nhiều bước thời gian để hoàn thành
* Quy trình quyết định Markov đa tác nhân (MMDP): Cho phép bao gồm nhiều tác nhân trong cùng 1 môi trường
* Quy trình quyết định Markov phi tập trung (Dec-MDP): Cho phép nhiều tác nhân cộng tác & tối đa hóa phần thưởng chung

Bandit walk (BW) MDP. 1. Outer dictionary keys are states. 2. Inner dictionary keys are actions. 3 Value of inner dictionary is a list with all positive transitions for that state-action pair. 4. Transition tuples have 4 values: probability of that transition, next state, reward, & a flag indicating whether next state is terminal. 5. Can also load MDP this way.

– Bandit walk (BW) MDP. 1. Khóa từ điển bên ngoài là các trạng thái. 2. Khóa từ điển bên trong là các hành động. 3. Giá trị của từ điển bên trong là 1 danh sách chứa tất cả các chuyển đổi tích cực cho cặp trạng thái-hành động đó. 4. Các bộ chuyển đổi có 4 giá trị: xác suất của chuyển đổi đó, trạng thái tiếp theo, phần thưởng, & 1 cờ cho biết trạng thái tiếp theo có phải là kết thúc hay không. 5. Cũng có thể tải MDP theo cách này.

```
import gym, gym_walk
P = gym.make('BanditWalk-v0').env.P
```

Bandit slippery walk (BSW) MDP. 1. Look at terminal states. State 0 & 2 are terminal. 2. This is how you build stochastic transitions. This is state 1, action 0. 3. These are transitions after taking action 1 in state 1. 4. This is how you can load Bandit Slippery walk in Notebook:

```
import gym, gym_walk
P = gym.make('BanditSlipperyWalk-v0').env.P
```

Frozen lake (FL) MDP. 1. Probability of landing in state 0 when selecting action 0 in state 0. 2. Probability of landing in state 4 when selecting action 0 in state 0. 3. Can group probabilities, e.g. in this line. 4. Or be explicit, e.g. in these 2 lines. It works fine either way. 6. Go to Notebook for complete FL MDP. 7. State 4 is only state that provides a nonzero reward. 3 out of 4 actions have a single transition that leads to state 15. Landing on state 15 provides a +1 reward. 8. State 15 is a terminal state. 9. Can load MDP like so.

– Hồ đóng băng (FL) MDP. 1. Xác suất hạ cánh ở trạng thái 0 khi chọn hành động 0 ở trạng thái 0. 2. Xác suất hạ cánh ở trạng thái 4 khi chọn hành động 0 ở trạng thái 0. 3. Có thể nhóm các xác suất, ví dụ: trong dòng này. 4. Hoặc rõ ràng, ví dụ: trong 2 dòng này. Cả hai cách đều ổn. 6. Vào Notebook để xem FL MDP đầy đủ. 7. Trạng thái 4 là trạng thái duy nhất cung cấp phần thưởng khác không. 3 trong số 4 hành động có 1 lần chuyển đổi duy nhất dẫn đến trạng thái 15. Hạ cánh ở trạng thái 15 cung cấp phần thưởng +1. 8. Trạng thái 15 là trạng thái cuối cùng. 9. Có thể tải MDP như vậy.

```
import gym
P = gym.make('FrozenLake-v0').env.P
```

- **Putting it all together.** Unfortunately, when go out to real world, find many different ways that MDPs are defined. Moreover, certain sources describe POMDPs & refer to them as MDPs without full disclosure. All of this creates confusion to newcomers, so have a few points to clarify. 1st, what you saw previously as Python code isn't a complete MDP, but instead only transition functions & reward signals. From these, can easily infer state & action spaces. These code snippets come from a few packages containing several environments I developed for this book, & FL environment is part of OpenAI Gym package mentioned in 1st chap. Several of additional components of an MDP that are missing from dictionaries above, e.g. initial state distribution S_θ that comes from set of initial state S^i, are handled internally by Gym framework & not shown here. Further, other components, e.g. discount factor γ & horizon H, are not shown in previous dictionary, & OpenAI Gym framework doesn't provide them to you. Discount factors are commonly considered hyperparameters, for better or worse. & horizon is often assumed to be infinity.

– Tổng hợp lại. Thật không may, khi bước ra thế giới thực, bạn sẽ thấy nhiều cách định nghĩa MDP khác nhau. Hơn nữa, 1 số nguồn mô tả POMDP & gọi chúng là MDP mà không tiết lộ đầy đủ. Tất cả những điều này gây nhầm lẫn cho người mới bắt đầu, vì vậy, có 1 vài điểm cần làm rõ. Thứ nhất, những gì bạn thấy trước đây là mã Python không phải là MDP hoàn chỉnh, mà thay vào đó chỉ là các hàm chuyển tiếp & tín hiệu phần thưởng. Từ những điều này, có thể dễ dàng suy ra trạng thái & không gian hành động. Các đoạn mã này đến từ 1 vài gói chứa 1 số môi trường mà tôi đã phát triển cho

cuốn sách này, & môi trường FL là 1 phần của gói OpenAI Gym được đề cập trong chương 1. 1 số thành phần bổ sung của MDP bị thiếu trong các từ điển ở trên, ví dụ: phân phối trạng thái ban đầu S_θ xuất phát từ tập hợp trạng thái ban đầu S^i, được xử lý nội bộ bởi khung Gym & không được hiển thị ở đây. Ngoài ra, các thành phần khác, ví dụ: Hệ số chiết khấu γ & horizon H không được hiển thị trong từ điển trước, & OpenAI Gym framework không cung cấp chúng cho bạn. Hệ số chiết khấu thường được coi là siêu tham số, dù tốt hay xấu. & horizon thường được coi là vô cực.

1st, to calculate optimal policies for MDPs presented in this chap (which we will do in next chap), only need dictionary shown previously containing transition function & reward signal; from these, can infer state & action spaces, & I will provide you with discount factors. Will assume horizons of infinity, & won't need initial state distribution. Additionally, most crucial part of this chap: give you an awareness of components of MDPs & POMDPs. Won't have to do much more buildings of MDPs than what you have done in this chap. Nevertheless, define MDPs & POMDPs so we are in sync.

– 1st, để tính toán các chính sách tối ưu cho các MDP được trình bày trong chương này (chúng ta sẽ làm điều này trong chương tiếp theo), chỉ cần từ điển đã hiển thị trước đó chứa hàm chuyển tiếp & tín hiệu phần thưởng; từ đó, có thể suy ra trạng thái & không gian hành động, & Tôi sẽ cung cấp cho bạn các hệ số chiết khấu. Sẽ giả định chân trời vô cực, & không cần phân phối trạng thái ban đầu. Ngoài ra, phần quan trọng nhất của chương này: cung cấp cho bạn nhận thức về các thành phần của MDP & POMDP. Sẽ không phải xây dựng nhiều MDP hơn những gì bạn đã làm trong chương này. Tuy nhiên, hãy định nghĩa MDP & POMDP để chúng ta đồng bộ.

MDPs vs. POMDPs. $MDP(S, A, T, R, S_\theta, \gamma, H)$. 1. MDPs have state space $\mathcal{S}$, action space $\mathcal{A}$, transition function T, reward signal $\mathcal{R}$. They also has a set of initial states distribution S_θ, discount factor γ, & horizon H. 2. To define a POMDP, add observation space O & an emission probability $\mathcal{E}$ that defines probability of showing an observation o_t given a state s_t. Very simple. $POMDP(S, A, T, R, S_\theta, \gamma, H, O, E)$.

– **MDP so với POMDP.** $MDP(S, A, T, R, S_\theta, \gamma, H)$. 1. MDP có không gian trạng thái $\mathcal{S}$, không gian hành động $\mathcal{A}$, hàm chuyển tiếp T, tín hiệu thưởng $\mathcal{R}$. Chúng cũng có 1 tập hợp phân phối trạng thái ban đầu S_θ, hệ số chiết khấu γ, & chân trời H. 2. Để định nghĩa 1 POMDP, hãy thêm không gian quan sát O & 1 xác suất phát xạ $\mathcal{E}$ định nghĩa xác suất hiển thị 1 quan sát o_t cho 1 trạng thái s_t. Rất đơn giản. $POMDP(S, A, T, R, S_\theta, \gamma, H, O, E)$.

- **Summary.** This chap is heavy on new terms, but that is its intent. Best summary for this chap is MDP so với POMDP. At highest level, a RL problem is about interactions between an agent & environment in which agent exists. A large variety of issues can be modeled under this setting. Markov decision process is a mathematical framework for representing complex decision-making problems under uncertainty.

– Chương này tập trung vào các thuật ngữ mới, nhưng đó chính là mục đích của nó. Tóm tắt tốt nhất cho chương này là MDP so với POMDP. Ở cấp độ cao nhất, 1 bài toán RL là về tương tác giữa 1 tác nhân & môi trường mà tác nhân tồn tại. Rất nhiều vấn đề có thể được mô hình hóa trong bối cảnh này. Quy trình quyết định Markov là 1 khuôn khổ toán học để biểu diễn các vấn đề ra quyết định phức tạp trong điều kiện không chắc chắn.

Markov decision processes (MDPs) are composed of a set of system states, a set of per-state actions, a transition function, a reward signal a horizon, a discount factor, & an initial state distribution. States describe configuration of environment. Actions allow agents to interact with environment. Transition function tells how environment evolves & reacts to agent's actions. Reward signal encodes goal to be achieved by agent. Horizon & discount factor add a notion of time to interactions.

– Quy trình quyết định Markov (MDP) bao gồm 1 tập hợp các trạng thái hệ thống, 1 tập hợp các hành động trên mỗi trạng thái, 1 hàm chuyển tiếp, 1 tín hiệu thưởng, 1 đường chân trời, 1 hệ số chiết khấu & 1 phân phối trạng thái ban đầu. Các trạng thái mô tả cấu hình của môi trường. Các hành động cho phép các tác nhân tương tác với môi trường. Hàm chuyển tiếp cho biết môi trường phát triển như thế nào & phản ứng với các hành động của tác nhân. Tín hiệu thưởng mã hóa mục tiêu mà tác nhân cần đạt được. Đường chân trời & hệ số chiết khấu bổ sung khái niệm về thời gian cho các tương tác.

State space, set of all possible states, can be infinite or finite. Number of variables that make up a single state, however, must be finite. States can be fully observable, but in a more general case of MDPs, a POMDP, states are partially observable. I.e., agent can't observe full state of system, but can observe a noisy state instead, called an observation.

– Không gian trạng thái, tập hợp tất cả các trạng thái khả dĩ, có thể vô hạn hoặc hữu hạn. Tuy nhiên, số lượng biến tạo nên 1 trạng thái duy nhất phải hữu hạn. Các trạng thái có thể quan sát được hoàn toàn, nhưng trong 1 trường hợp tổng quát hơn của MDP, tức POMDP, các trạng thái chỉ quan sát được 1 phần. I.e., tác nhân không thể quan sát toàn bộ trạng thái của hệ thống, mà thay vào đó có thể quan sát 1 trạng thái nhiễu, được gọi là 1 quan sát.

Action space is a set of actions that can vary from state to state. However, convention is to use same set for all states. Actions can be composed with > 1 variable, just like states. Action variables may be discrete or continuous.

– Không gian hành động là 1 tập hợp các hành động có thể thay đổi tùy theo trạng thái. Tuy nhiên, quy ước là sử dụng cùng 1 tập hợp cho tất cả các trạng thái. Các hành động có thể được cấu thành từ > 1 biến, giống như trạng thái. Biến hành động có thể là rời rạc hoặc liên tục.

Transition function links a state (a next state) to a state-action pair, & it defines probability of reaching that future state given state-action pair. Reward signal, in its more general form, maps a transition tuple s, a, s' to scalar, & it indicates goodness of transition. Both transition function & reward signal define model of environment & are assumed to be stationary, meaning probabilities stay same throughout.

– Hàm chuyển tiếp liên kết 1 trạng thái (trạng thái tiếp theo) với 1 cặp trạng thái-hành động, & nó xác định xác suất đạt đến trạng thái tương lai đó dựa trên cặp trạng thái-hành động. Tín hiệu phần thưởng, ở dạng tổng quát hơn, ánh xạ 1 bộ chuyển tiếp s, a, s' thành số vô hướng, & nó biểu thị độ tốt của chuyển tiếp. Cả hàm chuyển tiếp & tín hiệu phần thưởng đều xác định mô hình môi trường & được giả định là dừng, nghĩa là xác suất luôn giữ nguyên trong suốt quá trình.

By now:

* Understand components of a RL problem & how they interact with each other
* Recognize Markov decision processes & know what they are composed from & how they work
* Can represent sequential decision-making problems as MDPs

– Đến thời điểm hiện tại:

* Hiểu các thành phần của bài toán RL & cách chúng tương tác với nhau
* Nhận biết các quy trình quyết định Markov & biết chúng được cấu thành từ những gì & cách chúng hoạt động
* Có thể biểu diễn các bài toán ra quyết định tuần tự dưới dạng MDP

- **3. Balancing immediate & long-term goals.** In this chap: learn about challenges of learning from sequential feedback & how to properly balance immediate & long-term goals. Develop algorithms that can find best policies of behavior in sequential decision-making problems modeled with MDPs. Find optimal policies for all environments for which you built MDPs in prev chap.

  – Cân bằng mục tiêu trước mắt & dài hạn. Trong chương này: tìm hiểu về những thách thức của việc học từ phản hồi tuần tự & cách cân bằng đúng đắn các mục tiêu trước mắt & dài hạn. Phát triển các thuật toán có thể tìm ra các chính sách hành vi tốt nhất trong các bài toán ra quyết định tuần tự được mô hình hóa bằng MDP. Tìm các chính sách tối ưu cho tất cả các môi trường mà bạn đã xây dựng MDP trong chương trước.

  In last chap, built an MDP for BW, BSW, & FL environments. MDPs are motors moving RL environments. They define problem: they describe how agent interacts with environment through state & action spaces, agent's goal through reward function, how environment reacts from agent's actions through transition function, & how time should impact behavior through discount factor.

  – Trong chương trước, tôi đã xây dựng 1 MDP cho các môi trường BW, BSW, & FL. MDP là động cơ di chuyển môi trường RL. Chúng định nghĩa vấn đề: chúng mô tả cách tác nhân tương tác với môi trường thông qua không gian trạng thái & hành động, mục tiêu của tác nhân thông qua hàm thưởng, cách môi trường phản ứng với hành động của tác nhân thông qua hàm chuyển tiếp, & cách thời gian tác động đến hành vi thông qua hệ số chiết khấu.

  In this chap, learn about algorithms for solving MDPs. 1st discuss objective of an agent & why simple plans are not sufficient to solve MDPs. Then talk about 2 fundamental algorithms for solving MDPs under a technique called *dynamic programming: value iteration* (VI) & *policy iteration* (PI).

  – Trong chương này, chúng ta sẽ tìm hiểu về các thuật toán giải bài toán MDP. Trước tiên, hãy thảo luận về mục tiêu của 1 tác nhân & tại sao các kế hoạch đơn giản không đủ để giải bài toán MDP. Sau đó, hãy thảo luận về hai thuật toán cơ bản để giải bài toán MDP bằng 1 kỹ thuật gọi là *dynamic programming: value iteration* (VI) & *policy iteration* (PI).

  Soon notice these methods in a way "cheat": they require full access to MDP, & they depend on knowing dynamics of environment, which is something we can't always obtain. However, fundamentals you will learn are still useful for learning about more advanced algorithms. In end, VI & PI are foundations from which virtually every other RL (& DRL) algorithm originates.

  – Bạn sẽ sớm nhận ra những phương pháp này có phần "gian lận": chúng yêu cầu quyền truy cập đầy đủ vào MDP, & chúng phụ thuộc vào việc hiểu biết về động lực học của môi trường, điều mà chúng ta không phải lúc nào cũng có được. Tuy nhiên, những kiến thức cơ bản bạn sẽ học vẫn hữu ích cho việc tìm hiểu các thuật toán nâng cao hơn. Cuối cùng, VI & PI là nền tảng mà hầu như mọi thuật toán RL (& DRL) khác đều bắt nguồn từ đó.

  Also notice: when an agent has full access to an MDP, there is no uncertainty because you can look at dynamics & rewards & calculate expectations directly. Calculating expectations directly means: there is no need for exploration, i.e., there is no need to balance exploration & exploitation. There is no need for interaction, so there is no need for trial-&-error learning. All of this is because feedback we are using for learning in this chap isn't evaluative but supervised instead.

  – Cũng cần lưu ý: khi 1 tác nhân có toàn quyền truy cập vào MDP, sẽ không có sự bất định nào vì bạn có thể xem xét động lực & phần thưởng & tính toán kỳ vọng trực tiếp. Tính toán kỳ vọng trực tiếp có nghĩa là: không cần khám phá, i.e., không cần cân bằng giữa khám phá & khai thác. Không cần tương tác, nên không cần học thử & sai. Tất cả những điều này là do phản hồi mà chúng ta sử dụng cho việc học trong chương này không mang tính đánh giá mà thay vào đó là có giám sát.

  Remember, in DRL, agents learn from feedback that is simultaneously sequential (as opposed to 1 shot), evaluative (as opposed to supervised), & sampled (as opposed to exhaustive). What I am doing in this chap is eliminating complexity that comes with learning from evaluative & sampled feedback, & studying sequential feedback in isolation. In this chap, learn from feedback that is sequential, supervised, & exhaustive.

  – Nhớ rằng, trong DRL, các tác nhân học hỏi từ phản hồi vừa tuần tự (khác với 1 lần), vừa đánh giá (khác với giám sát), vừa lấy mẫu (khác với toàn diện). Điều tôi đang làm trong chương này là loại bỏ sự phức tạp đi kèm với việc học từ phản hồi đánh giá & lấy mẫu, vừa nghiên cứu phản hồi tuần tự 1 cách riêng biệt. Trong chương này, hãy học hỏi từ phản hồi tuần tự, vừa giám sát, vừa toàn diện.

  - **3.1. Objective of a decision-making agent.** At 1st, it seems agent's goal: find a sequence of actions that will maximize return: sum of rewards (discounted or undiscounted – depending on value of gamma) during an episode or entire life of agent, depending on task. Introduce a new environment to explain these concepts more concretely.

    – Mục tiêu của tác nhân ra quyết định. Đầu tiên, mục tiêu của tác nhân dường như là: tìm 1 chuỗi hành động giúp tối đa hóa lợi nhuận: tổng phần thưởng (đã chiết khấu hoặc chưa chiết khấu – tùy thuộc vào giá trị gamma) trong 1 giai đoạn

hoặc toàn bộ vòng đời của tác nhân, tùy thuộc vào nhiệm vụ. Giới thiệu 1 môi trường mới để giải thích các khái niệm này 1 cách cụ thể hơn.

Slippery Walk Five (SWF) environment. Slippery Walk Five (SWF) is a 1-row grid-world environment (a walk), that is stochastic, similar to Frozen Lake, & it has only 5 non-terminal states (7 total if count 2 terminal).

– **Môi trường Slippery Walk Five (SWF).** Slippery Walk Five (SWF) là môi trường thế giới lưới 1 hàng (một cuộc đi bộ), ngẫu nhiên, tương tự như Frozen Lake, & chỉ có 5 trạng thái không kết thúc (tổng cộng 7 nếu đếm 2 kết thúc).

Slippery Walk Five environment. 1. This environment is stochastic, & even if agent selects Right action, there is a chance it goes left. 2. 50% action is success. 3. 33.33% stays in place. 4. 16.66% goes backward. Agent starts in S, H is a hole, G is goal & provides a +1 reward.

– Môi trường Slippery Walk Five. 1. Môi trường này là ngẫu nhiên, & ngay cả khi đặc vụ chọn hành động Phải, vẫn có khả năng nó sẽ đi sang trái. 2. 50% hành động thành công. 3. 33,33% giữ nguyên vị trí. 4. 16,66% đi ngược lại. Đặc vụ bắt đầu ở S, H là 1 cái hố, G là mục tiêu & cung cấp phần thưởng +1.

Return G**.** 1. Return is sum of rewards encounter from step t, until final step T. $G_t = \sum_{i=t+1}^{T} R_i = R_{t+1} + R_{t+2} + \cdots + R_T$. 2. Can combine return & time using discount factor, gamma. This is then discounted return, which prioritizes early rewards. $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-1} R_T$ [NQBH: wrong index]. 3. Can simplify equation & have a more general equation, e.g. this one: $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$. 4. Start at this recursive definition of G for a while. $G_t = R_{t+1} + \gamma G_{t+1}$.

– **Trả về** G**.** 1. Trả về là tổng phần thưởng gặp phải từ bước t, cho đến bước cuối cùng T. $G_t = \sum_{i=t+1}^{T} R_i = R_{t+1} + R_{t+2} + \cdots + R_T$. 2. Có thể kết hợp thời gian trả về & bằng cách sử dụng hệ số chiết khấu, gamma. Sau đó, đây là lợi nhuận chiết khấu, ưu tiên các phần thưởng sớm. $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-1} R_T$ [NQBH: chỉ số sai]. 3. Có thể đơn giản hóa phương trình & có 1 phương trình tổng quát hơn, ví dụ như phương trình này: $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$. 4. Bắt đầu với định nghĩa đệ quy này của G trong 1 thời gian. $G_t = R_{t+1} + \gamma G_{t+1}$.

Can think of returns as backward looking – "how much you got" from a past time step; but another way to look at it is as a "reward-to-go" – basically, forward looking. E..g, imagine an episode in SWF environment went this way: State 3 (0 reward), state 4 (0 reward), state 5 (0 reward), state 4 (0 reward), state 5 (0 reward), state 6 (+1 reward). Can shorten it: 3/0, 4/0, 5/0, 4/0, 5/0, 6/1. What's the return of this trajectory/episode? If use discounting math would work out this way.

– Có thể coi lợi nhuận như 1 phép tính ngược – "bạn đã nhận được bao nhiêu" từ 1 bước thời gian trong quá khứ; nhưng 1 cách khác để xem xét nó là "phần thưởng để tiếp tục" – về cơ bản là hướng tới tương lai. Ví dụ, hãy tưởng tượng 1 tập phim trong môi trường SWF diễn ra như sau: Trạng thái 3 (0 phần thưởng), trạng thái 4 (0 phần thưởng), trạng thái 5 (0 phần thưởng), trạng thái 4 (0 phần thưởng), trạng thái 5 (0 phần thưởng), trạng thái 6 (0 phần thưởng), có thể rút gọn thành: 3/0, 4/0, 5/0, 4/0, 5/0, 6/1. Lợi nhuận của quỹ đạo/tập phim này là bao nhiêu? Nếu sử dụng phép tính chiết khấu, ta sẽ tính được như sau.

Discounted return in slippery walk 5 environment. If don't use discounting, well, return would be 1 for this trajectory & all trajectories that end in right-most cell, state 6, & 0 for all trajectories that terminate in left-most cell, state 0.

– **Lợi nhuận chiết khấu trong môi trường đi bộ trơn trượt 5.** Nếu không sử dụng chiết khấu, lợi nhuận sẽ là 1 cho quỹ đạo này & tất cả các quỹ đạo kết thúc ở ô ngoài cùng bên phải, trạng thái 6, & 0 cho tất cả các quỹ đạo kết thúc ở ô ngoài cùng bên trái, trạng thái 0.

In SWF environment, evident: going right is best thing to do. It may seem, therefore, all agent must find is something called a *plan* – i.e., a sequence of actions from START state to GOAL state. But this doesn't always work.

– Trong môi trường SWF, điều hiển nhiên là: đi đúng hướng là điều tốt nhất nên làm. Do đó, có vẻ như tất cả những gì tác nhân phải tìm là 1 thứ gọi là *plan* – i.e., 1 chuỗi hành động từ trạng thái BẮT ĐẦU đến trạng thái MỤC TIÊU. Nhưng điều này không phải lúc nào cũng hiệu quả.

In FL environment, a plane would look like following.

A solid plan in FL environment. 1. This is a solid plan. But, in a stochastic environment, even best of plans fail. Remember: in FL environment, unintended action effects have been higher probability: 66.66% & 33.33%. Need plan for unexpected.

– 1 kế hoạch vững chắc trong môi trường FL. 1. Đây là 1 kế hoạch vững chắc. Tuy nhiên, trong môi trường ngẫu nhiên, ngay cả những kế hoạch tốt nhất cũng thất bại. Hãy nhớ rằng: trong môi trường FL, các tác động không mong muốn có xác suất cao hơn: 66,66% & 33,33%. Cần có kế hoạch phòng ngừa những điều bất ngờ.

But this isn't enough! Problem with plans is they don't account for stochasticity in environments, & both SWF & FL are stochastic; actions taken won't always work way we intend. What would happen if, due to environment's stochasticity, our agent lands on a cell not covered by our plan?

– Nhưng như vậy vẫn chưa đủ! Vấn đề với kế hoạch là chúng không tính đến tính ngẫu nhiên trong môi trường, & cả SWF & FL đều ngẫu nhiên; các hành động được thực hiện không phải lúc nào cũng diễn ra theo đúng ý định. Điều gì sẽ xảy ra nếu, do tính ngẫu nhiên của môi trường, tác nhân của chúng ta rơi vào 1 ô không nằm trong kế hoạch?

A possible hole in our plane. 1. Say agent followed plan, but on 1st environment transition, agent was sent backward to state 2. 2. Now, what? You didn't plan an action for state 2. Maybe you need a plane B, C, D? Sample happens in FL environment.

– 1 lỗ hổng có thể có trên mặt phẳng của chúng ta. 1. Giả sử tác nhân đã làm theo kế hoạch, nhưng ở lần chuyển đổi môi trường đầu tiên, tác nhân bị gửi ngược về trạng thái 2. 2. Vậy thì sao? Bạn không lập kế hoạch hành động cho trạng thái 2. Có lẽ bạn cần 1 mặt phẳng B, C, D? Ví dụ xảy ra trong môi trường FL.

Plans aren't enough in stochastic environments. 1. Showing action & possible action effects. Notice: there is a 66.66% chance that an unintended consequence happens! 2. Imagine: agent is following plan, but in state 10, agent is sent to state 9, even if

it selected down action, as it apparently is right thing to do. 3. What we need is a plan for every possible state, a universal plan, a policy.

– Kế hoạch là không đủ trong môi trường ngẫu nhiên. 1. Hiển thị hành động & các hiệu ứng hành động có thể xảy ra. Lưu ý: có 66,66% khả năng xảy ra hậu quả không mong muốn! 2. Hãy tưởng tượng: tác nhân đang làm theo kế hoạch, nhưng ở trạng thái 10, tác nhân được chuyển đến trạng thái 9, ngay cả khi nó đã chọn hành động xuống, vì rõ ràng đó là điều nên làm. 3. Điều chúng ta cần là 1 kế hoạch cho mọi trạng thái có thể, 1 kế hoạch chung, 1 chính sách.

What agent needs to come up with is called a *policy*. Policies are universal plans; policies cover all possible states. Need to plan for every possible state. Policies can be stochastic or deterministic: policy can return action-probability distributions or single actions for a given state (or observation). For now, working with deterministic policies, which is a lookup table that maps actions to states.

– Những gì tác nhân cần đưa ra được gọi là *policy*. Chính sách là các kế hoạch phổ quát; chính sách bao gồm tất cả các trạng thái khả dĩ. Cần lập kế hoạch cho mọi trạng thái khả dĩ. Chính sách có thể là ngẫu nhiên hoặc xác định: chính sách có thể trả về phân phối xác suất hành động hoặc các hành động đơn lẻ cho 1 trạng thái (hoặc quan sát) nhất định. Hiện tại, chúng ta đang làm việc với các chính sách xác định, là 1 bảng tra cứu ánh xạ các hành động với các trạng thái.

In SWF environment, optimal policy is always going right, for every single state. Great, but there are still many unanswered questions. E.g., how much reward should I expect from this policy? Because, even though we know how to act optimally, environment might send our agent backward to hole even if we always select to go toward goal. This is why returns aren't enough. Agent is really looking to maximize expected return; i.e., return taking into account environment's stochasticity.

– Trong môi trường SWF, chính sách tối ưu luôn đúng, cho mọi trạng thái. Tuyệt vời, nhưng vẫn còn nhiều câu hỏi chưa được giải đáp. Ví dụ, tôi nên kỳ vọng bao nhiêu phần thưởng từ chính sách này? Bởi vì, mặc dù chúng ta biết cách hành động tối ưu, môi trường vẫn có thể đẩy tác nhân của chúng ta trở lại điểm xuất phát ngay cả khi chúng ta luôn chọn hướng đến mục tiêu. Đây là lý do tại sao lợi nhuận là không đủ. Tác nhân thực sự muốn tối đa hóa lợi nhuận kỳ vọng; i.e., lợi nhuận có tính đến tính ngẫu nhiên của môi trường.

Also, need a method to automatically find optimal policies, because in FL example, e.g., it isn't obvious at all what optimal policy looks like! There are a few components that are kept internal to agent that can help it find optimal behavior: there are policies, there can be multiple policies for a given environment, & in fact, in certain environments, there may be multiple optimal policies. Also, there are value functions to help us keep track of return estimates. There is a single optimal value function for a given MDP, but there may be multiple value functions in general.

– Ngoài ra, cần có 1 phương pháp để tự động tìm ra các chính sách tối ưu, vì trong ví dụ FL, chẳng hạn, chính sách tối ưu trông như thế nào hoàn toàn không rõ ràng! Có 1 vài thành phần được lưu trữ bên trong tác nhân có thể giúp nó tìm ra hành vi tối ưu: có các chính sách, có thể có nhiều chính sách cho 1 môi trường nhất định, & trên thực tế, trong 1 số môi trường nhất định, có thể có nhiều chính sách tối ưu. Ngoài ra, còn có các hàm giá trị để giúp chúng ta theo dõi ước tính lợi nhuận. Có 1 hàm giá trị tối ưu duy nhất cho 1 MDP nhất định, nhưng nhìn chung có thể có nhiều hàm giá trị.

Look at all components internal to a RL agent that allow them to learn & find optimal policies, with examples to make all of this more concrete.

– Xem xét tất cả các thành phần bên trong tác nhân RL cho phép chúng học & tìm ra các chính sách tối ưu, kèm theo các ví dụ để làm cho tất cả những điều này trở nên cụ thể hơn.

* **Policies: Per-state action prescriptions.** Given stochasticity in Frozen Lake environment (& most RL problems), agent needs to find a policy, denoted as π. A policy is a function that prescribes actions to take for a given nonterminal state. (Remember, policies can be stochastic: either directly over an action or a probability distribution over actions. Expand on stochastic policies in later chaps.). A sample policy:

  – Chính sách: Quy định hành động theo trạng thái. Với tính ngẫu nhiên trong môi trường Hồ Băng (và hầu hết các bài toán thực tế), tác nhân cần tìm 1 chính sách, ký hiệu là π. Chính sách là 1 hàm quy định các hành động cần thực hiện cho 1 trạng thái phi kết thúc nhất định. (Hãy nhớ rằng, chính sách có thể là ngẫu nhiên: trực tiếp trên 1 hành động hoặc phân phối xác suất trên các hành động. Chúng ta sẽ tìm hiểu thêm về chính sách ngẫu nhiên trong các chương sau.) 1 chính sách mẫu:

  A randomly generated policy. A policy generated uniformly at random. Nothing special so far. 1 immediate question question that arises when looking at a policy: how good is this policy? If find a way to put a number to policies, could also ask question, how much better is this policy compared to another policy?

  – 1 chính sách được tạo ngẫu nhiên. 1 chính sách được tạo ngẫu nhiên đồng đều. Cho đến nay không có gì đặc biệt. 1 câu hỏi thường gặp khi xem xét 1 chính sách: chính sách này tốt đến mức nào? Nếu tìm cách đưa ra con số cho các chính sách, cũng có thể đặt câu hỏi, chính sách này tốt hơn bao nhiêu so với chính sách khác?

* **State-value function: What to expect from here?** Something that would help us compare policies: put numbers to states for a given policy. I.e., if we are given a policy & MDP, should be able to calculate expected return starting from every single state (care mostly about START state). How can we calculate how valuable being in a state is? E.g., if our agent is in state 14 (to left of GOAL), how is that better than being in state 13 (to left of 14)? & precisely how much better is it? More importantly, under which policy would we have better results, Go-get-it or Careful policy?

  – Hàm giá trị trạng thái: Kỳ vọng gì ở đây? 1 điều có thể giúp chúng ta so sánh các chính sách: đặt số cho các trạng thái của 1 chính sách nhất định. Ví dụ, nếu chúng ta được cung cấp 1 chính sách & MDP, chúng ta sẽ có thể tính toán lợi nhuận kỳ vọng bắt đầu từ từng trạng thái (chủ yếu quan tâm đến trạng thái START). Làm thế nào chúng ta có thể tính được giá trị của việc ở trong 1 trạng thái? Ví dụ, nếu tác nhân của chúng ta ở trạng thái 14 (bên trái của GOAL), thì trạng thái đó tốt hơn trạng thái 13 (bên trái của 14) như thế nào? & chính xác thì tốt hơn bao nhiêu? Quan trọng hơn, theo chính sách nào chúng ta sẽ có kết quả tốt hơn, chính sách Go-get-it hay chính sách Careful?

What is value of being in state 14 when running Go-get-it policy? So it isn't that straightforward to calculate value of state 14 when following Go-get-it policy because of dependence on values of other states (10 & 14, in this case), which we don't have either. It is like chicken-or-egg problem. Keep going.
– Giá trị của trạng thái 14 khi chạy chính sách Go-get-it là bao nhiêu? Vậy, việc tính toán giá trị của trạng thái 14 khi chạy chính sách Go-get-it không hề đơn giản vì nó phụ thuộc vào giá trị của các trạng thái khác (trong trường hợp này là 10 & 14), mà chúng ta cũng không có. Giống như bài toán con gà hay quả trứng vậy. Cứ tiếp tục.
Defined return as sum of rewards agent obtains from a trajectory. Now, this return can be calculated without paying attention to policy agent is following: sum all of rewards obtained, & you are good to go. But, number we are looking for now is expectation of returns (from state 14) if follow a given policy π. Remember, we are under stochastic environments, so must account for all possible ways environment can react to our policy! That is what an expectation gives us.
– Lợi nhuận được định nghĩa là tổng phần thưởng mà tác nhân nhận được từ 1 quỹ đạo. Bây giờ, lợi nhuận này có thể được tính toán mà không cần chú ý đến chính sách của tác nhân: tổng tất cả phần thưởng nhận được, & bạn đã sẵn sàng. Tuy nhiên, con số chúng ta đang tìm kiếm bây giờ là kỳ vọng lợi nhuận (từ trạng thái 14) nếu tuân theo 1 chính sách π cho trước. Hãy nhớ rằng, chúng ta đang ở trong môi trường ngẫu nhiên, vì vậy phải tính đến tất cả các cách mà môi trường có thể phản ứng với chính sách của chúng ta! Đó chính là những gì kỳ vọng mang lại cho chúng ta.
Now define value of a state s when following a policy π: value of a state s under policy π is expectation of returns if agent follows policy π starting from state s. Calculate this for every state, & get state-value function, or V-function or value function. It represents expected return when following policy π from state s.
– Bây giờ hãy định nghĩa giá trị của trạng thái s khi tuân theo chính sách π: giá trị của trạng thái s theo chính sách π là kỳ vọng lợi nhuận nếu tác nhân tuân theo chính sách π bắt đầu từ trạng thái s. Tính giá trị này cho mọi trạng thái, & lấy hàm giá trị trạng thái, hoặc hàm V hoặc hàm giá trị. Giá trị này biểu thị lợi nhuận kỳ vọng khi tuân theo chính sách π từ trạng thái s.
State-value function V. Value of a state s under policy π is expectation over π of returns at time step t given you select state s at time step t.

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s].$$

Remember: returns are sum of discounted rewards:

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

& can define them recursively like so.

$$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s].$$

This equation is called Bellman equation:

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')],\ \forall s \in S.$$

& it tells us how to find value of states. Get action (or actions, if policy is stochastic) prescribed for state s & do a weighted sum. We also weight sum over probability of next states & rewards. Add reward & discounted value of landing state, then weight that by probability of that transition occurring. Do this for all states in state space.
– **Hàm giá trị trạng thái V.** Giá trị của trạng thái s theo chính sách π là kỳ vọng trên π lợi nhuận tại bước thời gian t với điều kiện bạn chọn trạng thái s tại bước thời gian t.

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s].$$

Lưu ý: lợi nhuận là tổng của các phần thưởng được chiết khấu:

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

& có thể định nghĩa chúng theo cách đệ quy như sau.

$$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s].$$

Phương trình này được gọi là phương trình Bellman:

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')],\ \forall s \in S.$$

& phương trình này cho chúng ta biết cách tìm giá trị của các trạng thái. Hãy lấy hành động (hoặc các hành động, nếu chính sách là ngẫu nhiên) được quy định cho trạng thái s & thực hiện phép tính tổng có trọng số. Chúng ta cũng tính trọng số tổng theo xác suất của các trạng thái tiếp theo & phần thưởng. Cộng phần thưởng & giá trị chiết khấu của trạng thái hạ cánh, sau đó tính trọng số theo xác suất xảy ra quá trình chuyển đổi đó. Thực hiện tương tự cho tất cả các trạng thái trong không gian trạng thái.
These equations are fascinating. A bit of a mess given recursive dependencies, but still interesting. Notice how value of a state depends recursively on value of possibly many other states, which values may also depend on others, including original state!

– Những phương trình này thật hấp dẫn. Tuy có hơi lộn xộn do phụ thuộc đệ quy, nhưng vẫn thú vị. Hãy chú ý cách giá trị của 1 trạng thái phụ thuộc đệ quy vào giá trị của nhiều trạng thái khác, mà những giá trị này cũng có thể phụ thuộc vào các trạng thái khác, bao gồm cả trạng thái ban đầu!

Recursive relationship between states & successive states will come back in next sect when look at algorithms that can iteratively solve these equations & obtain state-value function of any policy in FL environment (or any other environment, really).

– Mối quan hệ đệ quy giữa các trạng thái & các trạng thái liên tiếp sẽ xuất hiện trở lại trong phần tiếp theo khi xem xét các thuật toán có thể giải các phương trình này theo cách lặp đi lặp lại & thu được hàm giá trị trạng thái của bất kỳ chính sách nào trong môi trường FL (hoặc bất kỳ môi trường nào khác).

For now, continue exploring other components commonly found in RL agents. Learn how to calculate these values later in this chap. Note: state-value function is often referred to as value function, or even V-function, or more simply $V^\pi(s)$. It may be confusing, but you will get used to it.

– Hiện tại, hãy tiếp tục khám phá các thành phần khác thường thấy trong các tác nhân RL. Tìm hiểu cách tính các giá trị này ở phần sau của chương này. Lưu ý: hàm trạng thái-giá trị thường được gọi là hàm giá trị, hoặc thậm chí là hàm V, hay đơn giản hơn là $V^\pi(s)$. Có thể hơi khó hiểu, nhưng bạn sẽ quen dần thôi.

∗ Action-value function: What should I expect from here if I do this? Another critical question often needed to task isn't merely about value of a state but value of taking action a in a state s. Differentiating answers to this kind of question will help us decide between actions.

– Hàm hành động-giá trị: Tôi nên mong đợi điều gì nếu tôi làm điều này? 1 câu hỏi quan trọng khác thường cần giải quyết không chỉ là về giá trị của 1 trạng thái mà còn là giá trị của việc thực hiện hành động a trong trạng thái s. Việc phân biệt các câu trả lời cho loại câu hỏi này sẽ giúp chúng ta quyết định giữa các hành động.

E.g., notice: Go-get-it policy goes right when in state 14, but Careful policy goes down. But which action is better? More specifically, which action is better under each policy? I.e., what is value of going down, instead of right, & then following Go-get-it policy, & what is value of going right, instead of down, & then following Careful policy?

– Ví dụ, hãy chú ý: Chính sách Go-get-it đi đúng hướng khi ở trạng thái 14, nhưng chính sách Careful đi xuống. Nhưng hành động nào tốt hơn? Cụ thể hơn, hành động nào tốt hơn trong mỗi chính sách? I.e., giá trị của việc đi xuống, thay vì đi đúng hướng, & sau đó theo chính sách Go-get-it, & giá trị của việc đi đúng hướng, thay vì đi xuống, & sau đó theo chính sách Careful là gì?

By comparing between different actions under same policy, can select better actions, & therefore improve our policies. *Action-value function*, also known as *Q-function* or $Q^\pi(s, a)$, captures precisely this: expected return if agent follows policy π after taking action a in state s.

– Bằng cách so sánh giữa các hành động khác nhau theo cùng 1 chính sách, có thể lựa chọn các hành động tốt hơn, & do đó cải thiện chính sách của chúng ta. *Hàm giá trị hành động*, còn được gọi là *Q-hàm* hoặc $Q^\pi(s, a)$, nắm bắt chính xác điều này: lợi nhuận dự kiến nếu tác nhân tuân theo chính sách π sau khi thực hiện hành động a trong trạng thái s.

In fact, when care about improving policies, which is often referred to as control problem, need action-value functions. Think about it: if don't have an MDP, how can you decide what action to take merely by knowing values of all states? V-functions don't capture dynamics of environment. Q-function, on other hand, does somewhat capture dynamics of environment & allows you to improve policies without need for MDPs. Expand on this fact in later chaps.

– Trên thực tế, khi quan tâm đến việc cải thiện chính sách, thường được gọi là bài toán kiểm soát, cần các hàm hành động-giá trị. Hãy nghĩ xem: nếu không có MDP, làm sao bạn có thể quyết định hành động nào cần thực hiện chỉ bằng cách biết giá trị của tất cả các trạng thái? Hàm V không nắm bắt được động lực của môi trường. Mặt khác, hàm Q phần nào nắm bắt được động lực của môi trường & cho phép bạn cải thiện chính sách mà không cần MDP. Chúng ta sẽ tìm hiểu thêm về điều này trong các chương sau.

Action-value function Q. Value of action a in state s under policy π is expectation of returns, given we select action a in state s & follow policy π thereafter.

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a].$$

Can define this equation recursively like so:

$$q_\pi(s, a) = \mathbb{E}_\pi[R_t + \gamma G_{t+1} | S_t = s,\ A_t = a].$$

Bellman equation for actin values is defined as follows:

$$q_\pi(s, a) = \sum_{s', r} p(s', r|s, a)[r + \gamma v_\pi(s')],\ \forall s \in S,\ \forall a \in A(s).$$

Notice we don't weigh over actions because we are interested only in a specific action. We do weight, however, by probabilities of next states & rewards. What do we weigh? Sum of reward & discounted value of next state. Do that for all state-action pairs.

– **Hàm giá trị hành động Q.** Giá trị của hành động a ở trạng thái s theo chính sách π là kỳ vọng lợi nhuận, với điều kiện ta chọn hành động a ở trạng thái s & sau đó tuân theo chính sách π.

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a].$$

Có thể định nghĩa phương trình này 1 cách đệ quy như sau:

$$q_\pi(s, a) = \mathbb{E}_\pi[R_t + \gamma G_{t+1} | S_t = s,\ A_t = a].$$

Phương trình Bellman cho các giá trị actin được định nghĩa như sau:

$$q_\pi(s,a) = \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')], \forall s \in S, \forall a \in A(s).$$

Lưu ý rằng chúng ta không cân nhắc các hành động vì chúng ta chỉ quan tâm đến 1 hành động cụ thể. Tuy nhiên, chúng ta cân nhắc theo xác suất của các trạng thái tiếp theo & phần thưởng. Chúng ta cân nhắc những gì? Tổng phần thưởng & giá trị chiết khấu của trạng thái tiếp theo. Hãy làm như vậy cho tất cả các cặp trạng thái-hành động.

* **Action-advantage function: How much better if I do that?** Another type of value function is derived from previous 2. *Action-advantage function*, also known as *advantage function, A-function*, or $A^\pi(s,a)$, is difference between action – value function of action a in state s & state-value function of state s under policy π.
– **Hàm lợi thế hành động: Sẽ tốt hơn bao nhiêu nếu tôi làm điều đó?** 1 loại hàm giá trị khác được bắt nguồn từ 2. *Hàm lợi thế hành động*, còn được gọi là *hàm lợi thế, hàm A*, hoặc $A^\pi(s,a)$, là sự khác biệt giữa hàm giá trị hành động của hành động a trong trạng thái s & hàm giá trị trạng thái của trạng thái s theo chính sách π.
Action-advantage function A**.** Advantage of action a in state s under policy π is difference between value of that action & value of state s, both under policy π:

$$a_\pi(s,a) = q_\pi(s,a) - v_\pi(s).$$

Advantage function describes how much better it is to take action instead of following policy π: advantage of choosing action a over default action.
– **Hàm lợi thế hành động** A**.** Lợi thế của hành động a trong trạng thái s theo chính sách π là hiệu số giữa giá trị của hành động đó & giá trị của trạng thái s, cả hai đều theo chính sách π:

$$a_\pi(s,a) = q_\pi(s,a) - v_\pi(s).$$

Hàm lợi thế mô tả việc thực hiện hành động tốt hơn so với việc tuân theo chính sách π: lợi thế của việc chọn hành động a so với hành động mặc định.
Look at different value functions for a (dumb) policy in SWF environment. Remember, these values depend on policy. I.e., $Q_\pi(s,a)$ assumes you will follow policy π (always left in following example & right after taking action a in state s.
– Hãy xem xét các hàm giá trị khác nhau cho 1 chính sách (ngu ngốc) trong môi trường SWF. Hãy nhớ rằng, các giá trị này phụ thuộc vào chính sách. Ví dụ: $Q_\pi(s,a)$ giả định rằng bạn sẽ tuân theo chính sách π (luôn luôn ở bên trái trong ví dụ sau & bên phải sau khi thực hiện hành động a trong trạng thái s).
State-value, action-value, & action-advantage functions. 1. Notice how $Q_\pi(s,a)$ allows us to improve policy π, by showing highest valued action under policy. 2. Also notice there is no advantage for taking same action as policy π recommends.
– Các hàm giá trị trạng thái, giá trị hành động, & lợi thế hành động. 1. Lưu ý cách $Q_\pi(s,a)$ cho phép chúng ta cải thiện chính sách π, bằng cách hiển thị hành động có giá trị cao nhất theo chính sách. 2. Cũng lưu ý rằng không có lợi thế nào khi thực hiện cùng 1 hành động như chính sách π khuyến nghị.

* **Optimality.** Policies, state-value function, action-value functions, & action-advantage functions are components we use to describe, evaluate, & improve behaviors. Call it *optimality* when these components are best they can be.
– **Tối ưu.** Chính sách, hàm giá trị trạng thái, hàm giá trị hành động, & hàm lợi thế hành động là những thành phần chúng ta sử dụng để mô tả, đánh giá, & cải thiện hành vi. Gọi nó là *tối ưu* khi những thành phần này đạt đến mức tốt nhất có thể.
An *optimal policy* is a policy that for every state can obtain expected returns $\geq$ any other policy. An optimal state-value function is a state-value function with maximum value across all policies for all states. Likewise, an optimal action-value function is an action-value function with maximum value across all policies for all state-action pairs. Optimal action-advantage function follows a similar pattern, but notice an optimal advantage function would be ≤ 0 for all state-action pairs, since no action could have any advantage from optimal state-value function.
– Chính sách tối ưu là chính sách mà với mỗi trạng thái, ta có thể đạt được lợi nhuận kỳ vọng $\geq$ so với bất kỳ chính sách nào khác. Hàm giá trị trạng thái tối ưu là hàm giá trị trạng thái có giá trị cực đại trên tất cả các chính sách cho tất cả các trạng thái. Tương tự, hàm giá trị hành động tối ưu là hàm giá trị hành động có giá trị cực đại trên tất cả các chính sách cho tất cả các cặp trạng thái-hành động. Hàm lợi thế hành động tối ưu tuân theo 1 mô hình tương tự, nhưng lưu ý rằng hàm lợi thế tối ưu sẽ là ≤ 0 cho tất cả các cặp trạng thái-hành động, vì không hành động nào có thể có lợi thế từ hàm giá trị trạng thái tối ưu.
Also, notice: although there could be > 1 optimal policy for a given MDP, there can only be 1 optimal state-value function, optimal action-value function, & optimal action-advantage function.
– Ngoài ra, hãy lưu ý: mặc dù có thể có > 1 chính sách tối ưu cho 1 MDP nhất định, nhưng chỉ có thể có 1 hàm giá trị trạng thái tối ưu, hàm giá trị hành động tối ưu & hàm lợi thế hành động tối ưu.
May also notice: if you had the optimal V-function, could use MDP to do a 1-step search for optimal Q-function & then use this to build optimal policy. On other hand, if had optimal Q-function, don't need MDP at all. Could use optimal Q-function to find optimal V-function by merely taking maximum over actions. & could obtain optimal policy using optimal Q-function by taking argmax over actions.
– Cũng có thể lưu ý: nếu bạn có hàm V tối ưu, bạn có thể sử dụng MDP để tìm kiếm hàm Q tối ưu theo từng bước & sau đó sử dụng hàm này để xây dựng chính sách tối ưu. Mặt khác, nếu đã có hàm Q tối ưu, bạn hoàn toàn không cần

MDP. Bạn có thể sử dụng hàm Q tối ưu để tìm hàm V tối ưu chỉ bằng cách lấy cực đại trên các hành động. & có thể thu được chính sách tối ưu bằng hàm Q tối ưu bằng cách lấy argmax trên các hành động.

Bellman optimality equations. Optimal state-value function is state-value function with highest value across all policies.

$$v_*(s) = \max_\pi v_\pi(s),\ \forall s \in S.$$

Likewise, optimal action-value function is action-value function with highest values.

$$q_*(s,a) = \max_\pi q_\pi(s,a),\ \forall s \in S,\ \forall a \in A(s).$$

Optimal state-value function can be obtained this way.

$$v_*(s) = \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_*(s')].$$

Take max action of weighted sum of reward & discounted optimal value of next state. Similarly, optimal action-value function can be obtained this way.

$$q_*(s,a) = \sum_{s',r} p(s',r|s,a)[r + \gamma \max_{a'} q_*(s',a')].$$

Notice how max is now on inside.

– **Phương trình tối ưu Bellman.** Hàm giá trị trạng thái tối ưu là hàm giá trị trạng thái có giá trị cao nhất trong tất cả các chính sách.

$$v_*(s) = \max_\pi v_\pi(s), \forall s \in S.$$

Tương tự, hàm giá trị hành động tối ưu là hàm giá trị hành động có giá trị cao nhất.

$$q_*(s,a) = \max_\pi q_\pi(s,a), \forall s \in S, \forall a \in A(s).$$

Hàm giá trị trạng thái tối ưu có thể được tính theo cách này.

$$v_*(s) = \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_*(s')].$$

Lấy hành động tối đa của tổng phần thưởng có trọng số & giá trị tối ưu chiết khấu của trạng thái tiếp theo. Tương tự, hàm giá trị hành động tối ưu có thể được tính theo cách này.

$$q_*(s,a) = \sum_{s',r} p(s',r|s,a)[r + \gamma \max_{a'} q_*(s',a')].$$

Lưu ý rằng giá trị tối đa hiện đang ở bên trong.

- **3.2. Planning optimal sequences of actions.** Have state-value functions to keep track of values of states, action-value functions to keep track of values of state-action pairs, & action-advantage functions to show "advantage" of taking specific actions. Have equations for all of these to evaluate current policies, i.e., to go from policies to value functions, & to calculate & find optimal value functions &, therefore, optimal policies.

  – **Lập kế hoạch chuỗi hành động tối ưu.** Có các hàm trạng thái-giá trị để theo dõi giá trị của các trạng thái, hàm hành động-giá trị để theo dõi giá trị của các cặp trạng thái-hành động, & hàm hành động-lợi thế để thể hiện "lợi thế" của việc thực hiện các hành động cụ thể. Có các phương trình cho tất cả những điều này để đánh giá các chính sách hiện tại, i.e., chuyển từ chính sách sang hàm giá trị, & để tính toán & tìm các hàm giá trị tối ưu &, từ đó tìm ra các chính sách tối ưu.

  Now have discussed RL problem formulation, & have defined objective we are after, can start exploring methods for finding this objective. Iteratively computing equations presented in previous sect is 1 of most common ways to solve a RL problem & obtain optimal policies when dynamics of environment, MDPs, are known. Look at methods.

  – Bây giờ chúng ta đã thảo luận về việc xây dựng bài toán RL, & đã xác định mục tiêu chúng ta đang theo đuổi, có thể bắt đầu khám phá các phương pháp để tìm ra mục tiêu này. Tính toán lặp các phương trình được trình bày trong phần trước là 1 trong những cách phổ biến nhất để giải bài toán RL & thu được các chính sách tối ưu khi đã biết động lực của môi trường, MDP. Hãy xem xét các phương pháp.

  * **Policy evaluation: Rating policies.** Talked about comparing policies in prev sect. Established: policy π is better than or equal to policy π' if expected return is better than or equal to π' for all states. Before can use this definition, however, must devise an algorithm for evaluating an arbitrary policy. Such an algorithm is known as an *iterative policy evaluation* or just *policy evaluation*.

    – **Đánh giá chính sách: Xếp hạng chính sách.** Đã nói về việc so sánh các chính sách trong phần trước. Đã xác định: chính sách π tốt hơn hoặc bằng chính sách π' nếu lợi nhuận kỳ vọng tốt hơn hoặc bằng π' cho mọi trạng thái. Tuy nhiên, trước khi có thể sử dụng định nghĩa này, cần phải thiết kế 1 thuật toán để đánh giá 1 chính sách tùy ý. Thuật toán như vậy được gọi là *đánh giá chính sách lặp* hoặc đơn giản là *đánh giá chính sách*.

Policy-evaluation algorithm consists of calculating V-function for a given policy by sweeping through state space & iteratively improving estimates. Refer to type of algorithm that takes in a policy & outputs a value function as an algorithm that solves *prediction problem*, which is calculating values of a predetermined policy.
– Thuật toán đánh giá chính sách bao gồm việc tính toán hàm V cho 1 chính sách nhất định bằng cách quét qua không gian trạng thái & cải thiện ước tính theo từng bước. Thuật toán này được gọi là thuật toán tiếp nhận chính sách & đưa ra 1 hàm giá trị, được coi là thuật toán giải quyết *prediction problem*, i.e., tính toán giá trị của 1 chính sách được xác định trước.
Policy-evaluation equation. 1. Policy-evaluation algorithm consist of iterative approximation of state-value function of policy under evaluation. Algorithm converges as $k \to \infty$. 2. Initialize $v_0(s)$ for all s in s arbitrarily, & to 0 if s is terminal. Then, increase k & iteratively improve estimates by following equation below.

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')].$$

Calculate value of a state s as weighted sum of reward & discounted estimated value of next state s'.
– **Phương trình đánh giá chính sách.** 1. Thuật toán đánh giá chính sách bao gồm phép xấp xỉ lặp của hàm giá trị trạng thái của chính sách đang được đánh giá. Thuật toán hội tụ khi $k \to \infty$. 2. Khởi tạo $v_0(s)$ cho mọi s trong s 1 cách tùy ý, & bằng 0 nếu s là kết thúc. Sau đó, tăng k & lặp lại để cải thiện ước tính bằng cách làm theo phương trình dưới đây.

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')].$$

Tính giá trị của trạng thái s là tổng trọng số của phần thưởng & giá trị ước tính chiết khấu của trạng thái s' tiếp theo. Using this equation, can iteratively approximate true V-function of an arbitrary policy. Iterative policy-evaluation algorithm is guaranteed to converge to value function of policy if given enough iterations, more concretely as we approach ∞. In practice, however, use a small threshold to check for changes in value function we are approximating. Once changes in value function are less than this threshold, we stop. See how this algorithm works in SWF environment, for always-left policy.
– Sử dụng phương trình này, ta có thể xấp xỉ lặp lại hàm V thực của 1 chính sách bất kỳ. Thuật toán đánh giá chính sách lặp được đảm bảo hội tụ về hàm giá trị của chính sách nếu được lặp lại đủ nhiều lần, cụ thể hơn khi ta tiến gần đến ∞. Tuy nhiên, trên thực tế, hãy sử dụng 1 ngưỡng nhỏ để kiểm tra sự thay đổi của hàm giá trị mà ta đang xấp xỉ. Khi sự thay đổi của hàm giá trị nhỏ hơn ngưỡng này, ta sẽ dừng lại. Xem cách thuật toán này hoạt động trong môi trường SWF, đối với chính sách luôn trái.
Initial calculations of policy evaluation.

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')].$$

Have a deterministic policy, so $\sum_a \pi(a|s) = 1$. Use gamma of 1. An Always-left policy. State 5, Iteration 1 (initialized to 0 in iteration 0) $v_1^\pi(5)$. Then calculate values for all states 0–6, & when done, move to next iteration. Notice: to calculate $V_2^\pi(s)$ you would have to use estimates obtained in previous iteration $V_1^\pi(s)$. This technique of calculating an estimate from an estimate is referred to as *bootstrapping*, & it's a widely used technique in RL (including DRL).
– Các tính toán ban đầu cho việc đánh giá chính sách.

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')].$$

Có 1 chính sách xác định, vì vậy $\sum_a \pi(a|s) = 1$. Sử dụng gamma bằng 1. 1 chính sách luôn trái. Trạng thái 5, Lần lặp 1 (được khởi tạo thành 0 trong lần lặp 0) $v_1^\pi(5)$. Sau đó, tính toán các giá trị cho tất cả các trạng thái 0–6, & khi hoàn tất, chuyển sang lần lặp tiếp theo. Lưu ý: để tính $V_2^\pi(s)$, bạn sẽ phải sử dụng các ước tính thu được từ lần lặp trước đó $V_1^\pi(s)$. Kỹ thuật tính toán ước tính từ ước tính này được gọi là *bootstrapping*, & đây là kỹ thuật được sử dụng rộng rãi trong RL (bao gồm cả DRL).
Also, important to notice: k's here are iterations across estimates, but they are not interactions with environment. These aren't episodes that agent is out & about selecting actions & observing environment. These aren't time steps either. Instead, these are iterations of iterative policy-evaluation algorithm. Do a couple more of these estimates. Following table shows you results you should get.
– Ngoài ra, điều quan trọng cần lưu ý: k ở đây là các lần lặp lại giữa các ước tính, nhưng chúng không phải là tương tác với môi trường. Đây không phải là các tập mà tác nhân đang & lựa chọn hành động & quan sát môi trường. Đây cũng không phải là các bước thời gian. Thay vào đó, đây là các lần lặp lại của thuật toán đánh giá chính sách lặp lại. Hãy thực hiện thêm 1 vài ước tính như vậy. Bảng sau đây sẽ hiển thị kết quả bạn sẽ nhận được.
What are several of things resulting state-value function tells us? To begin with, can say get a return of 0.0357 in expectation when starting an episode in this environment & following always-left policy. Pretty low. Can also say, even when find ourselves in state 1 (leftmost non-terminal state), still have a chance, albeit $< 1\%$, to end up in GOAL cell (state 6). To be exact, have a 0.27% chance of ending up in GOAL state when we are in state 1. & select left all time! Pretty interesting.

– Hàm giá trị trạng thái kết quả cho chúng ta biết 1 số điều gì? Trước hết, có thể nói rằng ta sẽ nhận được giá trị kỳ vọng là 0,0357 khi bắt đầu 1 tập phim trong môi trường này & tuân theo chính sách luôn luôn sang trái. Khá thấp. Cũng có thể nói rằng, ngay cả khi ta ở trạng thái 1 (trạng thái ngoài cùng bên trái không phải trạng thái kết thúc), vẫn có cơ hội, mặc dù $< 1\%$, để kết thúc ở ô MỤC TIÊU (trạng thái 6). Chính xác hơn, ta có 0,27% cơ hội kết thúc ở trạng thái MỤC TIÊU khi ta ở trạng thái 1. & luôn chọn sang trái! Thật thú vị.

Interestingly also, due to stochasticity of this environment, have a 3.57% chance of reaching GOAL cell (remember this environment has 50% action success, 33.33% no effects, & 16.66% backward). Again, this is when under an always-left policy. Still, Left action could send us right, then right & right again, or left, right, right, right, right, & so on.

– Điều thú vị nữa là, do tính ngẫu nhiên của môi trường này, chúng ta có 3,57% cơ hội đạt đến ô MỤC TIÊU (hãy nhớ rằng môi trường này có 50% hành động thành công, 33,33% không có hiệu ứng, & 16,66% ngược lại). 1 lần nữa, điều này xảy ra khi áp dụng chính sách luôn sang trái. Tuy nhiên, hành động sang trái có thể đưa chúng ta sang phải, rồi sang phải & sang phải lần nữa, hoặc sang trái, phải, phải, phải, phải, & cứ thế.

Think about how probabilities of trajectories combine. Also, pay attention to iterations & how values propagate backward from reward (transition from state 5 to state 6) 1 step at a time. This backward propagation of values is a common characteristic among RL algorithms & comes up again several times.

– Hãy nghĩ về cách xác suất của các quỹ đạo kết hợp. Ngoài ra, hãy chú ý đến các lần lặp & cách các giá trị lan truyền ngược từ phần thưởng (chuyển từ trạng thái 5 sang trạng thái 6) từng bước một. Sự lan truyền ngược này của các giá trị là 1 đặc điểm chung giữa các thuật toán RL & xuất hiện lại nhiều lần.

Policy-evaluation algorithm. Now run policy evaluation in randomly generated policy presented earlier for FL environment. Recall randomly generated policy. A policy generated randomly is the same as before. No need to flip pages!

– **Thuật toán đánh giá chính sách.** Bây giờ hãy chạy đánh giá chính sách trong chính sách được tạo ngẫu nhiên đã trình bày trước đó cho môi trường FL. Gọi lại chính sách được tạo ngẫu nhiên. Chính sách được tạo ngẫu nhiên vẫn giống như trước. Không cần lật trang!

Following shows progress policy evaluation makes on accurately estimating state-value function of randomly generated policy after only 8 iterations. Policy evaluation on randomly generated policy for FL environment. 1. Values start propagating with every iteration. 2. Values continue to propagate & become more & more accurate.

– Sau đây là tiến trình đánh giá chính sách dựa trên việc ước tính chính xác hàm giá trị trạng thái của chính sách được tạo ngẫu nhiên chỉ sau 8 lần lặp. Đánh giá chính sách trên chính sách được tạo ngẫu nhiên cho môi trường FL. 1. Các giá trị bắt đầu lan truyền sau mỗi lần lặp. 2. Các giá trị tiếp tục lan truyền & trở nên & chính xác hơn.

State-value function of randomly generated policy. After 218 interactions, policy evaluation converges to these values (using a 1e-10 minimum change in values as a stopping condition). This final state-value function is state-value function for this policy. Note: even though this is still an estimate, because we are in a discrete state & action spaces, can assume this to be actual value function when using gamma of 0.99. In case you are wondering about state-value functions of 2 policies presented earlier, here are results.

– Hàm giá trị trạng thái của chính sách được tạo ngẫu nhiên. Sau 218 tương tác, quá trình đánh giá chính sách hội tụ về các giá trị này (sử dụng mức thay đổi giá trị tối thiểu 1e-10 làm điều kiện dừng). Hàm giá trị trạng thái cuối cùng này là hàm giá trị trạng thái của chính sách này. Lưu ý: mặc dù đây vẫn là ước tính, nhưng vì chúng ta đang ở trong không gian trạng thái rời rạc & hành động, nên có thể coi đây là hàm giá trị thực khi sử dụng gamma bằng 0,99. Trong trường hợp bạn đang thắc mắc về hàm giá trị trạng thái của 2 chính sách đã trình bày trước đó, đây là kết quả.

Results of policy evolution. 1. Go-get-it-policy: State-value function of this policy converges after 66 iterations. Policy reaches goal state a mere 3.4% of time. 2. For Careful policy: state-value function converges after 546 iterations. Policy reaches goal 53.7% of time. Calculated these values empirically by running policies 100 times. Therefore, these values are noisy, but you get idea. It seems being a Go-get-it policy doesn't pay well in FL environment! Fascinating results. But a question arises: Are there any better policies for this environment?

– Kết quả của quá trình tiến hóa chính sách. 1. Go-get-it-policy: Hàm giá trị trạng thái của chính sách này hội tụ sau 66 lần lặp. Chính sách đạt đến trạng thái mục tiêu chỉ trong 3,4% thời gian. 2. Đối với chính sách Cẩn thận: hàm giá trị trạng thái hội tụ sau 546 lần lặp. Chính sách đạt đến mục tiêu trong 53,7% thời gian. Đã tính toán các giá trị này theo kinh nghiệm bằng cách chạy chính sách 100 lần. Do đó, các giá trị này có nhiễu, nhưng bạn hiểu ý tôi rồi đấy. Có vẻ như chính sách Go-get-it không mang lại hiệu quả cao trong môi trường FL! Kết quả thật thú vị. Nhưng 1 câu hỏi được đặt ra: Có chính sách nào tốt hơn cho môi trường này không?

∗ **Policy improvement: Using ratings to get better.** Motivation is clear now. Have a way of evaluating any policy. This already gives you some freedom: can evaluate many policies & rank them by state-value function of START state. After all, that number tells you expected cumulative reward policy in question will obtain if you run many episodes. Cool, right? No! Makes no sense. Why would you randomly generate a bunch of policies & evaluate them all? 1st, that's a total waste of computing resources, but more importantly, it gives you no guarantee that you are finding better & better policies. There has to be a better way.

– **Cải thiện chính sách: Sử dụng xếp hạng để cải thiện.** Động lực giờ đã rõ ràng. Hãy có cách đánh giá bất kỳ chính sách nào. Điều này đã mang lại cho bạn 1 chút tự do: có thể đánh giá nhiều chính sách & xếp hạng chúng theo hàm giá trị trạng thái của trạng thái BẮT ĐẦU. Xét cho cùng, con số đó cho bạn biết chính sách phần thưởng tích lũy dự kiến sẽ đạt được nếu bạn chạy nhiều tập. Tuyệt vời, phải không? Không! Vô lý. Tại sao bạn lại ngẫu nhiên tạo ra 1 loạt chính sách & đánh giá tất cả chúng? Thứ nhất, đó là 1 sự lãng phí hoàn toàn tài nguyên máy tính, nhưng quan trọng hơn, nó không đảm bảo bạn sẽ tìm thấy những chính sách tốt hơn & tốt hơn. Phải có 1 cách tốt hơn.

Key to unlocking this problem is action-value function, Q-function. Using V-function & MDP, get an estimate of Q-

function. Q-function will give you a glimpse of values of all actions for all states, & these values, in turn, can hint at how to improve policies. Take a look at Q-function of Careful policy & ways we can improve this policy: How can Q-function help us improve policies? 1. This is careful policy. 2. Action-value function of careful policy. 3. Greedy policy over careful Q-function. 4. Calling this new policy careful+.
– Chìa khóa để giải quyết vấn đề này là hàm hành động-giá trị, hàm Q. Sử dụng hàm V & MDP, hãy ước tính hàm Q. Hàm Q sẽ cho bạn cái nhìn thoáng qua về giá trị của tất cả hành động cho tất cả trạng thái, & các giá trị này, đến lượt nó, có thể gợi ý cách cải thiện chính sách. Hãy xem hàm Q của chính sách Cẩn thận & các cách chúng ta có thể cải thiện chính sách này: Hàm Q có thể giúp chúng ta cải thiện chính sách như thế nào? 1. Đây là chính sách cẩn thận. 2. Hàm hành động-giá trị của chính sách cẩn thận. 3. Chính sách tham lam hơn hàm Q cẩn thận. 4. Gọi chính sách mới này là cẩn thận+.
Notice how if we act greedily w.r.t. Q-function of policy, obtain a new policy: Careful+. Is this policy any better? Well, policy evaluation can tell us. Find out: State-value function of Careful policy. 1. After 574 iterations policy evaluation converges to this state-value function for Careful+ policy. 2. This is difference between Careful+ & Careful V-functions. What an improvement! 3. This new policy, Careful+ can reach goal state 73.2% of time. An improvement! 4. Also empirically.
– Lưu ý rằng nếu chúng ta hành động 1 cách tham lam đối với hàm Q của chính sách, ta sẽ có được 1 chính sách mới: Careful+. Chính sách này có tốt hơn không? Việc đánh giá chính sách có thể cho chúng ta biết. Tìm hiểu: Hàm giá trị trạng thái của chính sách Careful. 1. Sau 574 lần lặp, việc đánh giá chính sách hội tụ về hàm giá trị trạng thái này cho chính sách Careful+. 2. Đây là sự khác biệt giữa hàm Careful+ & Careful V. Thật là 1 cải tiến! 3. Chính sách mới này, Careful+, có thể đạt trạng thái mục tiêu 73,2% thời gian. 1 cải tiến! 4. Cũng theo kinh nghiệm.
New policy is better than original policy. This is great. Used state-value function of original policy & MDP to calculate its action-value function. Then, acting greedily w.r.t. action-value function gave us an improved policy. This is what *policy-improvement* algorithm does: it calculates an action-value function using state-value function & MDP, & it returns a *greedy* policy w.r.t. action-value function of original policy. Let that sink in, it is pretty important.
– Chính sách mới tốt hơn chính sách gốc. Thật tuyệt vời. Đã sử dụng hàm giá trị trạng thái của chính sách gốc & MDP để tính toán hàm giá trị hành động của nó. Sau đó, hành động tham lam đối với hàm giá trị hành động đã cho chúng ta 1 chính sách được cải thiện. Đây chính là những gì thuật toán *policy-improvement* thực hiện: nó tính toán 1 hàm giá trị hành động bằng cách sử dụng hàm giá trị trạng thái & MDP, & nó trả về 1 chính sách *tham lam* đối với hàm giá trị hành động của chính sách gốc. Hãy suy nghĩ kỹ nhé, nó khá quan trọng đấy.
Policy-improvement equation. 1. To improve a policy, use a state-value function & an MDP to get a 1-step look-ahead & determine which of actions lead to highest value. This is policy-improvement equation.

$$\pi'(s) = \arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')].$$

2. Obtain a new policy π' by taking highest-valued action. 3. how do we get highest-valued action? 4. By calculating, for each action, weighted sum of all rewards & values of all possible next states. 5. Notice: this uses action with highest-valued Q-function.
– **Phương trình cải thiện chính sách.** 1. Để cải thiện chính sách, hãy sử dụng hàm giá trị trạng thái & MDP để có được dự đoán trước 1 bước & xác định hành động nào dẫn đến giá trị cao nhất. Đây là phương trình cải thiện chính sách.

$$\pi'(s) = \arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')].$$

2. Thu được 1 chính sách π' mới bằng cách thực hiện hành động có giá trị cao nhất. 3. Làm thế nào để có được hành động có giá trị cao nhất? 4. Bằng cách tính toán, đối với mỗi hành động, tổng có trọng số của tất cả phần thưởng & giá trị của tất cả các trạng thái tiếp theo có thể xảy ra. 5. Lưu ý: phương pháp này sử dụng hành động có hàm Q có giá trị cao nhất.
This is how policy-improvement algorithm looks in Python. Natural next questions are these: Is there a better policy than this one? Can we do any better than Careful+? Can we evaluate Careful+ policy, & then improve it again? Maybe! Can we improve over Careful+ policy? 1. This is Careful+ policy. 2. Action-value function of Careful+ policy. 3. Greedy policy over Careful+ Q-function. 4. Notice, greedy policy is the same as original policy. There is no improvement now.
– Thuật toán cải tiến chính sách trong Python trông như thế này. Những câu hỏi tiếp theo tự nhiên sẽ là: Có chính sách nào tốt hơn chính sách này không? Chúng ta có thể làm tốt hơn Careful+ không? Chúng ta có thể đánh giá chính sách Careful+, & rồi cải tiến nó lần nữa không? Có thể! Chúng ta có thể cải tiến hơn chính sách Careful+ không? 1. Đây là chính sách Careful+. 2. Hàm giá trị hành động của chính sách Careful+. 3. Chính sách tham lam so với hàm Q của Careful+. 4. Lưu ý, chính sách tham lam giống hệt chính sách gốc. Hiện tại không có cải tiến nào.
Ran policy evaluation on Careful+ policy, & then policy improvement. Q-functions of Careful & Careful+ are different, but greedy policies over Q-functions are identical. I.e., there is no improvement this time. No improvement occurs because Careful+ policy is an optimal policy of FL environment (with gamma 0.99). Only needed 1 improvement over Careful policy because this policy was good to begin with.
– Đánh giá chính sách trên chính sách Careful+, sau đó cải thiện chính sách. Các hàm Q của Careful & Careful+ khác nhau, nhưng các chính sách tham lam trên các hàm Q thì giống hệt nhau. I.e., lần này không có cải thiện nào. Không có cải thiện nào xảy ra vì chính sách Careful+ là chính sách tối ưu của môi trường FL (với gamma 0,99). Chỉ cần cải thiện 1 lần so với chính sách Careful vì chính sách này đã tốt ngay từ đầu.

Now, even if start with an adversarial policy designed to perform poorly, alternating over policy evaluation & improvement would still end up with an optimal policy. Want proof? Make up an adversarial policy for FL environment & see what happens. Adversarial policy for FL environment. 1. This policy is so mean that agent has 0% chance of reaching GOAL. 2. It has a state-value function of 0 for all states. Mean!

– Bây giờ, ngay cả khi bắt đầu với 1 chính sách đối kháng được thiết kế để hoạt động kém, việc luân phiên đánh giá chính sách & cải thiện vẫn sẽ cho ra 1 chính sách tối ưu. Muốn chứng minh không? Hãy tạo 1 chính sách đối kháng cho môi trường FL & xem điều gì xảy ra. Chính sách đối kháng cho môi trường FL. 1. Chính sách này quá tệ đến mức tác nhân có 0% cơ hội đạt được MỤC TIÊU. 2. Nó có hàm giá trị trạng thái bằng 0 cho tất cả các trạng thái.

* Policy iteration: Improving upon improved behaviors. Plan with this adversarial policy: alternate between policy evaluation & policy improvement until policy coming out of policy-improvement phase no longer yields a different policy. Fact: if instead of starting with an adversarial policy, start with a randomly generated policy, this is what an algorithm called *policy iteration* does.

– Lặp lại chính sách: Cải thiện dựa trên các hành vi đã được cải thiện. Lập kế hoạch với chính sách đối kháng này: luân phiên giữa đánh giá chính sách & cải thiện chính sách cho đến khi chính sách thoát khỏi giai đoạn cải thiện chính sách không còn tạo ra chính sách khác. Thực tế: nếu thay vì bắt đầu bằng 1 chính sách đối kháng, hãy bắt đầu bằng 1 chính sách được tạo ngẫu nhiên, thì đây chính là điều mà thuật toán có tên là *policy iteration* thực hiện.

1st try it starting with adversarial policy & see what happens. Improving upon adversarial policy. Alternating policy evaluating & policy improvement yields an optimal policy & state-value function regardless of policy you start with.

– Đầu tiên hãy thử bắt đầu với chính sách đối đầu & xem điều gì xảy ra. Cải thiện chính sách đối đầu. Đánh giá chính sách xen kẽ & cải tiến chính sách sẽ tạo ra 1 hàm giá trị trạng thái & chính sách tối ưu bất kể bạn bắt đầu với chính sách nào.

Notice how I use "*an* optimal policy", but also use "*the* optimal state-value function". This is not a coincidence or a poor choice of words. An MDP can have more than 1 optimal policy, but it can only have a single optimal state-value function.

– Lưu ý cách tôi sử dụng "*an* optimal policy", nhưng cũng sử dụng "*the* optimal state-value function". Đây không phải là sự trùng hợp ngẫu nhiên hay lựa chọn từ ngữ kém. 1 MDP có thể có nhiều hơn 1 chính sách tối ưu, nhưng nó chỉ có thể có 1 hàm trạng thái-giá trị tối ưu duy nhất.

State-value functions are collections of numbers. Numbers can have infinitesimal accuracy, because they are numbers. There will be only 1 optimal state-value function (collection with highest numbers for all states). However, a state-value function may have actions that are equally valued for a given state; this includes optimal state-value function. In this case, there could be multiple optimal policies, each optimal policy setting a different, but equally valued, action. Take a look: FL environment is a great example of this.

– Các hàm giá trị trạng thái là tập hợp các số. Các số có thể có độ chính xác vô cùng nhỏ, vì chúng là số. Sẽ chỉ có 1 hàm giá trị trạng thái tối ưu (tập hợp có số cao nhất cho tất cả các trạng thái). Tuy nhiên, 1 hàm giá trị trạng thái có thể có các hành động có giá trị như nhau cho 1 trạng thái nhất định; điều này bao gồm hàm giá trị trạng thái tối ưu. Trong trường hợp này, có thể có nhiều chính sách tối ưu, mỗi chính sách tối ưu thiết lập 1 hành động khác nhau nhưng có giá trị như nhau. Hãy xem: Môi trường FL là 1 ví dụ điển hình về điều này.

FL environment has multiple optimal policies. 1. Optimal action-value function. 2. A policy going left in state 6 is optimal. 4. Here is a policy that goes right in state 6, & it's as good, & also optimal. Btw, not shown here, but all actions in a terminal state have same value, 0, & therefore a similar issue that I am highlighting in state 6.

– Môi trường FL có nhiều chính sách tối ưu. 1. Hàm giá trị hành động tối ưu. 2. 1 chính sách đi sang trái ở trạng thái 6 là tối ưu. 4. Đây là 1 chính sách đi sang phải ở trạng thái 6, & nó cũng tốt, & cũng tối ưu. Nhân tiện, không được hiển thị ở đây, nhưng tất cả các hành động ở trạng thái kết thúc đều có cùng giá trị, 0, & do đó, 1 vấn đề tương tự mà tôi đang nêu bật ở trạng thái 6.

Want to highlight: policy iteration is guaranteed to converge to exact optimal policy: mathematical proof shows it will not get stuck in local optima. However, as a practical consideration, there is 1 thing to be careful about. If action-value function has a tie (e.g., right/left in state 6), must make sure not to break ties randomly. Otherwise, policy improvement could keep returning different policies, even without any real improvement. With that out of way, look at another essential algorithm for finding optimal state-value functions & optimal policies.

– Cần nhấn mạnh: phép lặp chính sách được đảm bảo hội tụ đến chính sách tối ưu chính xác: bằng chứng toán học cho thấy nó sẽ không bị kẹt trong tối ưu cục bộ. Tuy nhiên, trên thực tế, có 1 điều cần lưu ý. Nếu hàm hành động-giá trị có 1 điểm hòa (ví dụ: phải/trái ở trạng thái 6), phải đảm bảo không phá vỡ điểm hòa 1 cách ngẫu nhiên. Nếu không, việc cải thiện chính sách có thể tiếp tục trả về các chính sách khác nhau, ngay cả khi không có bất kỳ cải thiện thực sự nào. Sau khi đã giải quyết vấn đề đó, hãy xem xét 1 thuật toán thiết yếu khác để tìm hàm trạng thái-giá trị tối ưu & chính sách tối ưu.

* **Value iteration: Improving behaviors early.** Probably notice way policy evaluation works: values propagate consistently on each iteration, but slowly. Policy evaluation on always-left policy on SWF environment. 1. Calculating Q-function after each state sweep. 2. See how even after 1st iteration greedy policy over Q-function was already a different & better policy. 3. Fully converged state-value function for always-left policy.

– **Lặp lại giá trị: Cải thiện hành vi sớm.** Có thể bạn sẽ thấy cách đánh giá chính sách hoạt động: các giá trị lan truyền nhất quán ở mỗi lần lặp, nhưng chậm. Đánh giá chính sách về chính sách luôn trái trên môi trường SWF. 1. Tính toán hàm Q sau mỗi lần quét trạng thái. 2. Xem ngay cả sau lần lặp đầu tiên, chính sách tham lam so với hàm Q đã là 1 chính sách khác biệt & tốt hơn. 3. Hàm giá trị trạng thái hội tụ hoàn toàn cho chính sách luôn trái.

Image shows a single state-space sweep of policy evaluation followed by an estimation of Q-function. Do this by using truncated estimate of V-function & MDP, on each iteration. By doing so, can more easily see: even after 1st iteration,

a greedy policy over early Q-function estimates would be an improvement. Look at Q-values for state 5 in 1st iteration; changing action to point towards GOAL state is obvious already better.
– Hình ảnh cho thấy 1 lần quét không gian trạng thái duy nhất của quá trình đánh giá chính sách, tiếp theo là ước tính hàm Q. Thực hiện việc này bằng cách sử dụng ước tính hàm V bị cắt cụt & MDP, trên mỗi lần lặp. Bằng cách này, có thể dễ dàng thấy rằng: ngay cả sau lần lặp đầu tiên, 1 chính sách tham lam so với các ước tính hàm Q ban đầu sẽ là 1 cải tiến. Hãy xem xét các giá trị Q cho trạng thái 5 trong lần lặp đầu tiên; việc thay đổi hành động để hướng đến trạng thái MỤC TIÊU rõ ràng đã tốt hơn.
I.e., even if we truncated policy evaluation after a single iteration, could still improve upon initial policy by taking greedy policy of Q-function estimation after a single state-space sweep of policy evaluation. This algorithm is another fundamental algorithm in RL, called *value iteration* (VI).
– I.e., ngay cả khi chúng ta cắt ngắn quá trình đánh giá chính sách sau 1 lần lặp duy nhất, vẫn có thể cải thiện chính sách ban đầu bằng cách sử dụng chính sách tham lam của ước lượng hàm Q sau 1 lần quét không gian trạng thái của quá trình đánh giá chính sách. Thuật toán này là 1 thuật toán cơ bản khác trong RL, được gọi là lặp giá trị (VI).
VI can be thought of "greedy greedifying policies", because we calculate greedy policy as soon as we can, greedily. VI doesn't wait until we have an accurate estimate of policy before it improves it, but instead, VI truncates policy-evaluation phase after a single state sweep. Take a look at what I mean by "greedily greedifying policies." Greedily greedifying always-left policy of SFW environment. This is optimal action-value function & optimal policy.
– VI có thể được hiểu là "các chính sách tham lam hóa", bởi vì chúng ta tính toán chính sách tham lam càng sớm càng tốt, 1 cách tham lam. VI không đợi đến khi chúng ta có ước tính chính xác về chính sách mới cải thiện nó, mà thay vào đó, VI cắt ngắn giai đoạn đánh giá chính sách sau 1 lần quét trạng thái duy nhất. Hãy xem ý tôi khi nói "các chính sách tham lam hóa" là gì. Chính sách luôn luôn tham lam hóa của môi trường SFW. Đây là hàm giá trị hành động tối ưu & chính sách tối ưu.
If start with a randomly generated policy, instead of this adversarial policy always-left for SWF environment, VI would still converge to optimal state-value function. VI is a straightforward algorithm that can be expressed in a single equation.
– Nếu bắt đầu bằng 1 chính sách được tạo ngẫu nhiên, thay vì chính sách đối nghịch này luôn được giữ lại cho môi trường SWF, VI vẫn sẽ hội tụ về hàm giá trị trạng thái tối ưu. VI là 1 thuật toán đơn giản có thể được biểu diễn bằng 1 phương trình duy nhất.
Value-iteration equation. 1. Can merge a truncated policy-evaluation step & a policy improvement into same equation. 2. Calculate value of each action using sum of weighted sum of reward & discounted estimated value of next states.

$$v_{k+1}(s) = \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')].$$

5. Multiply by probability of each possible transition. 6. & add for all transitions in action. 7. Then, take max over values of actions.
– **Phương trình lặp giá trị.** 1. Có thể hợp nhất 1 bước đánh giá chính sách bị cắt ngắn & 1 bước cải tiến chính sách vào cùng 1 phương trình. 2. Tính giá trị của mỗi hành động bằng cách sử dụng tổng của tổng phần thưởng có trọng số & giá trị ước tính chiết khấu của các trạng thái tiếp theo.

$$v_{k+1}(s) = \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')].$$

5. Nhân với xác suất của mỗi chuyển đổi có thể xảy ra. 6. & cộng cho tất cả các chuyển đổi đang diễn ra. 7. Sau đó, lấy giá trị max trên các giá trị của các hành động.
In practice, in VI, don't have to deal with policies at all. VI doesn't have any separate evaluation phase that runs to convergence. While goal of VI is same as goal of PI – to find optimal policy for a given MDP – VI happens to do this through value functions; thus name value iteration.
– Trên thực tế, trong VI, bạn không cần phải xử lý chính sách nào cả. VI không có bất kỳ giai đoạn đánh giá riêng biệt nào chạy đến hội tụ. Mặc dù mục tiêu của VI giống với mục tiêu của PI – tìm chính sách tối ưu cho 1 MDP nhất định – VI thực hiện việc này thông qua các hàm giá trị; do đó, tên gọi là vòng lặp giá trị.
Again, only have to keep track of a V-function & a Q-function (depending on implementation). Remember: get greedy policy over a Q-function, take arguments of maxima (argmax) over actions of that Q-function. Instead of improving policy by taking argmax to get a better policy & then evaluating this improved policy to obtain a value function again, directly calculate maximum (max, instead of argmax) value across actions to be used for next sweep over states.
– 1 lần nữa, chỉ cần theo dõi hàm V & hàm Q (tùy thuộc vào cách triển khai). Hãy nhớ: lấy chính sách tham lam trên 1 hàm Q, lấy các đối số cực đại (argmax) trên các hành động của hàm Q đó. Thay vì cải thiện chính sách bằng cách lấy argmax để có được chính sách tốt hơn & rồi đánh giá chính sách đã cải thiện này để có lại hàm giá trị, hãy tính trực tiếp giá trị cực đại (max, thay vì argmax) trên các hành động để sử dụng cho lần quét tiếp theo trên các trạng thái.
Only at end of VI algorithm, after Q-function converges to optimal values, do we extract optimal policy by taking argmax over actions of Q-function, as before. Will see it more clearly in code snipper on next page.
– Chỉ đến cuối thuật toán VI, sau khi hàm Q hội tụ đến các giá trị tối ưu, chúng ta mới trích xuất được chính sách tối ưu bằng cách lấy argmax trên các hành động của hàm Q, như trước đây. Chúng ta sẽ thấy rõ hơn trong code snipper ở trang sau.
1 important to highlight: whereas VI & PI are 2 different algorithms, in a more general view, they are 2 instances of *generalized policy iteration* (GPI). GPI is a general idea in RL in which policies are improved using their value function

estimates, & value function estimates are improved toward actual value function for current policy. Whether you wait for perfect estimates or not is just a detail.

– 1 điểm quan trọng cần nhấn mạnh: mặc dù VI & PI là 2 thuật toán khác nhau, nhưng nhìn chung, chúng là 2 ví dụ của phép lặp chính sách tổng quát (GPI). GPI là 1 ý tưởng chung trong RL, trong đó các chính sách được cải thiện bằng cách sử dụng ước lượng hàm giá trị của chúng, & ước lượng hàm giá trị được cải thiện theo hướng hàm giá trị thực tế cho chính sách hiện tại. Việc bạn có chờ đợi ước lượng hoàn hảo hay không chỉ là 1 chi tiết.

- Summary. Objective a RL agent: maximize expected return, which is total reward over multiple episodes. For this, agent must use policies, which can be thought of as universal plans. Policies prescribe actions for states. They can be deterministic, i.e., they return single actions, or stochastic, i.e., they return probability distributions. To obtain policies, agents usually keep track of several summary values. Main ones are state-value, action-value, & action-advantage functions.

  – Mục tiêu của 1 tác nhân RL: tối đa hóa lợi nhuận kỳ vọng, tức là tổng phần thưởng trên nhiều tập. Để làm được điều này, tác nhân phải sử dụng các chính sách, có thể được coi là các kế hoạch phổ quát. Các chính sách quy định hành động cho các trạng thái. Chúng có thể là xác định, tức là chúng trả về các hành động đơn lẻ, hoặc ngẫu nhiên, tức là chúng trả về các phân phối xác suất. Để có được các chính sách, tác nhân thường theo dõi 1 số giá trị tóm tắt. Các giá trị chính là các hàm trạng thái-giá trị, hành động-giá trị, & hành động-lợi thế.

  State-value functions summarize expected return from a state. They indicate how much reward agent will obtain from a state until end of an episode in expectation. Action-value functions summarize expected return from a state-action pair. This type of value function tells expected reward-to-go after an agent selects a specific action in a given state. Action-value functions allow agent to compare across actions & therefore solve control problem. Action-advantage functions show agent how much better than default it can do if it were to opt for a specific state-action pair. All of these value functions are mapped to specific policies, perhaps an optimal policy. They depend on following what policies prescribe until end of episode.

  – Các hàm giá trị trạng thái tóm tắt lợi nhuận kỳ vọng từ 1 trạng thái. Chúng chỉ ra mức phần thưởng mà tác nhân sẽ nhận được từ trạng thái cho đến khi kết thúc 1 tập kỳ vọng. Các hàm giá trị hành động tóm tắt lợi nhuận kỳ vọng từ 1 cặp trạng thái-hành động. Loại hàm giá trị này cho biết phần thưởng kỳ vọng sẽ đạt được sau khi tác nhân chọn 1 hành động cụ thể trong 1 trạng thái nhất định. Các hàm giá trị hành động cho phép tác nhân so sánh giữa các hành động & do đó giải quyết vấn đề điều khiển. Các hàm lợi thế hành động cho thấy tác nhân có thể làm tốt hơn mặc định bao nhiêu nếu lựa chọn 1 cặp trạng thái-hành động cụ thể. Tất cả các hàm giá trị này được ánh xạ tới các chính sách cụ thể, có thể là 1 chính sách tối ưu. Chúng phụ thuộc vào việc tuân theo các chính sách quy định cho đến khi kết thúc tập.

  Policy evaluation is a method for estimating a value function from a policy & an MDP. Policy improvement is a method for extracting a greedy policy from a value function & an MDP. Policy iteration consists of alternating between policy-evaluation & policy improvement to obtain an optimal policy from an MDP. Policy evaluation phase may run for several iterations before it accurately estimates value function for given policy. In policy iteration, wait until policy evaluation finds this accurate estimate. An alternative method, called value iteration, truncates policy-evaluation phase & exits it, entering policy-improvement phase early.

  – Đánh giá chính sách là 1 phương pháp để ước lượng hàm giá trị từ 1 chính sách & 1 MDP. Cải thiện chính sách là 1 phương pháp để trích xuất 1 chính sách tham lam từ 1 hàm giá trị & 1 MDP. Lặp lại chính sách bao gồm việc luân phiên giữa đánh giá chính sách & cải thiện chính sách để có được chính sách tối ưu từ 1 MDP. Giai đoạn đánh giá chính sách có thể chạy nhiều lần lặp lại trước khi ước lượng chính xác hàm giá trị cho 1 chính sách nhất định. Trong quá trình lặp lại chính sách, hãy đợi cho đến khi đánh giá chính sách tìm thấy ước lượng chính xác này. 1 phương pháp thay thế, được gọi là lặp lại giá trị, cắt ngắn giai đoạn đánh giá chính sách & thoát khỏi nó, chuyển sang giai đoạn cải thiện chính sách sớm hơn.

  More general view of these methods is generalized policy iteration, which describes interaction of 2 processes to optimize policies: one moves value function estimates closer to real value function of current policy, another improves current policy using its value function estimates, getting progressively better & better policies as this cycle continues. By now:

  * Know objective of a RL agent & different statistics it may hold at any given time
  * Understand methods for estimating value functions from policies & methods for improving policies from value functions
  * Can find optimal policies in sequential decision-making problems modeled by MDPs.

  – Quan điểm tổng quát hơn về các phương pháp này là lặp lại chính sách tổng quát, mô tả sự tương tác của 2 quy trình để tối ưu hóa chính sách: 1 quy trình di chuyển ước lượng hàm giá trị gần hơn với hàm giá trị thực của chính sách hiện tại, 1 quy trình khác cải thiện chính sách hiện tại bằng cách sử dụng ước lượng hàm giá trị của nó, dần dần cải thiện & các chính sách tốt hơn khi chu trình này tiếp tục. Đến đây:

  * Biết mục tiêu của 1 tác nhân RL & các số liệu thống kê khác nhau mà nó có thể nắm giữ tại bất kỳ thời điểm nào
  * Hiểu các phương pháp ước lượng hàm giá trị từ chính sách & các phương pháp cải thiện chính sách từ hàm giá trị
  * Có thể tìm ra các chính sách tối ưu trong các bài toán ra quyết định tuần tự được mô hình hóa bởi MDP.

- **4. Balancing gathering & use of information.** In this chap: Learn about challenges of learning from evaluative feedback & how to properly balance gathering & utilization of information. Develop exploration strategies that accumulate low levels of regret in problems with unknown transition function & reward signals. Write code with trial-&-error learning agents that learn to optimize their behavior through their own experiences in many-options, 1-choice environments known as multi-armed bandits.

  – **Cân bằng giữa thu thập & sử dụng thông tin.** Trong chương này: Tìm hiểu về những thách thức của việc học từ phản hồi đánh giá & cách cân bằng hợp lý giữa thu thập & sử dụng thông tin. Phát triển các chiến lược khám phá tích lũy mức

độ hối tiếc thấp trong các bài toán có hàm chuyển tiếp & tín hiệu phần thưởng chưa biết. Viết mã với các tác nhân học thử & lỗi, học cách tối ưu hóa hành vi của chúng thông qua trải nghiệm của chính chúng trong môi trường nhiều lựa chọn, 1 lựa chọn, được gọi là máy đánh bạc nhiều tay.

> "Uncertainty & expectation are joys of life. Security is an insipid thing." – William Congreve, English playwright & poet of Restoration period & political figure in the British Whig Party
>
> – "Sự bất định & kỳ vọng là niềm vui của cuộc sống. An ninh là 1 thứ nhạt nhẽo." – William Congreve, nhà viết kịch người Anh & nhà thơ thời kỳ Phục hưng & nhân vật chính trị trong Đảng Whig Anh

No matter how small & unimportant a decision may seem, every decision you make is a trade-off between information gathering & information exploitation. E.g., when go to your favorite restaurant, should you order your favorite dish, yet again, or should you request that dish you have been meaning to try? If a Silicon Valley startup offers you a job, should you make a career move, or should you stay put in your current role?

– Dù 1 quyết định có vẻ nhỏ nhặt & không quan trọng đến đâu, mỗi quyết định bạn đưa ra đều là sự đánh đổi giữa việc thu thập thông tin & khai thác thông tin. Ví dụ, khi đến nhà hàng yêu thích, bạn nên gọi món yêu thích 1 lần nữa, hay nên yêu cầu món ăn mà bạn đã định thử? Nếu 1 công ty khởi nghiệp ở Thung lũng Silicon mời bạn làm việc, bạn nên chuyển hướng sự nghiệp hay nên tiếp tục công việc hiện tại?

These kinds of questions illustrate exploration-exploitation dilemma & are at core of RL problem. It boils down to deciding when to acquire knowledge & when to capitalize on knowledge previously learned. It is a challenge to know whether good we already have is good enough. When do we settle? When do we go for more? What are your thoughts: is a bird in hand worth 2 in bush or not?

– Những câu hỏi kiểu này minh họa cho thế lưỡng nan giữa khám phá & khai thác & là cốt lõi của vấn đề thực tế. Nó quy về việc quyết định khi nào nên tiếp thu kiến thức & khi nào nên tận dụng kiến thức đã học. Thật khó để biết liệu những điều tốt đẹp chúng ta đang có đã đủ tốt hay chưa. Khi nào chúng ta nên bằng lòng? Khi nào chúng ta nên tìm kiếm thêm? Bạn nghĩ sao: 1 con chim trên tay có đáng giá bằng hai con chim trong bụi rậm hay không?

Main issue: rewarding moments in life are relative; have to compare events to see a clear picture of their value. E.g., bet you felt amazed when you were offered your 1st job. Perhaps even thought that was best thing that ever happened to you. But, then life continues, & you experience things that appear even more rewarding – maybe, when you get a promotion, a raise, or get married, who knows!

– Vấn đề chính: những khoảnh khắc đáng giá trong đời chỉ mang tính tương đối; phải so sánh các sự kiện để thấy rõ giá trị của chúng. Ví dụ, bạn đã từng cảm thấy kinh ngạc khi được nhận công việc đầu tiên. Thậm chí có thể bạn còn nghĩ đó là điều tuyệt vời nhất từng xảy ra với mình. Nhưng rồi cuộc sống vẫn tiếp diễn, & bạn trải nghiệm những điều dường như còn đáng giá hơn nữa – có thể là khi bạn được thăng chức, tăng lương, hoặc kết hôn, biết đâu đấy!

& that is core issue: even if you rank moments you have experienced so far by "how amazing" they felt, you can't know what's most amazing moment you could experience in your life – life is uncertain; you don't have life's transition function & reward signal, so must keep on exploring. In this chap, learn about how important it is for your agent to explore when interacting with uncertain environments, problems in which MDP isn't available for planning.

– & Đó là vấn đề cốt lõi: ngay cả khi bạn xếp hạng những khoảnh khắc bạn đã trải qua cho đến nay theo mức độ "tuyệt vời", bạn cũng không thể biết khoảnh khắc nào là tuyệt vời nhất mà bạn có thể trải nghiệm trong đời – cuộc sống là bất định; bạn không có chức năng chuyển tiếp của cuộc sống & tín hiệu phần thưởng, vì vậy phải tiếp tục khám phá. Trong chương này, hãy tìm hiểu tầm quan trọng của việc tác nhân của bạn khám phá khi tương tác với các môi trường bất định, những vấn đề mà MDP không thể lập kế hoạch.

In prev chap, learned about challenges of learning from sequential feedback & how to properly balance immediate & long-term goals. In this chap, examine challenges of learning from evaluative feedback, & do so in environments that aren't sequential, but 1-shot instead: *multi-armed bandits* (MABs).

– Trong chương trước, chúng ta đã tìm hiểu về những thách thức của việc học từ phản hồi tuần tự & cách cân bằng hợp lý các mục tiêu ngắn hạn & dài hạn. Trong chương này, chúng ta sẽ xem xét những thách thức của việc học từ phản hồi đánh giá, & thực hiện điều này trong môi trường không tuần tự, mà là môi trường 1 lần: *multi-armed bandits* (MAB).

MABs isolate & expose challenges of learning from evaluative feedback. Dive into many different techniques for balancing exploration & exploitation in these particular type of environments: single-state environments with multiple options, but a single choice. Agents will operate under uncertainty, i.e., they won't have access to MDP. However, they will interact with 1-shot environments without sequential component.

– MAB cô lập & phơi bày những thách thức của việc học từ phản hồi đánh giá. Khám phá nhiều kỹ thuật khác nhau để cân bằng giữa khám phá & khai thác trong những môi trường đặc thù này: môi trường trạng thái đơn với nhiều tùy chọn, nhưng chỉ có 1 lựa chọn. Các tác nhân sẽ hoạt động trong điều kiện không chắc chắn, tức là chúng sẽ không được truy cập vào MDP. Tuy nhiên, chúng sẽ tương tác với môi trường 1-shot không có thành phần tuần tự.

Remember, in DRL, agents learn to feedback that is simultaneously sequential (as opposed to 1 shot), evaluative (as opposed to supervised), & sampled (as opposed to exhaustive). In this chap, eliminate complexity that comes along with learning from sequential & sampled feedback, & study intricacies of evaluative feedback in isolation. Get to it.

– Nhớ rằng, trong DRL, các tác nhân học cách phản hồi vừa tuần tự (khác với 1 lần), vừa đánh giá (khác với giám sát), vừa lấy mẫu (khác với toàn diện). Trong chương này, hãy loại bỏ sự phức tạp đi kèm với việc học từ phản hồi tuần tự & lấy mẫu, đồng thời nghiên cứu những phức tạp của phản hồi đánh giá 1 cách riêng biệt. Hãy bắt tay vào làm.

- **4.1. Challenge of interpreting evaluative feedback.** In last chap, when solved FL environment, knew beforehand how environment would react to any of your actions. Knowing exact transition function & reward signal of an environment allows us to compute an optimal policy using planning algorithms, e.g. PI & VI, without having to interact with environment at all.

  – **Thử thách trong việc diễn giải phản hồi đánh giá.** Trong chương trước, khi giải quyết môi trường FL, chúng ta đã biết trước môi trường sẽ phản ứng như thế nào với bất kỳ hành động nào của mình. Việc biết chính xác hàm chuyển tiếp & tín hiệu phần thưởng của 1 môi trường cho phép chúng ta tính toán chính sách tối ưu bằng các thuật toán lập kế hoạch, ví dụ: PI & VI, mà không cần phải tương tác với môi trường.

  But, knowing an MDP in advance oversimplifies things, perhaps unrealistically. Cannot always assume we will know with precision how an environment will react to our actions – that is not how world works. Could opt for learning such things, as you will learn in later chaps, but bottom line: need to let our agents interact & experience environment by themselves, learning this way to behave optimally, solely from their own experience. This is what is called trial-&-error learning.

  – Tuy nhiên, việc biết trước 1 MDP sẽ đơn giản hóa mọi thứ, thậm chí là phi thực tế. Chúng ta không thể luôn cho rằng mình sẽ biết chính xác môi trường sẽ phản ứng như thế nào với hành động của mình – đó không phải là cách thế giới vận hành. Bạn có thể chọn học những điều như vậy, như bạn sẽ tìm hiểu trong các chương sau, nhưng điểm mấu chốt: cần để các tác nhân tương tác & tự trải nghiệm môi trường, học cách hành xử tối ưu này, hoàn toàn dựa trên kinh nghiệm của chính chúng. Đây chính là cái gọi là học thử &-sai.

  In RL, when agent learns to behave from interaction with environment, environment asks agent same question over & over: what do you want to do now? This question presents a fundamental challenge to a decision-making agent. What action should it do now? Should agent exploit its current knowledge & select action with highest current estimate? Or should it explore actions that it hasn't tried enough? But many additional questions follow: when do you know your estimates are good enough? How do you know you have tried an apparently bad action enough? & so on. Will learn more effective ways for dealing with exploration-exploitation trade-off. Key intuition: exploration builds knowledge that allows for effective exploitation, & maximum exploitation is ultimate goa of any decision maker.

  – Trong RL, khi tác nhân học cách ứng xử từ tương tác với môi trường, môi trường sẽ hỏi tác nhân cùng 1 câu hỏi lặp đi lặp lại: bạn muốn làm gì bây giờ? Câu hỏi này đặt ra 1 thách thức cơ bản cho tác nhân ra quyết định. Tác nhân nên làm gì bây giờ? Tác nhân có nên khai thác kiến thức hiện tại của mình & chọn hành động có ước tính hiện tại cao nhất không? Hay nó nên khám phá các hành động mà nó chưa thử đủ? Nhưng nhiều câu hỏi bổ sung theo sau: khi nào bạn biết ước tính của mình đủ tốt? Làm thế nào để bạn biết mình đã thử 1 hành động có vẻ tệ đủ rồi? & vân vân. Sẽ học được những cách hiệu quả hơn để giải quyết sự đánh đổi giữa thăm dò & khai thác. Trực giác chính: thăm dò xây dựng kiến thức cho phép khai thác hiệu quả, & khai thác tối đa là mục tiêu cuối cùng của bất kỳ người ra quyết định nào.

  * **Bandits: Single-state decision problems.** *Multi-armed bandits* (MAB) are a special case of an RL problem in which size of state space & horizon equal 1. A MAB has multiple actions, a single state, & a greedy horizon; can also think of it as a "many-options, single-choice" environment. Name comes from slot machines (bandits) with multiple arms to choose from (more realistically, multiple slot machines to choose from).

    – **Bandits: Bài toán quyết định trạng thái đơn.** *Multi-armed bandits* (MAB) là 1 trường hợp đặc biệt của bài toán RL trong đó kích thước không gian trạng thái & đường chân trời bằng 1. MAB có nhiều hành động, 1 trạng thái duy nhất, & đường chân trời tham lam; cũng có thể coi nó là 1 môi trường "nhiều lựa chọn, 1 lựa chọn duy nhất". Tên gọi này bắt nguồn từ máy đánh bạc (bandits) với nhiều tay để lựa chọn (thực tế hơn là nhiều máy đánh bạc để lựa chọn).

    Multi-armed bandit problem. A 2-armed bandit is a decision-making problem with 2 choices. Need to try them both sufficient to correctly assess each option. How do you best handle exploration-exploitation trade-off?

    – Bài toán máy đánh bạc nhiều tay. Máy đánh bạc hai tay là bài toán ra quyết định với 2 lựa chọn. Cần thử cả hai đủ nhiều để đánh giá chính xác từng lựa chọn. Làm thế nào để xử lý tốt nhất sự đánh đổi giữa thăm dò & khai thác?

    There are many commercial applications for methods coming out of MAB research. Advertising companies need to find right way to balance showing you an ad they predict you are likely to click on & showing you a new ad with potential of it being an even better fit for you. Websites that raise money, e.g. charities or political campaigns, need to balance between showing layout that has led to most contributions & new designs that haven't been sufficiently utilized but still have potential for even better outcomes. Likewise, e-commerce websites need to balance recommending you best-seller products as well as promising new products. In medical trials, there is a need to learn effects of medicines in patients as quickly as possible. Many other problems benefit from study of exploration-exploitation trade-off: oil drilling, game playing, & search engines, to name a few. Our reason for studying MABs isn't so much a direct application to real world, but instead how to integrate a suitable method for balancing exploration & exploitation in RL agents.

    – Có rất nhiều ứng dụng thương mại cho các phương pháp xuất phát từ nghiên cứu MAB. Các công ty quảng cáo cần tìm ra cách phù hợp để cân bằng giữa việc hiển thị cho bạn 1 quảng cáo mà họ dự đoán bạn có khả năng nhấp vào & hiển thị cho bạn 1 quảng cáo mới có tiềm năng phù hợp hơn với bạn. Các trang web gây quỹ, ví dụ như các tổ chức từ thiện hoặc chiến dịch chính trị, cần cân bằng giữa việc hiển thị bố cục đã dẫn đến hầu hết các khoản đóng góp & các thiết kế mới chưa được sử dụng đầy đủ nhưng vẫn có tiềm năng mang lại kết quả tốt hơn nữa. Tương tự, các trang web thương mại điện tử cần cân bằng giữa việc giới thiệu cho bạn các sản phẩm bán chạy nhất cũng như các sản phẩm mới đầy hứa hẹn. Trong các thử nghiệm y tế, cần phải tìm hiểu tác dụng của thuốc trên bệnh nhân càng nhanh càng tốt. Nhiều vấn

đề khác được hưởng lợi từ việc nghiên cứu sự đánh đổi giữa thăm dò & khai thác: khoan dầu, chơi trò chơi, & công cụ tìm kiếm, v.v. Lý do chúng tôi nghiên cứu MAB không hẳn là ứng dụng trực tiếp vào thế giới thực, mà là cách tích hợp 1 phương pháp phù hợp để cân bằng giữa thăm dò & khai thác trong các tác nhân thực tế.

Multi-armed bandit. 1. MABs are MDPs with a single non-terminal state, & a single time step per episode. 2. Q-function of action a is expected reward given a was sampled. $q(a) = \mathbb{E}[R_t|A_t = a]$. 3. Best we can do in a MAB is represented by optimal V-function, or selecting action that maximizes Q-function. $v_* = q(a_*) = \max_{a \in A} q(a)$. 4. Optimal action is action that maximizes optimal Q-function, & optimal V-function (only 1 state). $q(a_*) = v_*$.

– **Máy đánh bạc nhiều tay.** 1. MAB là MDP có 1 trạng thái không kết thúc duy nhất, & 1 bước thời gian duy nhất cho mỗi tập. 2. Hàm Q của hành động a là phần thưởng mong đợi khi a được lấy mẫu. $q(a) = \mathbb{E}[R_t|A_t = a]$. 3. Điều tốt nhất chúng ta có thể làm trong MAB được biểu thị bằng hàm V tối ưu hoặc chọn hành động tối đa hóa hàm Q. $v_* = q(a_*) = \max_{a \in A} q(a)$. 4. Hành động tối ưu là hành động tối đa hóa hàm Q tối ưu, & hàm V tối ưu (chỉ có 1 trạng thái). $q(a_*) = v_*$.

∗ **Regret: Cost of exploration.** Goal of MABs is very similar to that of RL. In Rl, agent needs to maximize expected cumulative discounted reward (maximize expected return). I.e., get as much reward (maximize) through course of an episode (cumulative) as soon as possible (if discounted – later rewards are discounted more) despite environment's stochasticity (expected). This make sense when environment has multiple states & agent interacts with it for multiple time steps per episode. But in MABs, while there are multiple episodes, only have a single chance of selecting an action in each episode.

– **Tiếc nuối: Chi phí khám phá.** Mục tiêu của MAB rất giống với RL. Trong R1, tác nhân cần tối đa hóa phần thưởng tích lũy kỳ vọng (tối đa hóa lợi nhuận kỳ vọng). Tức là, nhận được càng nhiều phần thưởng (tối đa hóa) càng tốt trong suốt 1 tập (tích lũy) càng sớm càng tốt (nếu được chiết khấu – phần thưởng sau được chiết khấu nhiều hơn) bất chấp tính ngẫu nhiên của môi trường (dự kiến). Điều này hợp lý khi môi trường có nhiều trạng thái & tác nhân tương tác với nó trong nhiều bước thời gian cho mỗi tập. Nhưng trong MAB, mặc dù có nhiều tập, nhưng tác nhân chỉ có 1 cơ hội duy nhất để chọn 1 hành động trong mỗi tập.

Therefore, can exclude words that don't apply to MAB case from RL goal: remove "cumulative" because there's only a single time step per episode, & "discounted" because there are no next states to account for. I.e., in MABs, goal is for agent to maximize expected reward. Notice: word "expected" stays there because there is stochasticity in environment. In fact, that's what MAB agents need to learn: underlying probability distribution of reward signal.

– Do đó, có thể loại trừ các từ không áp dụng cho trường hợp MAB khỏi mục tiêu RL: loại bỏ "tích lũy" vì chỉ có 1 bước thời gian duy nhất cho mỗi tập, & "giảm giá" vì không có trạng thái tiếp theo nào cần tính đến. Ví dụ, trong MAB, mục tiêu là để tác nhân tối đa hóa phần thưởng kỳ vọng. Lưu ý: từ "kỳ vọng" vẫn ở đó vì có tính ngẫu nhiên trong môi trường. Trên thực tế, đó là điều mà các tác nhân MAB cần học: phân phối xác suất cơ bản của tín hiệu phần thưởng.

However, if leave goal to "maximize expected reward", it wouldn't be straightforward to compare agents. E.g., say an agent learns to maximize expected reward by selecting random actions in all but final episode, while a much more sample-efficient agent uses a clever strategy to determine optimal action quickly. If only compare final-episode performance of these agents, which isn't uncommon to see in RL, these 2 agents would have equally good performance, which is obviously not what we want.

– Tuy nhiên, nếu đặt mục tiêu "tối đa hóa phần thưởng kỳ vọng", việc so sánh các tác nhân sẽ không hề đơn giản. Ví dụ, giả sử 1 tác nhân học cách tối đa hóa phần thưởng kỳ vọng bằng cách chọn các hành động ngẫu nhiên trong tất cả các tập ngoại trừ tập cuối, trong khi 1 tác nhân hiệu quả hơn nhiều về mặt mẫu lại sử dụng 1 chiến lược thông minh để nhanh chóng xác định hành động tối ưu. Nếu chỉ so sánh hiệu suất tập cuối của các tác nhân này, điều không hiếm thấy trong đời thực, thì 2 tác nhân này sẽ có hiệu suất tốt như nhau, rõ ràng đây không phải là điều chúng ta mong muốn.

A robust way to capture a more complete goal is for agent to maximize per-episode expected reward while still minimizing total expected reward loss of rewards across all episodes. To calculate this value, called *total regret*, sum per-episode difference of true expected reward of optimal action & true expected reward of selected action. Obviously, the lower total regret, the better. Notice use word true here; to calculate regret, must have access to MDP. That doesn't mean your agent needs MDP, only that you need it to compare agents' exploration strategy efficiency.

– 1 cách mạnh mẽ để nắm bắt mục tiêu hoàn chỉnh hơn là để tác nhân tối đa hóa phần thưởng kỳ vọng cho mỗi tập phim trong khi vẫn giảm thiểu tổng phần thưởng kỳ vọng bị mất trên tất cả các tập phim. Để tính giá trị này, được gọi là *total regret*, hãy cộng chênh lệch giữa phần thưởng kỳ vọng thực tế của hành động tối ưu & phần thưởng kỳ vọng thực tế của hành động đã chọn cho mỗi tập phim. Rõ ràng, tổng số hối tiếc càng thấp thì càng tốt. Lưu ý sử dụng từ "true" ở đây; để tính toán sự hối tiếc, phải có quyền truy cập vào MDP. Điều đó không có nghĩa là tác nhân của bạn cần MDP, mà chỉ là bạn cần nó để so sánh hiệu quả chiến lược khám phá của các tác nhân.

Total regret equation. 1. To calculate total regret $\mathcal{T}$, need to add, for all episodes, difference between optimal value of MAB & true value of action selected.

– **Phương trình tổng hối tiếc.** 1. Để tính tổng hối tiếc $\mathcal{T}$, cần phải cộng thêm, đối với tất cả các tập, sự khác biệt giữa giá trị tối ưu của MAB & giá trị thực của hành động đã chọn.

$$\mathcal{T} = \sum_{e=1}^{E} \mathbb{E}[v_* - q_*(A_e)].$$

∗ **Approaches to solving MAB environments.** There are 3 major kinds of approaches to tackling MABs. Most popular & straightforward approach involves exploring by injecting randomness in our action-selection process; i.e., our agent will exploit most of time, & sometimes it will explore using randomness. This family of approaches is called *random exploration strategies*. A basic example of this family would be a strategy that selects greedy action most of time, & with an epsilon

threshold, it chooses uniformly at random. Now, multiple questions arise from this strategy; e.g., should we keep this epsilon value constant throughout episodes? Should we maximize exploration early on? Should we periodically increase epsilon value to ensure agent always explores?

– **Các phương pháp tiếp cận để giải quyết môi trường MAB.** Có 3 loại phương pháp chính để xử lý MAB. Phương pháp phổ biến nhất & đơn giản nhất liên quan đến việc khám phá bằng cách đưa tính ngẫu nhiên vào quy trình lựa chọn hành động; tức là, tác nhân của chúng ta sẽ khai thác hầu hết thời gian, & đôi khi nó sẽ khám phá bằng cách sử dụng tính ngẫu nhiên. Nhóm các phương pháp này được gọi là *các chiến lược khám phá ngẫu nhiên*. 1 ví dụ cơ bản của nhóm này là 1 chiến lược chọn hành động tham lam hầu hết thời gian, & với ngưỡng epsilon, nó chọn ngẫu nhiên đồng đều. Giờ đây, nhiều câu hỏi nảy sinh từ chiến lược này; ví dụ, chúng ta có nên giữ giá trị epsilon này không đổi trong suốt các tập phim không? Chúng ta có nên tối đa hóa việc khám phá ngay từ đầu không? Chúng ta có nên tăng giá trị epsilon định kỳ để đảm bảo tác nhân luôn khám phá không?

Another approach to dealing with exploration-exploitation dilemma is to be optimistic. Family of *optimistic exploration strategies* is a more systematic approach that quantifies uncertainty in decision-making problem & increases preference for states with highest uncertainty. Bottom line: being optimistic will naturally drive you toward uncertain states because you will assume: states you haven't experienced yet are best they can be. This assumption will help you explore, & as you explore & come face to face with reality, your estimates will get lower & lower as they approach their true values.

– 1 cách tiếp cận khác để giải quyết tình thế tiến thoái lưỡng nan giữa thăm dò & khai thác là lạc quan. Nhóm các chiến lược thăm dò lạc quan là 1 cách tiếp cận có hệ thống hơn, định lượng sự không chắc chắn trong vấn đề ra quyết định & tăng cường sự ưu tiên cho các trạng thái có độ không chắc chắn cao nhất. Tóm lại: lạc quan sẽ tự nhiên đưa bạn đến các trạng thái không chắc chắn vì bạn sẽ cho rằng: những trạng thái bạn chưa trải nghiệm là tốt nhất có thể. Giả định này sẽ giúp bạn khám phá, & khi bạn khám phá & đối mặt với thực tế, ước tính của bạn sẽ giảm & thấp hơn khi chúng tiến gần đến giá trị thực của chúng.

3rd approach to dealing with exploration-exploitation dilemma is family of *information state-space exploration strategies*. These strategies will model information state of agent as part of environment. Encoding uncertainty as part of state space means: an environment state will be seen differently when unexplored or explored. Encoding uncertainty as part of environment is a sound approach but can also considerably increase size of state space &, therefore, its complexity.

– Cách tiếp cận thứ ba để giải quyết tình huống khó xử trong việc khai thác-khám phá là 1 nhóm các chiến lược khai thác không gian trạng thái thông tin. Các chiến lược này sẽ mô hình hóa trạng thái thông tin của tác nhân như 1 phần của môi trường. Mã hóa sự không chắc chắn như 1 phần của không gian trạng thái có nghĩa là: trạng thái của môi trường sẽ được nhìn nhận khác nhau khi chưa được khám phá hoặc được khám phá. Mã hóa sự không chắc chắn như 1 phần của môi trường là 1 cách tiếp cận hợp lý nhưng cũng có thể làm tăng đáng kể kích thước của không gian trạng thái &, do đó, độ phức tạp của nó.

In this chap, explore a few instances of 1st 2 approaches. Do this in a handful of different MAB environments with different properties, pros & cons, & this will allow us to compare strategies in depth.

– Trong chương này, chúng ta sẽ khám phá 1 vài ví dụ về cách tiếp cận 1/2. Thực hiện điều này trong 1 số môi trường MAB khác nhau với các đặc tính, ưu & nhược điểm khác nhau, & điều này sẽ cho phép chúng ta so sánh các chiến lược 1 cách sâu sắc.

Important to notice: estimation of Q-function in MAB environments is pretty straightforward & something all strategies will have in common. Because MABs are 1-step environments, to estimate Q-function, need to calculate per-action average reward. I.e., estimate of an action a is equal to total reward obtained when selecting action a, divided by number of times action a has been selected.

– Điều quan trọng cần lưu ý: ước tính hàm Q trong môi trường MAB khá đơn giản & là điểm chung của tất cả các chiến lược. Vì MAB là môi trường 1 bước, để ước tính hàm Q, cần tính toán phần thưởng trung bình cho mỗi hành động. Ví dụ, ước tính của 1 hành động a bằng tổng phần thưởng thu được khi chọn hành động a, chia cho số lần hành động a được chọn.

Essential to highlight: there are no differences in how strategies we evaluate in this chap estimate Q-function; only difference is in how each strategy uses Q-function estimates to select actions.

– Điều cần nhấn mạnh là: không có sự khác biệt nào trong cách chúng ta đánh giá các chiến lược trong chương này để ước tính hàm Q; sự khác biệt duy nhất là cách mỗi chiến lược sử dụng ước tính hàm Q để lựa chọn hành động.

Slippery bandit walk (SBW) environment. 1st MAB environment considered is bandit slippery walk (BSW). Remember, BSW is a grid world with a signle row, thus, a walk. But a special feature of this walk: agent starts at middle, & any action sends agent to a terminal state immediately. Because it is a 1-time-step, it is a bandit environment.

– **Môi trường Slippery Bandit Walk (SBW).** Môi trường MAB đầu tiên được xem xét là Bandit Slippery Walk (BSW). Hãy nhớ rằng, BSW là 1 thế giới lưới với 1 hàng đơn, do đó, là 1 bước đi. Nhưng 1 tính năng đặc biệt của bước đi này: tác nhân bắt đầu ở giữa, & bất kỳ hành động nào cũng sẽ đưa tác nhân đến trạng thái kết thúc ngay lập tức. Vì đây là bước 1 lần, nên nó là 1 môi trường Bandit.

p. 103+++

    - **Greedy: Always exploit.**
- **4.2. Strategic exploration.**

- **5. Evaluating agents' behaviors.**
  - **5.1. Learning to estimate value of policies.**
  - **5.2. Learning to estimate from multiple steps.**

- **6. Improving agents' behaviors.**
  - **6.1. Anatomy of RL agents.**
  - **6.2. Learning to improve policies of behavior.**
  - **6.3. Decoupling behavior from learning.**
- **7. Achieving goals more effectively & efficiently.**
  - **7.1. Learning to improve policies using robust targets.**
  - **7.2. Agents that interact, learn, & plan.**
- **8. Introduction to value-based deep reinforcement learning.**
  - **8.1. Kind of feedback DRL agents use.**
  - **8.2. Introduction to function approximation for RL.**
  - **8.3. NFQ: 1st attempt at value-based DRL.**
- **9. More stable value-based methods.**
  - **9.1. DQN: Making RL more like supervised learning.**
  - **9.2. Double DQN: Mitigating overestimation of action-value functions.**
- **10. Sample-efficient value-based methods.**
  - **10.1. Dueling DDQN: A RL-aware neural network architecture.**
  - **10.2. PER: Prioritizing replay of meaningful experiences.**
- **11. Policy-gradient & actor-critic methods.**
  - **11.1. REINFORCE: Outcome-based policy learning.**
  - **11.2. VPG: Learning a value function.**
  - **11.3. A3C: Parallel policy updates.**
  - **11.4. GAE: Robust advantage estimation.**
  - **11.5. A2C: Synchronous policy updates.**
- **12. Advanced actor-critic methods.**
  - **12.1. DDPG: Approximating a deterministic policy.**
  - **12.2. TD3: State-of-art improvements over DDPG.**
  - **12.3. SAC: Maximizing expected return & entropy.**
  - **12.4. PPO: Restricting optimization steps.**
- **13. Toward artificial general intelligence.**
  - **13.1. What was covered & what notably wasn't?**
  - **13.2. More advanced concepts toward AGI.**
  - **13.3. What happens next?**

1.2 Richard S. Sutton, Andrew G. Barto. Reinforcement Learning: An Introduction. 2e. 2020

- **Preface to 2e.** 20 years since publication of 1e of this book have seen tremendous progress in AI, propelled in large part by advances in ML, including advances in RL. Although impressive computational power that became available is responsible for some of these advances, new developments in theory & algorithms have been driving forces as well. In face of this progress, a 2e of 1998 book was long overdue, & finally began project in 2012. Goal for 2e was same as goal for 1e: provide a clear & simple account of key ideas & algorithms of RL that is accessible to readers in all related disciplines. Edition remains an introduction, & retain a focus on core, online learning algorithms. This edition includes some new topics that rose to importance over intervening years, & expanded coverage of topics that we now understand better. But made no attempt to provide comprehensive coverage of field, which has exploded in many different directions. Apologize for having to leave out all but a handful of these contributions.

  – 20 năm kể từ khi xuất bản ấn bản đầu tiên của cuốn sách này, chúng ta đã chứng kiến những tiến bộ vượt bậc trong lĩnh vực AI, phần lớn nhờ vào những tiến bộ trong Học máy (ML), bao gồm cả những tiến bộ trong Học máy (RL). Mặc dù sức mạnh tính toán ấn tượng đã góp phần tạo nên 1 số tiến bộ này, nhưng những phát triển mới về lý thuyết & thuật toán cũng là động lực thúc đẩy. Trước những tiến bộ này, ấn bản thứ 2 của cuốn sách năm 1998 đã bị trì hoãn từ lâu, & cuối cùng đã

được khởi động dự án vào năm 2012. Mục tiêu của ấn bản thứ 2 cũng giống như mục tiêu của ấn bản thứ 1: cung cấp 1 bản tóm tắt rõ ràng & đơn giản về những ý tưởng chính & thuật toán của Học máy (RL), dễ hiểu cho độc giả trong mọi lĩnh vực liên quan. Ấn bản này vẫn là phần giới thiệu, & tập trung vào các thuật toán học tập trực tuyến cốt lõi. Ấn bản này bao gồm 1 số chủ đề mới đã trở nên quan trọng trong những năm tiếp theo, & mở rộng phạm vi bao quát các chủ đề mà giờ đây chúng ta đã hiểu rõ hơn. Tuy nhiên, ấn bản này không cố gắng cung cấp phạm vi bao quát toàn diện về lĩnh vực này, vốn đã bùng nổ theo nhiều hướng khác nhau. Xin lỗi vì đã phải bỏ qua hầu hết những đóng góp này.

As in 1e, chose not to produce a rigorous formal treatment of RL, or to formulate it in most general terms. However, our deeper understanding of some topics since 1e required a bit more mathematics to explain; have set off more mathematical parts in shaded boxes that non-mathematically-inclined may choose to skip. Also use a slightly different notation used in 1e. In teaching, have found: new notation helps to address some common points of confusion. It emphasizes difference between random variables, denoted with capital letters, & their instantiations, denoted in lower case. E.g., state, action, & reward at time step t are denoted S_t, A_t, R_t, while their possible values might be denoted s, a, r. Along with this, natural to use lower case for value functions (e.g., v_π) & restrict capitals to their tabular estimates (e.g., $Q_t(s, a)$). Approximate value functions are deterministic functions of random parameters & are thus also in lower case (e.g., $\hat{v}(s, \mathbf{w}_t) \approx v_\pi(s)$). Vectors, e.g. weight vector $\mathbf{w}_t$ (formerly $\boldsymbol{\theta}_t$) & feature vector $\mathbf{x}_t$ (formerly $\boldsymbol{\phi}_t$), are bold & written in lowercase even if they are random variables. Uppercase bold is reserved for matrices. In 1e used special notations, $\mathcal{P}^a_{ss'}, \mathcal{R}^a_{ss'}$, for transition probabilities & expected rewards. 1 weakness of that notation: it still did not fully characterize dynamics of rewards, giving only their expectations, which is sufficient for dynamic programming but not for RL. Another weakness is excess of subscripts & superscripts. In this edition, use explicit notation of $p(s', r|s, a)$ for joint probability for next state & reward given current state & action. All changes in notation are summarized in a table on p. 6.

– Giống như trong 1e, chúng tôi đã chọn không đưa ra 1 cách xử lý chính thức nghiêm ngặt về RL, hoặc xây dựng nó theo những thuật ngữ chung nhất. Tuy nhiên, sự hiểu biết sâu sắc hơn của chúng tôi về 1 số chủ đề kể từ 1e đòi hỏi nhiều kiến thức toán học hơn để giải thích; đã đặt nhiều phần toán học hơn trong các ô tô đậm mà những người không có thiên hướng toán học có thể chọn bỏ qua. Ngoài ra, chúng tôi cũng sử dụng 1 ký hiệu hơi khác được sử dụng trong 1e. Trong giảng dạy, chúng tôi đã tìm thấy: ký hiệu mới giúp giải quyết 1 số điểm dễ nhầm lẫn phổ biến. Nó nhấn mạnh sự khác biệt giữa các biến ngẫu nhiên, được ký hiệu bằng chữ in hoa, & các thể hiện của chúng, được ký hiệu bằng chữ thường. Ví dụ: trạng thái, hành động, & phần thưởng tại bước thời gian t được ký hiệu là S_t, A_t, R_t, trong khi các giá trị khả dĩ của chúng có thể được ký hiệu là s, a, r. Cùng với điều này, việc sử dụng chữ thường cho các hàm giá trị (ví dụ: v_π) & giới hạn chữ hoa trong các ước lượng dạng bảng của chúng là điều tự nhiên (ví dụ: $Q_t(s, a)$). Hàm giá trị gần đúng là hàm xác định của các tham số ngẫu nhiên & do đó cũng được viết thường (ví dụ: $\hat{v}(s, \mathbf{w}_t) \approx v_\pi(s)$). Các vectơ, ví dụ: vectơ trọng số $\mathbf{w}_t$ (trước đây là $\boldsymbol{\theta}_t$) & vectơ đặc trưng $\mathbf{x}_t$ (trước đây là $\boldsymbol{\phi}_t$), được viết đậm & bằng chữ thường ngay cả khi chúng là các biến ngẫu nhiên. Chữ in hoa đậm được dành riêng cho các ma trận. Trong 1e đã sử dụng các ký hiệu đặc biệt, $\mathcal{P}^a_{ss'}, \mathcal{R}^a_{ss'}$, cho xác suất chuyển đổi & phần thưởng mong đợi. 1 điểm yếu của ký hiệu đó: nó vẫn không mô tả đầy đủ động lực của phần thưởng, chỉ đưa ra kỳ vọng của chúng, đủ cho lập trình động nhưng không đủ cho RL. 1 điểm yếu khác là có quá nhiều chỉ số dưới & chỉ số trên. Trong phiên bản này, hãy sử dụng ký hiệu rõ ràng $p(s', r|s, a)$ cho xác suất kết hợp cho trạng thái tiếp theo & phần thưởng dựa trên trạng thái hiện tại & hành động. Tất cả các thay đổi về ký hiệu được tóm tắt trong bảng ở trang 6.

2e is significantly expanded, & its top-level organization has been changed. After introductory 1st chap, 2e is divided into 3 new parts. 1st part (Chaps. 2–8) treats as much of RL as possible without going beyond tabular case for which exact solutions can be found. Cover both learning & planning methods for tabular case, as well as their unification in n-step methods & in Dyna. Many algorithms presented in this part are new to 2e, including UCB, Expected Sarsa, Double learning, tree-backup, $Q(\sigma)$, RTDP, & MCTS. Doing tabular case 1st, & thoroughly, enables core ideas to be developed in simplest possible setting. 2nd part of book (Chaps. 9–13) is then devoted to extending ideas to function approximation. It has new sects on ANNs, Fourier basis, LSTD, kernel-based methods, Gradient-TD & Emphatic-TD methods, average-reward methods, true online TD(λ), & policy-gardient methods. 2e significantly expands treatment of off-policy learning, 1st for tabular case in Chaps. 5–7, then with function approximation in Chaps. 11–12. Another change: 2e separates forward-view idea of n-step boostrapping (now treated more fully in Chap. 7) from backward-view idea of eligibility traces (now treated independently in Chap. 12). 3rd part of book has large new chaps on RL's relationships to psychology (Chap. 14) & neuroscience (Chap. 15), as well as an updated case-studies chap including Atari game playing, Watson's wagering strategy, & Go playing programs AlphaGo & AlphaGo Zero (Chap. 16). Still, out of necessity we have included only a small subset of all that has been done in field. Our choices reflect our long-standing interests in inexpensive model-free methods that should scale well to large applications. Final chap now includes a discussion of future societal impacts of RL. For better or worst, 2e is about twice as large as 1e.

– 2e is significantly expanded, & its top-level organization has been changed. After introductory 1st chap, 2e is divided into 3 new parts. 1st part (Chaps. 2–8) treats as much of RL as possible without going beyond tabular case for which exact solutions can be found. Cover both learning & planning methods for tabular case, as well as their unification in n-step methods & in Dyna. Many algorithms presented in this part are new to 2e, including UCB, Expected Sarsa, Double learning, tree-backup, $Q(\sigma)$, RTDP, & MCTS. Doing tabular case 1st, & thoroughly, enables core ideas to be developed in simplest possible setting. 2nd part of book (Chaps. 9–13) is then devoted to extending ideas to function approximation. It has new sects on ANNs, Fourier basis, LSTD, kernel-based methods, Gradient-TD & Emphatic-TD methods, average-reward methods, true online TD(λ), & policy-gardient methods. 2e significantly expands treatment of off-policy learning, 1st for tabular case in Chaps. 5–7, then with function approximation in Chaps. 11–12. Another change: 2e separates forward-view idea of n-step boostrapping (now treated more fully in Chap. 7) from backward-view idea of eligibility traces (now treated independently in Chap. 12). 3rd part of book has large new chaps on RL's relationships to psychology (Chap. 14) & neuroscience (Chap. 15), as well as

an updated case-studies chap including Atari game playing, Watson's wagering strategy, & Go playing programs AlphaGo & AlphaGo Zero (Chap. 16). Still, out of necessity we have included only a small subset of all that has been done in field. Our choices reflect our long-standing interests in inexpensive model-free methods that should scale well to large applications. Final chap now includes a discussion of future societal impacts of RL. For better or worst, 2e is about twice as large as 1e.

This book is designed to be used as primary text for a 1- or 2-semester course on RL. For a 1-semester course, 1st 10 chaps should be covered in order & form a good core, to which can be added material from other chaps, from other books e.g. Bertsekas & Tsitsiklis (1996), Wiering & van Otterlo (2012), & Szepesvári (2010), or from literature, according to taste. Depending of students' background, some additional material on online supervised learning may be helpful. Ideas of options & option models are a natural addition (Sutton, Precup & Singh, 1999). A 2-semester course can cover all chaps as well as supplementary material. Book can also be used as part of broader courses on Ml, AI, or neural networks. In this case, it may be desirable to cover only a subset of material. Recommend covering Chap. 1 for a brief overview, Chap. 2 through Sect. 2.4, Chap. 3, & then selecting sections from remaining chaps according to time & interests. Chap. 6 is most important for subset & for rest of book. A course focusing on ML or neural networks should cover Chaps. 9–10, & a course focusing on AI or planning should cover Chap. 8. Throughout book, sects & chaps that are more difficult & not essential to rest of book are marked with a *. These can be omitted on 1st reading without creating problems later on. Some exercises are also marked with a * to indicate: they are more advanced & not essential to understanding basic material of chap.

– Cuốn sách này được thiết kế để sử dụng làm tài liệu chính cho khóa học 1 hoặc 2 học kỳ về RL. Đối với khóa học 1 học kỳ, 10 chương đầu tiên nên được học theo thứ tự & tạo thành 1 cốt lõi tốt, có thể thêm tài liệu từ các chương khác, từ các cuốn sách khác, ví dụ như Bertsekas & Tsitsiklis (1996), Wiering & van Otterlo (2012) & Szepesvári (2010) hoặc từ tài liệu tham khảo, tùy theo sở thích. Tùy thuộc vào nền tảng của sinh viên, 1 số tài liệu bổ sung về học tập có giám sát trực tuyến có thể hữu ích. Ý tưởng về các tùy chọn & mô hình tùy chọn là 1 sự bổ sung tự nhiên (Sutton, Precup & Singh, 1999). 1 khóa học 2 học kỳ có thể bao gồm tất cả các chương cũng như tài liệu bổ sung. Sách cũng có thể được sử dụng như 1 phần của các khóa học rộng hơn về ML, AI hoặc mạng nơ-ron. Trong trường hợp này, có thể chỉ nên học 1 phần nhỏ tài liệu. Đề xuất học Chương 1 để có cái nhìn tổng quan ngắn gọn, Chương Từ Chương 2 đến Chương 2.4, Chương 3, & sau đó chọn các phần từ các chương còn lại theo thời gian & sở thích. Chương 6 là quan trọng nhất đối với tập con & cho phần còn lại của sách. 1 khóa học tập trung vào Học máy hoặc mạng nơ-ron nên bao gồm các Chương 9–10, & 1 khóa học tập trung vào Trí tuệ nhân tạo hoặc Lập kế hoạch nên bao gồm Chương 8. Xuyên suốt sách, các chương & khó hơn & không cần thiết cho phần còn lại của sách được đánh dấu bằng dấu *. Những chương này có thể được bỏ qua khi đọc lần đầu mà không gây ra vấn đề gì sau này. 1 số bài tập cũng được đánh dấu bằng dấu * để chỉ ra: chúng nâng cao hơn & không cần thiết cho việc hiểu nội dung cơ bản của chương.

- **Preface to 1e.** 1st came to focus on what is now known as RL in late 1979. Both at University of Massachusetts, work on 1 of earliest projects to revive idea that networks of neuronlike adaptive elements might prove to be a promising approach to artificial adaptive intelligence. Project explored "heterostatic theory of adaptive systems" developed by A. Harry Klopf. Harry's work was a rich source of ideas, & we were permitted to explore them critically & compare them with long history of prior work in adaptive systems. Our task become 1 of teasing ideas apart & understanding their relationships & relative importance. This continues today, but in 1979 we came to realize: perhaps simplest of ideas, which had long been taken for granted, had received surprisingly little attention from a computational perspective. This was simply idea of a learning system that *wants* something, that adapts its behavior in order to maximize a special signal from its environment. This was idea of a "hedonistic" learning system, or, as we would say now, idea of RL.

  – Đầu tiên, chúng tôi tập trung vào cái mà ngày nay được gọi là RL vào cuối năm 1979. Cả 2 đều tại Đại học Massachusetts, làm việc trên 1 trong những dự án đầu tiên nhằm khôi phục ý tưởng rằng mạng lưới các yếu tố thích ứng giống nơ-ron có thể là 1 phương pháp tiếp cận đầy hứa hẹn cho trí tuệ nhân tạo thích ứng. Dự án đã khám phá "lý thuyết dị biệt về các hệ thống thích ứng" do A. Harry Klopf phát triển. Công trình của Harry là 1 nguồn ý tưởng phong phú, & chúng tôi được phép khám phá chúng 1 cách phê phán & so sánh chúng với lịch sử lâu dài của các công trình về hệ thống thích ứng trước đây. Nhiệm vụ của chúng tôi là phân tích các ý tưởng & hiểu mối quan hệ & tầm quan trọng tương đối của chúng. Điều này vẫn tiếp tục cho đến ngày nay, nhưng vào năm 1979, chúng tôi nhận ra: có lẽ những ý tưởng đơn giản nhất, vốn từ lâu đã được coi là hiển nhiên, lại nhận được rất ít sự chú ý từ góc độ tính toán. Đây chỉ đơn giản là ý tưởng về 1 hệ thống học tập *nó muốn* 1 thứ gì đó, điều chỉnh hành vi của nó để tối đa hóa 1 tín hiệu đặc biệt từ môi trường của nó. Đây là ý tưởng về 1 hệ thống học tập "hưởng thụ", hay như chúng ta thường nói bây giờ, ý tưởng về RL.

  Like others, had a sense: RL had been thoroughly explored in early days of cybernetics & AI. On closer inspection, though, found: it had been explored only slightly. While RL had clearly motivated some of earliest computational studies of learning, most of these researchers had gone on to other things, e.g. pattern classification, supervised learning, & adaptive control, or they had abandoned study of learning altogether. As a result, special issues involved in learning how to get something from environment received relatively little attention. In retrospect, focusing on this idea was critical step that set this branch of research in motion. Little progress could be made in computational study of RL until it was recognized that such a fundamental idea had not yet been thoroughly explored.

  – Giống như những người khác, tôi có cảm giác: RL đã được khám phá kỹ lưỡng trong những ngày đầu của ngành điều khiển học & AI. Tuy nhiên, khi xem xét kỹ hơn, tôi thấy rằng: nó chỉ được khám phá 1 chút. Trong khi RL rõ ràng đã thúc đẩy 1 số nghiên cứu tính toán sớm nhất về học tập, hầu hết các nhà nghiên cứu này đã chuyển sang những thứ khác, ví dụ như phân loại mẫu, học có giám sát, & điều khiển thích ứng, hoặc họ đã từ bỏ hoàn toàn việc nghiên cứu về học tập. Kết quả là, các vấn đề đặc biệt liên quan đến việc học cách lấy thứ gì đó từ môi trường nhận được tương đối ít sự chú ý. Nhìn lại, việc

tập trung vào ý tưởng này là bước quan trọng đưa nhánh nghiên cứu này vào hoạt động. Nghiên cứu tính toán về RL chỉ có thể đạt được rất ít tiến bộ cho đến khi người ta nhận ra rằng 1 ý tưởng cơ bản như vậy vẫn chưa được khám phá kỹ lưỡng.

Field has come a long way since then, evolving & maturing in several directions. RL has gradually become 1 of most active research areas in ML, AI, & neural network research. Field has developed strong mathematical foundations & impressive applications. Computational study of RL is now a large field, with hundreds of active researchers around world in diverse disciplines e.g. psychology, control theory, AI, & neuroscience. Particularly important have been contributions establishing & developing relationships to theory of optimal control & dynamic programming. Overall problem of learning from interaction to achieve goals is still far from being solved, but our understanding of it has improved significantly. Can now place component ideas, e.g. temporal-difference learning, dynamic programming, & function approximation, within a coherent perspective w.r.t. overall problem.

– Lĩnh vực này đã đi 1 chặng đường dài kể từ đó, phát triển & trưởng thành theo nhiều hướng. RL đã dần trở thành 1 trong những lĩnh vực nghiên cứu tích cực nhất trong ML, AI, & nghiên cứu mạng nơ-ron. Lĩnh vực này đã phát triển nền tảng toán học vững chắc & các ứng dụng ấn tượng. Nghiên cứu tính toán của RL hiện là 1 lĩnh vực lớn, với hàng trăm nhà nghiên cứu tích cực trên khắp thế giới trong nhiều lĩnh vực khác nhau, ví dụ như tâm lý học, lý thuyết điều khiển, AI, & khoa học thần kinh. Đặc biệt quan trọng là những đóng góp thiết lập & phát triển mối quan hệ với lý thuyết điều khiển tối ưu & lập trình động. Vấn đề chung về học hỏi từ tương tác để đạt được mục tiêu vẫn còn lâu mới được giải quyết, nhưng sự hiểu biết của chúng ta về nó đã được cải thiện đáng kể. Giờ đây, có thể đặt các ý tưởng thành phần, ví dụ như học chênh lệch thời gian, lập trình động, & xấp xỉ hàm, trong 1 quan điểm mạch lạc đối với vấn đề tổng thể.

In some sense we have been working toward this book for 30 years, & have lots of people to thank.

- **Summary of notation.** Capital letters are used for random variables, whereas lower case letters are used for values of random variable & for scalar functions. Quantities that are required to be real-valued vectors are written in bold & in lower case (even if random variables). Matrices are bold capitals.

- **1. Introduction.** Idea that we learn by interacting with our environment is probably the 1st to occur to us when we think about nature of learning. When an infant plays, waves its arms, or looks about, it has not explicit teacher, but it does have gave a direct sensorimotor connection to its environment. Exercising this connection produces a wealth of information about cause & effect, about consequences of actions, & about what to do in order to achieve goals. Throughout our lives, such interactions are undoubtedly a major source of knowledge about our environment & ourselves. Whether we are learning to drive a car or to hold a conversation, we are acutely aware of how our environment responds to what we do, & we seek to influence what happens through our behavior. Learning from interaction is a foundational idea underlying nearly all theories of learning & intelligence.

  – Ý tưởng rằng chúng ta học bằng cách tương tác với môi trường có lẽ là ý tưởng đầu tiên xuất hiện khi chúng ta nghĩ về bản chất của việc học. Khi 1 đứa trẻ sơ sinh chơi đùa, vẫy tay hoặc nhìn xung quanh, nó không có giáo viên rõ ràng, nhưng nó đã tạo ra 1 kết nối cảm biến vận động trực tiếp với môi trường của mình. Việc thực hiện kết nối này tạo ra rất nhiều thông tin về nguyên nhân & kết quả, về hậu quả của hành động, & về những việc cần làm để đạt được mục tiêu. Trong suốt cuộc đời của chúng ta, những tương tác như vậy chắc chắn là 1 nguồn kiến thức chính về môi trường & bản thân chúng ta. Cho dù chúng ta đang học lái xe hay trò chuyện, chúng ta đều nhận thức sâu sắc về cách môi trường của chúng ta phản ứng với những gì chúng ta làm, & chúng ta tìm cách ảnh hưởng đến những gì xảy ra thông qua hành vi của mình. Học hỏi từ tương tác là 1 ý tưởng nền tảng làm nền tảng cho hầu hết mọi lý thuyết về học tập & trí thông minh.

  In this book, explore a *computational* approach to learning from interaction. Rather than directly theorizing about how people or animals learn, primarily explore idealized learning situations & evaluate effectiveness of various learning methods. I.e., adopt perspective of an AI researcher or engineer. Explore designs for machines that are effective in solving learning problems of scientific or economic interest, evaluating designs through mathematical analysis of computational experiments. Approach we explore, called *reinforcement learning*, is much more focused on goal-directed learning from interaction than other approaches to ML.

  – Trong cuốn sách này, chúng ta sẽ khám phá phương pháp tiếp cận *computation* để học từ tương tác. Thay vì trực tiếp lý thuyết hóa cách con người hoặc động vật học tập, chúng ta sẽ chủ yếu khám phá các tình huống học tập lý tưởng & đánh giá hiệu quả của các phương pháp học tập khác nhau. Ví dụ, hãy áp dụng góc nhìn của 1 nhà nghiên cứu hoặc kỹ sư AI. Khám phá các thiết kế máy móc hiệu quả trong việc giải quyết các vấn đề học tập mang tính khoa học hoặc kinh tế, đánh giá các thiết kế thông qua phân tích toán học các thí nghiệm tính toán. Phương pháp chúng tôi khám phá, được gọi là *reinforcement learning*, tập trung nhiều hơn vào việc học tập hướng mục tiêu từ tương tác so với các phương pháp tiếp cận ML khác.

  - **1.1. RL.** RL is learning what to do – how to map situations to actions – so as to maximize a numerical reward signal. Learner is not told which actions to take, but instead must discover which actions yield most reward by trying them. In most interesting & challenging cases, actions may affect not only immediate reward but also next situation &, through that, all subsequent rewards. These 2 characteristics – trial-&-error search & delayed reward – are 2 most important distinguishing features of RL.

    – RL đang học cách làm gì – cách liên kết tình huống với hành động – để tối đa hóa tín hiệu phần thưởng số. Người học không được chỉ dẫn phải thực hiện hành động nào, mà thay vào đó, phải tự khám phá hành động nào mang lại phần thưởng cao nhất bằng cách thử nghiệm chúng. Trong hầu hết các trường hợp & thử thách thú vị, hành động có thể ảnh hưởng không chỉ đến phần thưởng tức thời mà còn cả tình huống tiếp theo &, qua đó, tất cả các phần thưởng tiếp theo. Hai đặc điểm này – tìm kiếm thử nghiệm & lỗi & phần thưởng trì hoãn – là 2 đặc điểm phân biệt quan trọng nhất của RL.

RL, like many topics whose names end with "ing", e.g. ML & mountaineering, is simultaneously a problem, a class of solution methods that work well on problem, & field that studies this problem & its solution methods. Convenient to use a single name for all 3 things, but at same time essential to keep 3 conceptually separate. In particular, distinction between problems & solution methods is very important in RL; failing to make this distinction is source of many confusions.

– RL, giống như nhiều chủ đề có tên kết thúc bằng "ing", ví dụ: ML & mountaineering, đồng thời là 1 bài toán, 1 lớp các phương pháp giải quyết hiệu quả cho bài toán, & lĩnh vực nghiên cứu bài toán này & các phương pháp giải quyết của nó. Việc sử dụng 1 tên gọi duy nhất cho cả 3 lĩnh vực rất tiện lợi, nhưng đồng thời cũng cần thiết phải tách biệt 3 khái niệm này về mặt khái niệm. Đặc biệt, việc phân biệt giữa bài toán & phương pháp giải quyết là rất quan trọng trong RL; việc không phân biệt được điều này là nguyên nhân gây ra nhiều nhầm lẫn.

Formalize problem of RL using ideas from dynamical systems theory, specifically, as optimal control of incompletely-known Markov decision processes. Details of this formalization must wait until Chap. 3, but basic idea is simply to capture most important aspects of real problem facing a learning agent interacting over time with its environment to achieve a goal. A learning agent must be able to sense state of its environment to some extent & must be able to take actions that affect state. Agent also must have a goal or goals relating to state of environment. Markov decision processes are intended to include just these 3 aspects – sensation, action, & goal – in their simplest possible forms without trivializing any of them. Any method that is well suited to solving such problems we consider to be RL method.

– Chính thức hóa bài toán RL bằng các ý tưởng từ lý thuyết hệ thống động, cụ thể là điều khiển tối ưu các quá trình quyết định Markov chưa được biết đầy đủ. Chi tiết về chính thức hóa này phải chờ đến Chương 3, nhưng ý tưởng cơ bản chỉ đơn giản là nắm bắt hầu hết các khía cạnh quan trọng của vấn đề thực tế mà 1 tác nhân học tập tương tác theo thời gian với môi trường của nó để đạt được mục tiêu. 1 tác nhân học tập phải có khả năng cảm nhận trạng thái của môi trường ở 1 mức độ nào đó & phải có khả năng thực hiện các hành động ảnh hưởng đến trạng thái. Tác nhân cũng phải có 1 hoặc nhiều mục tiêu liên quan đến trạng thái của môi trường. Các quá trình quyết định Markov được thiết kế để chỉ bao gồm 3 khía cạnh này – cảm giác, hành động, & mục tiêu – ở dạng đơn giản nhất có thể mà không tầm thường hóa bất kỳ khía cạnh nào. Bất kỳ phương pháp nào phù hợp để giải quyết các vấn đề như vậy, chúng tôi coi là phương pháp RL.

RL is different from *supervised learning*, kind of learning studied in most current research in field of ML. Supervised learning is learning from a training set of labeled examples provided by a knowledgeable external supervisor. Each example is a description of a situation together with a specification – label – of correct action system should take in that situation, which is often to identify a category to which situation belongs. Object of this kind of learning is for system to extrapolate, or generalize, its responses so that it acts correctly in situations not present in training set. This is an important kind of learning, but alone it is not adequate for learning from interaction. In interactive problems, often impractical to obtain examples of desired behavior that are both correct & representative of all situations in which agent has to act. In uncharted territory – where one would expect learning to be most beneficial – an agent must be able to learn from its own experience.

– RL khác với *học có giám sát*, loại học được nghiên cứu trong hầu hết các nghiên cứu hiện tại trong lĩnh vực ML. Học có giám sát là học từ 1 tập huấn luyện các ví dụ được gắn nhãn do 1 người giám sát bên ngoài có hiểu biết cung cấp. Mỗi ví dụ là 1 mô tả về 1 tình huống cùng với 1 thông số kỹ thuật – nhãn – về hành động đúng mà hệ thống nên thực hiện trong tình huống đó, thường là để xác định 1 danh mục mà tình huống đó thuộc về. Mục tiêu của loại học này là để hệ thống suy rộng hoặc khái quát hóa các phản ứng của nó để nó hành động chính xác trong các tình huống không có trong tập huấn luyện. Đây là 1 loại học quan trọng, nhưng chỉ riêng nó thì không đủ để học từ tương tác. Trong các vấn đề tương tác, thường không thực tế để có được các ví dụ về hành vi mong muốn vừa chính xác & đại diện cho tất cả các tình huống mà tác nhân phải hành động. Trong lãnh thổ chưa được khám phá – nơi mà người ta mong đợi việc học có lợi nhất – 1 tác nhân phải có khả năng học hỏi từ kinh nghiệm của chính mình.

RL is also different from what ML researchers call *unsupervised learning*, which is typically about finding structure hidden in collections of unlabeled data. Terms supervised learning & unsupervised learning would seem to exhaustively classify ML paradigms, but they do not. Although one might be tempted to think of RL as a kind of unsupervised learning because it does not rely on examples of correct behavior, RL is trying to maximize a reward signal instead of trying to find hidden structure. Uncovering structure in an agent's experience can certainly be useful in RL, but by itself does not address RL problem of maximizing a reward signal. Therefore consider RL to be a 3rd ML paradigm, alongside supervised learning & unsupervised learning & perhaps other paradigms.

– RL cũng khác với cái mà các nhà nghiên cứu ML gọi là *unsupervised learning*, thường là về việc tìm kiếm cấu trúc ẩn trong các tập hợp dữ liệu chưa được gắn nhãn. Các thuật ngữ học có giám sát & unsupervised learning dường như phân loại đầy đủ các mô hình ML, nhưng chúng không phải vậy. Mặc dù người ta có thể bị cám dỗ nghĩ về RL như 1 loại học không giám sát vì nó không dựa trên các ví dụ về hành vi đúng, RL đang cố gắng tối đa hóa tín hiệu phần thưởng thay vì cố gắng tìm kiếm cấu trúc ẩn. Việc khám phá cấu trúc trong trải nghiệm của 1 tác nhân chắc chắn có thể hữu ích trong RL, nhưng bản thân nó không giải quyết được vấn đề RL về việc tối đa hóa tín hiệu phần thưởng. Do đó, hãy coi RL là mô hình ML thứ 3, cùng với học có giám sát & unsupervised learning & có lẽ là các mô hình khác.

1 of challenges that arise in RL, & not in other kinds of learning, is trade-off between exploration & exploitation. To obtain a lot of reward, a RL agent must prefer actions that it has tried in past & found to be effective in producing reward. But to discover such actions, it has to try actions that it has not selected before. Agent has to *exploit* what it has already experienced in order to obtain reward, but it also has to *explore* in order to make better action selections in future. Dilemma: neither exploration nor exploitation can be pursued exclusively without failing at task. Agent must try a variety of action & progressively favor those that appear to be best. On a stochastic task, each action must be tried many times to gain a reliable estimate of its expected reward. Exploration–exploitation dilemma has been intensively studied by mathematicians

for many decades, yet remains unresolved. For now, simply note: entire issue of balancing exploration & exploitation does not even arise in supervised & unsupervised learning, at least in purest forms of these paradigms.

– 1 trong những thách thức nảy sinh trong RL, & không phải trong các loại hình học tập khác, là sự đánh đổi giữa khám phá & khai thác. Để nhận được nhiều phần thưởng, 1 tác nhân RL phải ưu tiên các hành động mà nó đã thử trong quá khứ & thấy hiệu quả trong việc tạo ra phần thưởng. Nhưng để khám phá ra những hành động như vậy, nó phải thử các hành động mà nó chưa từng chọn trước đây. Tác nhân phải *nó khai thác* những gì nó đã trải nghiệm để nhận được phần thưởng, nhưng nó cũng phải *nó khám phá* để đưa ra lựa chọn hành động tốt hơn trong tương lai. Thế tiến thoái lưỡng nan: cả khám phá lẫn khai thác đều không thể được theo đuổi độc quyền mà không thất bại trong nhiệm vụ. Tác nhân phải thử nhiều hành động & dần dần ưu tiên những hành động có vẻ tốt nhất. Trong 1 nhiệm vụ ngẫu nhiên, mỗi hành động phải được thử nhiều lần để có được ước tính đáng tin cậy về phần thưởng dự kiến của nó. Thế tiến thoái lưỡng nan giữa khám phá & khai thác đã được các nhà toán học nghiên cứu sâu rộng trong nhiều thập kỷ, nhưng vẫn chưa được giải quyết. Hiện tại, chỉ cần lưu ý: toàn bộ vấn đề cân bằng giữa khám phá & khai thác thậm chí không phát sinh trong học tập có giám sát & không giám sát, ít nhất là ở dạng tinh khiết nhất của các mô hình này.

Another key feature of RL: it explicitly considers *whole* problem of a goal-directed agent interacting with an uncertain environment. This is in contrast to many approaches that consider subproblems without addressing how they might fit into a larger picture. E.g., have mentioned: many ML researchers have studied supervised learning without specifying how such an ability would ultimately be useful. Other researchers have developed theories of planning with general goals, but without considering planning's role in real-time decision making, or question of where predictive models necessary for planning would come from. Although these approaches have yielded many useful results, their focus on isolated subproblems is a significant limitation.

– 1 đặc điểm quan trọng khác của RL: nó xem xét 1 cách rõ ràng *toàn bộ* vấn đề của 1 tác nhân hướng đến mục tiêu tương tác với 1 môi trường không chắc chắn. Điều này trái ngược với nhiều phương pháp tiếp cận chỉ xem xét các bài toán con mà không giải quyết cách chúng có thể phù hợp với bức tranh tổng thể. Ví dụ, đã đề cập: nhiều nhà nghiên cứu ML đã nghiên cứu học có giám sát mà không chỉ rõ khả năng đó cuối cùng sẽ hữu ích như thế nào. Các nhà nghiên cứu khác đã phát triển các lý thuyết về lập kế hoạch với các mục tiêu chung, nhưng lại không xem xét vai trò của lập kế hoạch trong việc ra quyết định theo thời gian thực, hoặc câu hỏi về việc các mô hình dự đoán cần thiết cho việc lập kế hoạch sẽ đến từ đâu. Mặc dù các phương pháp tiếp cận này đã mang lại nhiều kết quả hữu ích, nhưng việc tập trung vào các bài toán con riêng lẻ là 1 hạn chế đáng kể.

RL takes opposite tack, starting with a complete, interactive, goal-seeking agent. All RL agents have explicit goals, can sense aspects of their environments, & can choose actions to influence their environments. Moreover, usually assumed from beginning: agent has to operate despite significant uncertainty about environment it faces. When RL involves planning, it has to address interplay between planning & real-time action selection, as well as question of how environment models are acquired & improved. When RL involves supervised learning, it does so for specific reasons that determine which capabilities are critical & which are not. For learning research to make progress, important subproblems have to be isolated & studied, but they should be subproblems that play clear roles in complete, interactive, goal-seeking agents, even if all details of complete agent cannot yet be filled in.

– RL có hướng đi ngược lại, bắt đầu với 1 tác nhân hoàn chỉnh, tương tác & tìm kiếm mục tiêu. Tất cả các tác nhân RL đều có mục tiêu rõ ràng, có thể cảm nhận các khía cạnh của môi trường của chúng, & có thể chọn hành động để tác động đến môi trường của chúng. Hơn nữa, thường được giả định ngay từ đầu: tác nhân phải hoạt động bất chấp sự không chắc chắn đáng kể về môi trường mà nó phải đối mặt. Khi RL liên quan đến việc lập kế hoạch, nó phải giải quyết sự tương tác giữa việc lập kế hoạch & lựa chọn hành động theo thời gian thực, cũng như câu hỏi về cách các mô hình môi trường được tiếp thu & cải thiện. Khi RL liên quan đến việc học có giám sát, nó làm như vậy vì những lý do cụ thể để xác định khả năng nào là quan trọng & khả năng nào không. Để nghiên cứu về học tập đạt được tiến bộ, các vấn đề phụ quan trọng phải được phân lập & nghiên cứu, nhưng chúng phải là những vấn đề phụ đóng vai trò rõ ràng trong các tác nhân hoàn chỉnh, tương tác & tìm kiếm mục tiêu, ngay cả khi tất cả các chi tiết của tác nhân hoàn chỉnh vẫn chưa thể được điền đầy đủ.

By a complete, interactive, goal-seeking agent we do not always mean something like a complete organism or robot. These are clearly examples, but a complete, interactive, goal-seeking agent can also be a component of a larger behaving system. In this case, agent directly interacts with rest of larger system & indirectly interacts with larger system's environment. A simple example is an agent that monitors charge level of robot's battery & sends commands to robot's control architecture. This agent's environment is rest of robot together with robot's environment. Important to look beyond most obvious examples of agents & their environments to appreciate generality of RL framework.

– Khi nói đến 1 tác nhân hoàn chỉnh, tương tác & tìm kiếm mục tiêu, chúng ta không phải lúc nào cũng muốn nói đến 1 sinh vật hay robot hoàn chỉnh. Đây rõ ràng là những ví dụ, nhưng 1 tác nhân hoàn chỉnh, tương tác & tìm kiếm mục tiêu cũng có thể là 1 thành phần của 1 hệ thống hoạt động lớn hơn. Trong trường hợp này, tác nhân tương tác trực tiếp với phần còn lại của hệ thống lớn hơn & gián tiếp tương tác với môi trường của hệ thống lớn hơn. 1 ví dụ đơn giản là 1 tác nhân theo dõi mức sạc pin của robot & gửi lệnh đến kiến trúc điều khiển của robot. Môi trường của tác nhân này là phần còn lại của robot cùng với môi trường của robot. Điều quan trọng là phải xem xét kỹ hơn hầu hết các ví dụ rõ ràng về tác nhân & môi trường của chúng để đánh giá được tính tổng quát của khuôn khổ RL.

1 of most exciting aspects of modern RL is its substantive & fruitful interactions with other engineering & scientific disciplines. RL is part of a decades-long trend within AI & ML toward greater integration with statistics, optimization, & other mathematical subjects. E.g., ability of some RL methods to learn with parameterized approximators addresses classical "curse of dimensionality" in operations research & control theory. More distinctively, RL has also interacted strongly with psychology & neuroscience, with substantial benefits going both ways. Of all forms of ML, RL is closest to kind of learning

that humans & other animals do, & many of core algorithms of RL were originally inspired by biological learning systems. RL has also given back, both through a psychological model of animal learning that better matches some of empirical data, & through an influential model of parts of brain's reward system. Body of this book develops ideas of RL that pertain to engineering & AI, with connections to psychology & neuroscience summarized in Chaps. 14–15.

– 1 trong những khía cạnh thú vị nhất của Học máy (RL) hiện đại là những tương tác thiết thực & hiệu quả của nó với các ngành kỹ thuật & khoa học khác. RL là 1 phần của xu hướng kéo dài hàng thập kỷ trong AI & ML hướng tới sự tích hợp sâu hơn với thống kê, tối ưu hóa & các môn toán học khác. Ví dụ, khả năng học với các bộ xấp xỉ tham số của 1 số phương pháp RL giải quyết "lời nguyền đa chiều" cổ điển trong nghiên cứu vận hành & lý thuyết điều khiển. Đặc biệt hơn, RL cũng tương tác mạnh mẽ với tâm lý học & khoa học thần kinh, mang lại những lợi ích đáng kể cho cả 2 bên. Trong tất cả các dạng ML, RL gần nhất với loại hình học tập mà con người & các loài động vật khác thực hiện, & nhiều thuật toán cốt lõi của RL ban đầu được lấy cảm hứng từ các hệ thống học tập sinh học. RL cũng đã đóng góp trở lại, thông qua 1 mô hình tâm lý học về học tập của động vật phù hợp hơn với 1 số dữ liệu thực nghiệm, & thông qua 1 mô hình có ảnh hưởng về các bộ phận của hệ thống khen thưởng của não. Nội dung cuốn sách này phát triển các ý tưởng RL liên quan đến kỹ thuật & AI, với mối liên hệ với tâm lý học & khoa học thần kinh được tóm tắt trong các Chương 14–15.

Finally, RL is also part of a larger trend in AI back toward simple general principles. Since late 1960s, many AI researchers presumed: there are no general principles to be discovered, that intelligence is instead due to possession of a vast number of special purpose tricks, procedures, & heuristics. It was sometimes said: if we could just get enough relevant facts into a machine, say 1 million, or 1 billion, then it would become intelligent. Methods based on general principles, e.g. search or learning, were characterized as "weak methods", whereas those based on specific knowledge were called "strong methods". This view is uncommon today. From our point of view, it was premature: too little effort had been put into search for general principles to conclude that there were none. Modern AI now includes much research looking for general principles of learning, search, & decision making. Not clear how far back pendulum will swing, but RL research is certainly part of swing back toward simpler & fewer general principles of AI.

– Cuối cùng, RL cũng là 1 phần của xu hướng lớn hơn trong AI hướng về các nguyên lý chung đơn giản. Từ cuối những năm 1960, nhiều nhà nghiên cứu AI đã cho rằng: không có nguyên lý chung nào cần được khám phá, mà trí thông minh thực chất là do sở hữu vô số các thủ thuật, quy trình, & phương pháp tìm kiếm chuyên biệt. Đôi khi người ta nói: nếu chúng ta có thể đưa đủ dữ liệu liên quan vào 1 cỗ máy, chẳng hạn như 1 triệu, hoặc 1 tỷ, thì nó sẽ trở nên thông minh. Các phương pháp dựa trên các nguyên lý chung, ví dụ như tìm kiếm hoặc học tập, được gọi là "phương pháp yếu", trong khi các phương pháp dựa trên kiến thức cụ thể được gọi là "phương pháp mạnh". Quan điểm này không phổ biến ngày nay. Theo quan điểm của chúng tôi, điều đó là quá sớm: quá ít nỗ lực được bỏ ra để tìm kiếm các nguyên lý chung để kết luận rằng không có nguyên lý nào cả. AI hiện đại ngày nay bao gồm nhiều nghiên cứu tìm kiếm các nguyên lý chung về học tập, tìm kiếm, & ra quyết định. Không rõ con lắc sẽ quay ngược lại bao xa, nhưng nghiên cứu RL chắc chắn là 1 phần của quá trình quay trở lại với các nguyên lý chung đơn giản hơn & ít nguyên lý chung hơn của AI.

- **1.2. Examples.** A good way to understand RL: consider some of examples & possible applications that have guided its development.
  - A master chess player makes a move. Choice is informed both by planning – anticipating possible replies & counterreplies – & by immediate, intuitive judgments of desirability of particular positions & moves.
  - An adaptive controller adjusts parameters of a petroleum refinery's operation in real time. Controller optimizes yield/cost/quality trade-off on basis of specified marginal costs without sticking strictly to set points originally suggested by engineers.
  - A gazelle calf struggles to its feet minutes after being born. Half an hour later it is running at 20 miles per hour.
  - A mobile robot decides whether it should enter a new room in search of more trash to collect or start trying to find its way back to its battery recharging station. It makes its decision based on current charge level of its battery & how quickly & easily it has been able to find recharger in past.
  - Phil prepares his breakfast. Closely examined, even this apparently mundane activity reveals a complex web of conditional behavior & interlocking goal-subgoal relationships: walking to cupboard, opening it, selecting a cereal box, then reaching for, grasping, & retrieving box. Other complex, tuned, interactive sequences of behavior are required to obtain a bowl, spoon, & milk carton. Each step involves a series of eye movements to obtain information & to guide reaching & locomotion. Rapid judgments are continually made about how to carry objects or whether it is better to ferry some of them to dining table before obtaining others. Each step is guided by goals, e.g. grasping a spoon or getting to refrigerator, & is in service of other goals, e.g. having spoon to eat with once cereal is prepared & ultimately obtaining nourishment. Whether he is aware of it or not, Phil is accessing information about state of his body that determines his nutritional needs, level of hunger, & food preferences.

– 1 cách hay để hiểu RL: hãy xem xét 1 số ví dụ & các ứng dụng khả thi đã định hướng cho sự phát triển của nó.

- 1 kỳ thủ cờ vua lão luyện thực hiện 1 nước đi. Lựa chọn được đưa ra dựa trên cả việc lập kế hoạch – dự đoán các câu trả lời khả thi & các câu trả lời ngược lại – & bằng những phán đoán trực quan, tức thời về mức độ mong muốn của các vị trí & nước đi cụ thể.
- 1 bộ điều khiển thích ứng điều chỉnh các thông số vận hành của 1 nhà máy lọc dầu theo thời gian thực. Bộ điều khiển tối ưu hóa sự cân bằng giữa năng suất/chi phí/chất lượng dựa trên chi phí cận biên cụ thể mà không tuân thủ nghiêm ngặt các điểm đặt ban đầu do các kỹ sư đề xuất.
- 1 con linh dương con đang cố gắng đứng dậy sau khi sinh ra vài phút. Nửa giờ sau, nó chạy với tốc độ 20 dặm 1 giờ.

* 1 robot di động quyết định liệu nó nên đi vào 1 căn phòng mới để tìm thêm rác để thu gom hay bắt đầu tìm đường quay trở lại trạm sạc pin. Nó đưa ra quyết định dựa trên mức sạc pin hiện tại & tốc độ & dễ dàng mà nó đã tìm thấy bộ sạc trong quá khứ.
* Phil chuẩn bị bữa sáng. Khi xem xét kỹ lưỡng, ngay cả hoạt động tưởng chừng như tầm thường này cũng hé lộ 1 mạng lưới phức tạp của các hành vi có điều kiện & các mối quan hệ mục tiêu-mục tiêu phụ đan xen: đi đến tủ bếp, mở tủ, chọn hộp ngũ cốc, rồi với lấy, nắm lấy, & lấy hộp. Các chuỗi hành vi phức tạp, được điều chỉnh & tương tác khác cũng cần thiết để lấy được bát, thìa, & hộp sữa. Mỗi bước bao gồm 1 loạt chuyển động mắt để thu thập thông tin & hướng dẫn việc với & di chuyển. Những phán đoán nhanh chóng liên tục được đưa ra về cách mang đồ vật hoặc liệu có nên mang 1 số đồ vật đến bàn ăn trước khi lấy những đồ vật khác hay không. Mỗi bước đều được hướng dẫn bởi các mục tiêu, ví dụ: cầm thìa hay đến tủ lạnh, & phục vụ cho các mục tiêu khác, ví dụ: có thìa để ăn sau khi ngũ cốc đã được chuẩn bị & cuối cùng là có được dinh dưỡng. Dù có nhận thức được điều đó hay không, Phil đang tiếp cận thông tin về trạng thái cơ thể của mình, từ đó quyết định nhu cầu dinh dưỡng, mức độ đói, & sở thích ăn uống của cậu bé.

These examples share features that are so basic that they are easy to overlook. All involve *interaction* between an active decision-making agent & its environment, within which agent seeks to achieve a *goal* despite *uncertainty* about its environment. Agent's actions are permitted to affect future state of environment (e.g., next chess position, level of reservoirs of refinery, robot's next location & future charge level of its battery), thereby affecting actions & opportunities available to agent at later times. Correct choice requires taking into account indirect, delayed consequences of actions, & thus may require foresight or planning.

– These examples share features that are so basic that they are easy to overlook. All involve *interaction* between an active decision-making agent & its environment, within which agent seeks to achieve a *goal* despite *uncertainty* about its environment. Agent's actions are permitted to affect future state of environment (e.g., next chess position, level of reservoirs of refinery, robot's next location & future charge level of its battery), thereby affecting actions & opportunities available to agent at later times. Correct choice requires taking into account indirect, delayed consequences of actions, & thus may require foresight or planning.

At same time, in all of these examples effects of actions cannot be fully predicted; thus agent must monitor its environment frequently & react appropriately. E.g., Phil must watch milk he pours into his cereal bowl to keep it from overflowing. All these examples involve goals that are explicit in sense: agent can judge progress toward its goal based on what it can sense directly. Chess player knows whether or not he wins, refinery controller knows how much petroleum is being produced, gazelle calf knows when it falls, mobile robot knows when its batteries run down, & Phil knows whether or not he is enjoying his breakfast.

– Đồng thời, trong tất cả các ví dụ này, tác động của hành động không thể được dự đoán đầy đủ; do đó, tác nhân phải thường xuyên theo dõi môi trường xung quanh & phản ứng phù hợp. Ví dụ, Phil phải để ý sữa anh ta đổ vào bát ngũ cốc của mình để tránh bị tràn. Tất cả các ví dụ này đều liên quan đến các mục tiêu rõ ràng: tác nhân có thể đánh giá tiến độ đạt được mục tiêu dựa trên những gì nó có thể cảm nhận trực tiếp. Người chơi cờ vua biết mình thắng hay thua, người quản lý nhà máy lọc dầu biết lượng dầu đang được sản xuất, con linh dương con biết khi nào nó ngã, robot di động biết khi nào pin của nó hết, & Phil biết mình có đang thưởng thức bữa sáng hay không.

In all of these examples agent can use its experience to improve its performance over time. Chess player refines intuition he uses to evaluate positions, thereby improving his play; gazelle calf improves efficiency with which it can run; Phil learns to streamline making his breakfast. Knowledge agent brings to task at start – either from previous experience with related tasks or built into it by design or evolution – influences what is useful or easy to learn, but interaction with environment is essential for adjusting behavior to exploit specific features of task.

– Trong tất cả các ví dụ này, tác nhân có thể sử dụng kinh nghiệm của mình để cải thiện hiệu suất theo thời gian. Người chơi cờ vua tinh chỉnh trực giác mà anh ta sử dụng để đánh giá vị trí, từ đó cải thiện lối chơi của mình; con linh dương Gazelle cải thiện hiệu suất chạy; Phil học cách đơn giản hóa việc chuẩn bị bữa sáng. Kiến thức mà tác nhân mang đến cho nhiệm vụ ngay từ đầu – từ kinh nghiệm trước đó với các nhiệm vụ liên quan hoặc được tích hợp sẵn trong đó theo thiết kế hoặc quá trình tiến hóa – sẽ quyết định những gì hữu ích hoặc dễ học, nhưng tương tác với môi trường là điều cần thiết để điều chỉnh hành vi nhằm khai thác các đặc điểm cụ thể của nhiệm vụ.

- **1.3. Elements of RL.** Beyond agent & environment, one can identify 4 main subelements of a RL system: a *policy*, a *reward signal*, a *value function*, &, optionally, a *model* of environment.

– Ngoài tác nhân & môi trường, người ta có thể xác định 4 yếu tố phụ chính của hệ thống RL: *chính sách*, *tín hiệu phần thưởng*, *hàm giá trị*, &, tùy chọn, *mô hình* của môi trường.

A *policy* defines learning agent's way of behaving at a given time. Roughly speaking, a policy is a mapping from perceived states of environment to actions to be taken when in those states. It corresponds to what in psychology would be called a set of stimulus–response rules or associations. In some cases policy may be a simple function or lookup table, whereas in others it may involve extensive computation e.g. a search process. Policy is core of a RL agent in sense: it alone is sufficient to determine behavior. In general, policies may be stochastic, specifying probabilities for each action.

– Chính sách xác định cách thức hành xử của tác nhân học tập tại 1 thời điểm nhất định. Nói 1 cách đại khái, chính sách là 1 phép ánh xạ từ các trạng thái nhận thức của môi trường đến các hành động cần thực hiện khi ở trong các trạng thái đó. Nó tương ứng với cái mà trong tâm lý học được gọi là 1 tập hợp các quy tắc hoặc mối liên hệ giữa kích thích & phản ứng. Trong 1 số trường hợp, chính sách có thể là 1 hàm đơn giản hoặc bảng tra cứu, trong khi ở những trường hợp khác, nó có thể liên quan đến các phép tính phức tạp, ví dụ như 1 quy trình tìm kiếm. Chính sách là cốt lõi của 1 tác nhân học

tập theo nghĩa: chỉ riêng nó là đủ để xác định hành vi. Nhìn chung, các chính sách có thể là ngẫu nhiên, chỉ định xác suất cho mỗi hành động.

A *reward signal* defines goal of RL problem. On each time step, environment sends to RL agent a single number called *reward*. Agent's sole objective: maximize total reward it receives over long run. Reward signal thus defines what are good & bad events for agent. in a biological system, might think of rewards as analogous to experiences of pleasure or pain. They are immediate & defining features of problem faced by agent. Reward signal is primary basis for altering policy; if an action selected by policy is followed by low reward, then policy may be changed to select some other action in that situation in future. In general, reward signals may be stochastic functions of state of environment & actions taken.

– Tín hiệu thưởng xác định mục tiêu của bài toán thực tế ảo (RL). Ở mỗi bước thời gian, môi trường gửi đến tác nhân thực tế ảo 1 số duy nhất gọi là phần thưởng. Mục tiêu duy nhất của tác nhân: tối đa hóa tổng phần thưởng mà nó nhận được trong thời gian dài. Do đó, tín hiệu thưởng xác định những sự kiện tốt & xấu đối với tác nhân. Trong 1 hệ thống sinh học, có thể coi phần thưởng tương tự như trải nghiệm khoái cảm hoặc đau đớn. Chúng là những đặc điểm & xác định tức thời của vấn đề mà tác nhân gặp phải. Tín hiệu thưởng là cơ sở chính để thay đổi chính sách; nếu 1 hành động được chính sách lựa chọn tiếp theo là phần thưởng thấp, thì chính sách có thể được thay đổi để lựa chọn 1 hành động khác trong tình huống đó trong tương lai. Nhìn chung, tín hiệu thưởng có thể là các hàm ngẫu nhiên của trạng thái môi trường & hành động được thực hiện.

Whereas reward signal indicates what is good in an immediate sense, a *value function* specifies what is good in long run. Roughly speaking, *value* of a state is total amount of reward an agent can expect to accumulate over future, starting from that state. Whereas rewards determine immediate, intrinsic desirability of environmental states, values indicate *long-term* desirability of states after taking into account states that are likely to follow & rewards available in those states. E.g., a state might always yield a low immediate reward but still have a high value because it is regularly followed by other states that yield high rewards. Or reverse could be true. To make a human analogy, rewards are somewhat like pleasure (if high) & pain (if low), whereas values correspond to a more refined & farsighted judgment of how pleased or displeased we are that our environment is in a particular state.

– Trong khi tín hiệu phần thưởng chỉ ra điều gì là tốt theo nghĩa tức thời, thì hàm giá trị chỉ ra điều gì là tốt trong thời gian dài. Nói 1 cách đại khái, *giá trị* của 1 trạng thái là tổng lượng phần thưởng mà 1 tác nhân có thể mong đợi tích lũy trong tương lai, bắt đầu từ trạng thái đó. Trong khi phần thưởng xác định mức độ mong muốn nội tại, tức thời của các trạng thái môi trường, thì các giá trị chỉ ra mức độ mong muốn dài hạn của các trạng thái sau khi tính đến các trạng thái có khả năng theo sau & phần thưởng có sẵn trong các trạng thái đó. Ví dụ: 1 trạng thái có thể luôn mang lại phần thưởng tức thời thấp nhưng vẫn có giá trị cao vì nó thường xuyên được theo sau bởi các trạng thái khác mang lại phần thưởng cao. Hoặc ngược lại cũng có thể đúng. Để đưa ra 1 phép so sánh với con người, phần thưởng có phần giống như niềm vui (nếu cao) & nỗi đau (nếu thấp), trong khi các giá trị tương ứng với 1 & phán đoán tinh tế hơn & có tầm nhìn xa về mức độ chúng ta hài lòng hay không hài lòng khi môi trường của chúng ta ở trong 1 trạng thái cụ thể.

Rewards are in a sense primary, whereas values, as predictions of rewards, are secondary. Without rewards there could be no values, & only purpose of estimating values is to achieve more reward. Nevertheless, it is values with which we are most concerned when making & evaluating decisions. Action choices are made based on value judgments. Seek actions that bring about states of highest value, not highest reward, because these actions obtain greatest amount of reward for us over long run. Unfortunately, much harder to determine values than it is to determine rewards. Rewards are basically given directly by environment, but values must be estimated & re-estimated from sequences of observations an agent makes over its entire lifetime. In fact, most important component of almost all RL algorithms we consider is a method for efficiently estimating values. Central role of value estimation is arguably most important thing that has been learned about RL over last 6 decades.

– Phần thưởng theo 1 nghĩa nào đó là chính, trong khi giá trị, như dự đoán về phần thưởng, là thứ yếu. Không có phần thưởng thì không thể có giá trị, & mục đích duy nhất của việc ước tính giá trị là để đạt được nhiều phần thưởng hơn. Tuy nhiên, chính giá trị là thứ chúng ta quan tâm nhất khi đưa ra & đánh giá các quyết định. Các lựa chọn hành động được đưa ra dựa trên các đánh giá về giá trị. Hãy tìm kiếm các hành động mang lại trạng thái có giá trị cao nhất, không phải phần thưởng cao nhất, vì những hành động này mang lại cho chúng ta phần thưởng lớn nhất trong thời gian dài. Thật không may, việc xác định giá trị khó hơn nhiều so với việc xác định phần thưởng. Phần thưởng về cơ bản được trao trực tiếp bởi môi trường, nhưng giá trị phải được ước tính & ước tính lại từ các chuỗi quan sát mà 1 tác nhân thực hiện trong toàn bộ vòng đời của nó. Trên thực tế, thành phần quan trọng nhất của hầu hết các thuật toán RL mà chúng tôi xem xét là 1 phương pháp để ước tính giá trị 1 cách hiệu quả. Vai trò trung tâm của việc ước tính giá trị có thể được cho là điều quan trọng nhất đã được tìm hiểu về RL trong 6 thập kỷ qua.

4th & final element of some RL systems is a *model* of environment. This is something that mimics behavior of environment, or more generally, that allows inferences to be made about how environment will behave. E.g., given a state & action, model might predict resultant next state & next reward. Models are used for *planning*, by which we mean any way of deciding on a course of action by considering possible future situations before they re actually experienced. Methods for solving RL problems that use models & planning are called *model-based* methods, as opposed to simpler *model-free* methods that are explicitly trial-&-error learners – viewed as almost *opposite* of planning. In Chap. 8 explore RL systems that simultaneously learn by trial & error, learn a model of environment, & use model for planning. Modern RL spans spectrum from low-level, trial-&-error learning to high-level, deliberative planning.

– Yếu tố cuối cùng thứ 4 của 1 số hệ thống RL là 1 mô hình môi trường. Đây là thứ mô phỏng hành vi của môi trường, hay nói rộng hơn, cho phép suy luận về cách môi trường sẽ hoạt động. Ví dụ, với 1 trạng thái & hành động, mô hình có thể dự đoán trạng thái tiếp theo & phần thưởng tiếp theo. Các mô hình được sử dụng để lập kế hoạch, i.e., bất kỳ cách nào để quyết định 1 hành động bằng cách xem xét các tình huống có thể xảy ra trong tương lai trước khi chúng thực sự được trải

nghiệm. Các phương pháp giải quyết các bài toán RL sử dụng mô hình & lập kế hoạch được gọi là các phương pháp dựa trên mô hình, trái ngược với các phương pháp đơn giản hơn là các phương pháp học thử sai rõ ràng – được xem là gần như đối lập với lập kế hoạch. Trong Chương 8, chúng ta sẽ khám phá các hệ thống RL đồng thời học bằng thử sai, học 1 mô hình môi trường & sử dụng mô hình để lập kế hoạch. RL hiện đại trải rộng từ học thử sai cấp thấp đến lập kế hoạch cân nhắc cấp cao.

- **1.4. Limitations & Scope.** RL relies heavily on concept of state – as input to policy & value function, & as both input to & output from model. Informally, can think of state as a signal conveying to agent some sense of "how environment is" at a particular time. Formal definition of state as we use it here is given by framework of Markov decision processes presented in Chap. 3. More generally, however, encourage reader to follow informal meaning & think of state as whatever information is available to agent about its environment. In effect, assume: state signal is produced by some preprocessing system that is nominally part of agent's environment. Do not address issues of constructing, changing, or learning state signal in this book (other than briefly in Sect. 17.3). Take this approach not because we consider state representation to be unimportant, but in order to focus fully on decision-making issues. I.e., our concern in this book is not with designing state signal, but with deciding what action to take as a function of whatever state signal is available.

  – RL dựa rất nhiều vào khái niệm trạng thái – là đầu vào cho chính sách & hàm giá trị, & vừa là đầu vào cho & đầu ra từ mô hình. 1 cách không chính thức, có thể coi trạng thái là 1 tín hiệu truyền đạt cho tác nhân 1 số cảm giác về "môi trường như thế nào" tại 1 thời điểm cụ thể. Định nghĩa chính thức về trạng thái khi chúng ta sử dụng ở đây được đưa ra bởi khuôn khổ các quy trình quyết định Markov được trình bày trong Chương 3. Tuy nhiên, nói chung hơn, hãy khuyến khích người đọc theo ý nghĩa không chính thức & coi trạng thái là bất kỳ thông tin nào có sẵn cho tác nhân về môi trường của nó. Trên thực tế, hãy giả sử: tín hiệu trạng thái được tạo ra bởi 1 số hệ thống tiền xử lý mà về danh nghĩa là 1 phần của môi trường của tác nhân. Không đề cập đến các vấn đề về xây dựng, thay đổi hoặc học tín hiệu trạng thái trong cuốn sách này (ngoại trừ phần tóm tắt trong Mục 17.3). Sử dụng cách tiếp cận này không phải vì chúng tôi coi biểu diễn trạng thái là không quan trọng, mà là để tập trung hoàn toàn vào các vấn đề ra quyết định. I.e., mối quan tâm của chúng tôi trong cuốn sách này không phải là thiết kế tín hiệu trạng thái, mà là quyết định hành động nào sẽ thực hiện như 1 hàm của bất kỳ tín hiệu trạng thái nào có sẵn.

  Most of RL methods we consider in this book are structured around estimating value functions, but not strictly necessary to do this to solve RL problems. E.g., solution methods e.g. genetic algorithms, genetic programming, simulated annealing, & other optimization methods never estimate value functions. These methods apply multiple static policies each interacting over an extended period of time with a separate instance of environment. Policies that obtain most reward, & random variations of them, are carried over to next generation of policies, & process repeats. Call these *evolutionary* methods because their operation is analogous to way biological evolution produces organisms with skilled behavior even if they do not learn during their individual lifetimes. If space of policies is sufficiently small, or can be structured so that good policies are common or easy to find – or if a lot of time is available for search – then evolutionary methods can be effective. In addition, evolutionary methods have advantages on problems on which learning agent cannot sense complete state of its environment.

  – Hầu hết các phương pháp RL mà chúng tôi xem xét trong cuốn sách này đều được cấu trúc xung quanh việc ước lượng các hàm giá trị, nhưng không nhất thiết phải làm điều này để giải quyết các bài toán RL. Ví dụ, các phương pháp giải như thuật toán di truyền, lập trình di truyền, ủ mô phỏng, & các phương pháp tối ưu hóa khác không bao giờ ước lượng các hàm giá trị. Các phương pháp này áp dụng nhiều chính sách tĩnh, mỗi chính sách tương tác trong 1 khoảng thời gian dài với 1 môi trường riêng biệt. Các chính sách đạt được nhiều phần thưởng nhất, & các biến thể ngẫu nhiên của chúng, được chuyển sang thế hệ chính sách tiếp theo, & lặp lại quy trình. Gọi chúng là phương pháp *tiến hóa* vì hoạt động của chúng tương tự như cách tiến hóa sinh học tạo ra các sinh vật có hành vi lành nghề ngay cả khi chúng không học trong suốt vòng đời của chúng. Nếu không gian chính sách đủ nhỏ, hoặc có thể được cấu trúc sao cho các chính sách tốt trở nên phổ biến hoặc dễ tìm thấy – hoặc nếu có nhiều thời gian để tìm kiếm – thì phương pháp tiến hóa có thể hiệu quả. Ngoài ra, phương pháp tiến hóa có lợi thế đối với các bài toán mà tác nhân học tập không thể cảm nhận được trạng thái hoàn chỉnh của môi trường.

  Our focus is on RL methods that learn while interacting with environment, which evolutionary methods do not do. Methods able to take advantage of details of individual behavioral interactions can be much more efficient than evolutionary methods in many cases. Evolutionary methods ignore much of useful structure of RL problem: they do not use fact: policy they are searching for is a function from states to actions; they do not notice which states an individual passes through during its lifetime, or which actions it selects. In some cases such information can be misleading (e.g., when states are misperceived), but more often it should enable more efficient search. Although evolution & learning share many features & naturally work together, do not consider evolutionary methods by themselves to be especially well suited to RL problems &, accordingly, we do not cover them in this book.

  – Trọng tâm của chúng tôi là các phương pháp RL học trong khi tương tác với môi trường, điều mà các phương pháp tiến hóa không làm được. Các phương pháp có thể tận dụng các chi tiết của tương tác hành vi cá nhân có thể hiệu quả hơn nhiều so với các phương pháp tiến hóa trong nhiều trường hợp. Các phương pháp tiến hóa bỏ qua nhiều cấu trúc hữu ích của vấn đề RL: chúng không sử dụng thực tế: chính sách mà chúng đang tìm kiếm là 1 hàm từ trạng thái đến hành động; chúng không nhận thấy trạng thái nào mà 1 cá thể trải qua trong suốt cuộc đời của nó hoặc hành động nào mà nó chọn. Trong 1 số trường hợp, thông tin như vậy có thể gây hiểu lầm (ví dụ: khi các trạng thái bị nhận thức sai), nhưng thông thường hơn, nó sẽ cho phép tìm kiếm hiệu quả hơn. Mặc dù tiến hóa & học tập chia sẻ nhiều đặc điểm & hoạt động tự nhiên cùng nhau, nhưng không coi các phương pháp tiến hóa tự thân là đặc biệt phù hợp với các vấn đề RL &, do đó, chúng tôi không đề cập đến chúng trong cuốn sách này.

- **1.5. An Extended Example: Tic-Tac-Toe.** To illustrate general idea of RL & contrast it with other approaches, next consider a single example in more detail. Consider familiar child's game of tic-tac-toe. 2 players take turns playing on a 3-by-3 board. 1 player plays Xs & the other Os until 1 player wins by placing 3 marks in a row, horizontally, vertically, or diagonally, as X player has in game shown to right. If board fills up with neither player getting 3 in a row, then game is a draw. Because a skilled player can play so as never to lose, assume: we are playing against an imperfect player, one whose play is sometimes incorrect & allows us to win. For moment, in fact, consider draws & losses to be equally bad for us. How might we construct a player that will find imperfections in its opponent's play & learn to maximize its chances of winning?

  – Để minh họa ý tưởng chung về RL & so sánh nó với các cách tiếp cận khác, tiếp theo hãy xem xét 1 ví dụ chi tiết hơn. Hãy xem xét trò chơi ô ăn quan quen thuộc của trẻ em. 2 người chơi lần lượt chơi trên 1 bàn cờ 3x3. 1 người chơi đánh X & người kia đánh O cho đến khi 1 người chơi thắng bằng cách đặt 3 điểm liên tiếp, theo chiều ngang, chiều dọc hoặc đường chéo, như người chơi X đã chỉ ra trong trò chơi ở bên phải. Nếu bàn cờ đầy mà không có người chơi nào đạt được 3 điểm liên tiếp, thì trò chơi hòa. Bởi vì 1 người chơi có kỹ năng có thể chơi để không bao giờ thua, hãy giả sử: chúng ta đang chơi với 1 người chơi không hoàn hảo, người chơi đôi khi chơi không chính xác & cho phép chúng ta thắng. Trên thực tế, hiện tại, hãy coi hòa & thua đều tệ như nhau đối với chúng ta. Làm thế nào chúng ta có thể xây dựng 1 người chơi sẽ tìm ra điểm không hoàn hảo trong cách chơi của đối thủ & học cách tối đa hóa cơ hội chiến thắng của mình?

  Although this is a simple problem, it cannot readily be solved in a satisfactory way through classical techniques. E.g., classical "minimax" solution from game theory is not correct here because it assumes a particular way of playing by opponent. E.g., a minimax player would never reach a game state from which it could lose, even if in fact it always won from that state because of incorrect play by opponent. Classical optimization methods for sequential decision problems, e.g. dynamic programming, can compute an optimal solution for any opponent, but require as input a complete specification of that opponent, including probabilities with which opponent makes each move in each board state. Assume: this information is not available a priori for this problem, as it is not for vast majority of problems of practical interest. On other hand, such information can be estimated from experience, in this case by playing many games against opponent. About the best one can do on this problem is 1st to learn a model of opponent's behavior, up to some level of confidence, & then apply dynamic programming to compute an optimal solution given approximate opponent model. In end, this is not that different from some of RL methods we examine later in this book.

  – Mặc dù đây là 1 bài toán đơn giản, nhưng nó không thể dễ dàng được giải quyết 1 cách thỏa đáng bằng các kỹ thuật cổ điển. Ví dụ, giải pháp "minimax" cổ điển từ lý thuyết trò chơi không đúng ở đây vì nó giả định 1 cách chơi cụ thể của đối thủ. Ví dụ, 1 người chơi minimax sẽ không bao giờ đạt đến trạng thái trò chơi mà họ có thể thua, ngay cả khi trên thực tế họ luôn thắng từ trạng thái đó do đối thủ chơi không đúng. Các phương pháp tối ưu hóa cổ điển cho các bài toán quyết định tuần tự, ví dụ như lập trình động, có thể tính toán 1 giải pháp tối ưu cho bất kỳ đối thủ nào, nhưng yêu cầu đầu vào là 1 thông số kỹ thuật đầy đủ về đối thủ đó, bao gồm xác suất đối thủ thực hiện mỗi nước đi trong mỗi trạng thái bàn cờ. Giả sử: thông tin này không có sẵn trước cho bài toán này, vì nó không có sẵn cho phần lớn các bài toán thực tế. Mặt khác, thông tin đó có thể được ước tính từ kinh nghiệm, trong trường hợp này là bằng cách chơi nhiều ván với đối thủ. Cách tốt nhất có thể làm cho bài toán này là đầu tiên tìm hiểu 1 mô hình hành vi của đối thủ, với 1 mức độ tin cậy nhất định, & sau đó áp dụng lập trình động để tính toán 1 giải pháp tối ưu dựa trên mô hình đối thủ gần đúng. Cuối cùng, điều này không khác mấy so với 1 số phương pháp RL mà chúng ta sẽ xem xét sau trong cuốn sách này.

  An evolutionary method applied to this problem would directly search space of possible policies for one with a high probability of winning against opponent. Here, a policy is a rule that tells player what move to make for every state of game – every possible configuration of Xs & Os on 3×3 board. For each policy considered, an estimate of its winning probability would be obtained by playing some number of games against opponent. This evaluation would then direct which policy or policies were considered next. A typical evolutionary method would hill-climb in policy space, successively generating & evaluating policies in an attempt to obtain incremental improvements. Or, perhaps, a genetic-style algorithm could be used that would maintain & evaluate a population of policies. Literally hundreds of different optimization methods could be applied.

  – 1 phương pháp tiến hóa được áp dụng cho vấn đề này sẽ trực tiếp tìm kiếm không gian các chính sách khả thi cho 1 chính sách có xác suất chiến thắng cao trước đối thủ. Ở đây, 1 chính sách là 1 quy tắc cho người chơi biết phải thực hiện nước đi nào cho mọi trạng thái của trò chơi – mọi cấu hình có thể có của Xs & O trên bàn cờ 3×3. Đối với mỗi chính sách được xem xét, ước tính xác suất chiến thắng của chính sách đó sẽ được thu được bằng cách chơi 1 số ván đấu nhất định với đối thủ. Đánh giá này sau đó sẽ chỉ ra chính sách hoặc các chính sách nào được xem xét tiếp theo. 1 phương pháp tiến hóa điển hình sẽ leo đồi trong không gian chính sách, liên tục tạo ra & đánh giá các chính sách nhằm cố gắng đạt được những cải tiến gia tăng. Hoặc, có lẽ, 1 thuật toán kiểu di truyền có thể được sử dụng để duy trì & đánh giá 1 quần thể các chính sách. Hàng trăm phương pháp tối ưu hóa khác nhau có thể được áp dụng.

  Here is how tic-tac-toe problem would be approached with a method making use of a value function. 1st would set up a table of numbers, one for each possible state of game. Each number will be latest estimate of probability of our winning from that state. Treat this estimate as state's *value*, & whole table is learned value function. State A is higher value than state B, or is considered "better" than state B, if current estimate of probability of our winning from A is higher than it is from B. Assuming we always play Xs, then for all states with 3 Xs in a row probability of winning is 1, because we have already won. Similarly, for all states with 3 Os in a row, or that are filled up, correct probability is 0, as we cannot win from them. Set initial values of all other states to 0.5, representing a guess that we have a 50% chance of winning.

  – Sau đây là cách tiếp cận bài toán ô ăn quan bằng phương pháp sử dụng hàm giá trị. Đầu tiên, hãy lập 1 bảng số, mỗi số cho 1 trạng thái có thể xảy ra của trò chơi. Mỗi số sẽ là ước tính mới nhất về xác suất chiến thắng của chúng ta từ trạng thái đó. Hãy coi ước tính này là *giá trị của nó* của trạng thái, & toàn bộ bảng là hàm giá trị đã học. Trạng thái A có giá trị cao hơn trạng thái B, hoặc được coi là "tốt hơn" trạng thái B, nếu ước tính hiện tại về xác suất chiến thắng của chúng

ta từ A cao hơn từ B. Giả sử chúng ta luôn chơi X, thì đối với tất cả các trạng thái có 3 X liên tiếp, xác suất chiến thắng là 1, vì chúng ta đã thắng. Tương tự, đối với tất cả các trạng thái có 3 O liên tiếp, hoặc đã được lấp đầy, xác suất đúng là 0, vì chúng ta không thể thắng từ chúng. Đặt giá trị ban đầu của tất cả các trạng thái khác thành 0,5, thể hiện dự đoán rằng chúng ta có 50% cơ hội chiến thắng.

Then play many games against opponent. To select our moves, examine states that would result from each of our possible moves (one for each blank space on board) & look up their current values in table. Most of time, we move *greedily*, selecting move that leads to state with greatest value, i.e., with highest estimated probability of winning. Occasionally, however, select randomly from among the other moves instead. These are called *exploratory* moves because they cause us to experience states that we might otherwise never see. A sequence of moves made & considered during a game can be diagrammed as in Fig. 1.1: A sequence of tic-tac-toe moves. Solid black lines represent moves taken during a game; dashed lines represent moves that we (our RL player) considered but did not make. * indicates move currently estimated to be the best. Our 2nd move was an exploratory move, i.e., it was taken even though another sibling move, the one leading to e^*, was ranked higher. Exploratory moves do not result in any learning, but each of our other moves does, causing updates are suggested by red arrows in which estimated values are moved up tree from later nodes to earlier nodes as detailed in text.

– Sau đó chơi nhiều ván đấu với đối thủ. Để chọn nước đi, hãy kiểm tra các trạng thái có thể xảy ra từ mỗi nước đi khả thi của chúng ta (một nước đi cho mỗi ô trống trên bàn cờ) & tra cứu giá trị hiện tại của chúng trong bảng. Hầu hết thời gian, chúng ta di chuyển *1 cách tham lam*, chọn nước đi dẫn đến trạng thái có giá trị lớn nhất, i.e., có xác suất chiến thắng ước tính cao nhất. Tuy nhiên, đôi khi, chúng ta chọn ngẫu nhiên từ các nước đi khác. Đây được gọi là nước đi *khám phá* vì chúng khiến chúng ta trải nghiệm các trạng thái mà nếu không thì chúng ta có thể sẽ không bao giờ thấy. 1 chuỗi các nước đi được thực hiện & được xem xét trong 1 ván cờ có thể được biểu diễn như trong Hình 1.1: 1 chuỗi các nước đi ô ăn quan. Các đường liền màu đen biểu diễn các nước đi được thực hiện trong 1 ván cờ; các đường đứt nét biểu diễn các nước đi mà chúng ta (người chơi thực tế của chúng ta) đã cân nhắc nhưng không thực hiện. * biểu thị nước đi hiện được ước tính là tốt nhất. Nước đi thứ 2 của chúng ta là 1 nước đi khám phá, i.e., nó được thực hiện mặc dù 1 nước đi khác cùng nhóm, nước đi dẫn đến e^*, được xếp hạng cao hơn. Các bước di chuyển khám phá không mang lại bất kỳ bài học nào, nhưng mỗi bước di chuyển khác của chúng ta đều mang lại, khiến các bản cập nhật được gợi ý bằng các mũi tên màu đỏ trong đó các giá trị ước tính được di chuyển lên cây từ các nút sau đến các nút trước như được trình bày chi tiết trong văn bản.

While we are playing, change values of states in which we find ourselves during game. Attempt to make them more accurate estimates of probabilities of winning. To do this, "back up" value of state after each greedy move to state before move, as suggested by arrows in Fig. 1.1. More precisely, current value of earlier state is updated to be closer to value of later state. This can be done by moving earlier state's value a fraction of way toward value of later state. If let S_t denote state before greedy move, & S_{t+1} state after that move, then update to estimated value of S_t, denoted $V(S_t)$, can be written as

$$V(S_t) \leftarrow V(S_t) + \alpha[V(S_{t+1}) - V(S_t)],$$

where α is a small positive fraction called *step-size parameter*, which influences rate of learning. This update rule is an example of a *temporal-difference* learning method, so called because its changes are based on a difference, $V(S_{t+1}) - V(S_t)$, between estimates at 2 successive times.

– Trong khi chơi, hãy thay đổi giá trị của các trạng thái mà chúng ta đang ở trong trò chơi. Cố gắng ước tính chính xác hơn xác suất chiến thắng. Để làm điều này, hãy "sao lưu" giá trị của trạng thái sau mỗi lần di chuyển tham lam đến trạng thái trước khi di chuyển, như được gợi ý bằng các mũi tên trong Hình 1.1. Chính xác hơn, giá trị hiện tại của trạng thái trước đó được cập nhật để gần hơn với giá trị của trạng thái sau đó. Điều này có thể được thực hiện bằng cách di chuyển giá trị của trạng thái trước đó 1 phần về phía giá trị của trạng thái sau đó. Nếu cho S_t biểu thị trạng thái trước khi di chuyển tham lam, & S_{t+1} trạng thái sau khi di chuyển đó, thì cập nhật thành giá trị ước tính của S_t, ký hiệu là $V(S_t)$, có thể được viết như sau

$$V(S_t) \leftarrow V(S_t) + \alpha[V(S_{t+1}) - V(S_t)],$$

trong đó α là 1 phân số dương nhỏ được gọi là *tham số kích thước bước*, ảnh hưởng đến tốc độ học. Quy tắc cập nhật này là 1 ví dụ về phương pháp học *chênh lệch thời gian*, được gọi như vậy vì các thay đổi của nó dựa trên chênh lệch $V(S_{t+1}) - V(S_t)$, giữa các ước tính tại 2 thời điểm liên tiếp.

Method described above performs quite well on this task. E.g., if step-size parameter is reduced properly over time, then this method converges, for any fixed opponent, to true probabilities of winning from each state given optimal play by our player. Furthermore, moves then taken (except on exploratory moves) are in fact optimal moves against this (imperfect) opponent. I.e., method converges to an optimal policy for playing game against this opponent. If step-size parameter is not reduced all way to 0 over time, then this player also plays well against opponents that slowly change their way of playing.

– Phương pháp được mô tả ở trên thực hiện khá tốt nhiệm vụ này. Ví dụ, nếu tham số bước nhảy được giảm dần theo thời gian, thì phương pháp này sẽ hội tụ, đối với bất kỳ đối thủ cố định nào, về xác suất chiến thắng thực sự từ mỗi trạng thái với lối chơi tối ưu của người chơi. Hơn nữa, các nước đi được thực hiện sau đó (trừ các nước đi thăm dò) thực chất là các nước đi tối ưu khi đối đầu với đối thủ (không hoàn hảo) này. I.e., phương pháp này hội tụ về 1 chiến lược tối ưu để chơi với đối thủ này. Nếu tham số bước nhảy không giảm hoàn toàn về 0 theo thời gian, thì người chơi này cũng chơi tốt với những đối thủ dần thay đổi cách chơi.

This example illustrates differences between evolutionary methods & methods that learn value functions. To evaluate a policy, an evolutionary method holds policy fixed & plays many games against opponent or simulates many games using a model of opponent. Frequency of wins gives an unbiased estimate of probability of winning with that policy, & can be used to direct next policy selection. But each policy change is made only after many games, & only final outcome of each

game is used: what happens *during* the games is ignored. E.g., if player wins, then *all* of its behavior in game is given credit, independently of how specific moves might have been critical to win. Credit is even given to moves that never occurred! Value function methods, in contrast, allow individual states to be evaluated. In end, evolutionary & value function methods both search space of policies, but learning a value function takes advantage of information available during course of play.

– Ví dụ này minh họa sự khác biệt giữa các phương pháp tiến hóa & các phương pháp học hàm giá trị. Để đánh giá 1 chính sách, 1 phương pháp tiến hóa giữ chính sách cố định & chơi nhiều ván đấu với đối thủ hoặc mô phỏng nhiều ván đấu bằng cách sử dụng mô hình của đối thủ. Tần suất chiến thắng đưa ra ước tính khách quan về xác suất chiến thắng với chính sách đó, & có thể được sử dụng để chỉ đạo lựa chọn chính sách tiếp theo. Nhưng mỗi lần thay đổi chính sách chỉ được thực hiện sau nhiều ván đấu, & chỉ kết quả cuối cùng của mỗi ván đấu được sử dụng: những gì xảy ra *trong* các ván đấu bị bỏ qua. Ví dụ: nếu người chơi thắng, thì *tất cả* hành vi của người chơi trong ván đấu đều được ghi nhận, bất kể các nước đi cụ thể có thể quan trọng như thế nào để giành chiến thắng. Thậm chí còn ghi nhận cả những nước đi chưa bao giờ xảy ra! Ngược lại, các phương pháp hàm giá trị cho phép đánh giá các trạng thái riêng lẻ. Cuối cùng, các phương pháp hàm giá trị & tiến hóa đều tìm kiếm không gian của các chính sách, nhưng việc học 1 hàm giá trị sẽ tận dụng thông tin có sẵn trong quá trình chơi.

This simple example illustrates some of key features of RL methods. 1st, there is emphasis on learning while interacting with an environment in this case with an opponent player. 2nd, there is a clear goal, & correct behavior requires planning or foresight that takes into account delayed effects of one's choices. E.g., simple RL player would learn to set up multi-move traps for a shortsighted opponent. It is a striking feature of RL solution that it can achieve effects of planning & lookahead without using a model of opponent & without conducting an explicit search over possible sequences of future states & actions.

– Ví dụ đơn giản này minh họa 1 số đặc điểm chính của phương pháp RL. Thứ nhất, phương pháp này nhấn mạnh vào việc học hỏi trong khi tương tác với môi trường, trong trường hợp này là với đối thủ. Thứ hai, có 1 mục tiêu rõ ràng, & hành vi đúng đắn đòi hỏi sự lập kế hoạch hoặc tầm nhìn xa, có tính đến các tác động chậm trễ của lựa chọn. Ví dụ, 1 người chơi RL đơn giản sẽ học cách đặt bẫy nhiều nước đi cho 1 đối thủ thiển cận. 1 đặc điểm nổi bật của giải pháp RL là nó có thể đạt được hiệu quả của việc lập kế hoạch & nhìn trước mà không cần sử dụng mô hình của đối thủ & không cần thực hiện tìm kiếm rõ ràng trên các chuỗi trạng thái & hành động có thể xảy ra trong tương lai.

While this example illustrates some of key features of RL, so simple: it might give impression that RL is more limited than it really is. Although tic-tac-toe is a 2-person game, RL also applies in case in which there is no external adversary, i.e., in case of a "game against nature". RL also is not restricted to problems in which behavior breaks down into separate episodes, like separate games of tic-tac-toe, with reward only at end of each episode. It is just as applicable when behavior continues indefinitely & when rewards of various magnitudes can be received at any time. RL is also applicable to problems that do not even break down into discrete time steps like plays of tic-tac-toe. General principles apply to continuous-time problems as well, although theory gets more complicated & omit it from this introductory treatment.

– Trong khi ví dụ này minh họa 1 số tính năng chính của RL, rất đơn giản: nó có thể tạo ấn tượng rằng RL bị hạn chế hơn thực tế. Mặc dù tic-tac-toe là trò chơi 2 người, RL cũng áp dụng trong trường hợp không có đối thủ bên ngoài, i.e., trong trường hợp "trò chơi chống lại tự nhiên". RL cũng không bị giới hạn ở các vấn đề trong đó hành vi bị chia thành các tập riêng biệt, như các ván tic-tac-toe riêng biệt, với phần thưởng chỉ ở cuối mỗi tập. Nó cũng áp dụng khi hành vi tiếp tục vô thời hạn & khi phần thưởng có nhiều mức độ khác nhau có thể được nhận bất cứ lúc nào. RL cũng áp dụng cho các vấn đề thậm chí không bị chia thành các bước thời gian rời rạc như các ván tic-tac-toe. Các nguyên tắc chung cũng áp dụng cho các vấn đề thời gian liên tục, mặc dù lý thuyết trở nên phức tạp hơn & bỏ qua nó khỏi cách xử lý giới thiệu này.

Tic-tact-toe has a relatively small, finite state set, whereas RL can be used when state set is very large, or even infinite. E.g., Gerry Tesauro (1992, 1995) combined algorithm described above with an ANN to learn to play backgammon, which has $\approx 10^{20}$ states. With this many states, impossible ever to experience more than a small fraction of them. Tesauro's program learned to play far better than any previous program & eventually better than world's best human players (see Sect. 16.1). ANN provides program with ability to generalize from its experience, so that in new states it selects moves based on information saved from similar states faced in past, as determined by network. How well a RL system can work in problems with such large state sets is intimately tied to how appropriately it can generalize from past experiences. It is in this role: we have greatest need for supervised learning methods within RL. ANNs & DL (Sect. 9.7) are not the only, or necessary the best, way to do this.

– Tic-tact-toe có tập trạng thái hữu hạn tương đối nhỏ, trong khi RL có thể được sử dụng khi tập trạng thái rất lớn hoặc thậm chí vô hạn. Ví dụ, Gerry Tesauro (1992, 1995) đã kết hợp thuật toán được mô tả ở trên với ANN để học chơi cờ thỏ cáo, có $\approx 10^{20}$ trạng thái. Với nhiều trạng thái như vậy, không bao giờ có thể trải nghiệm nhiều hơn 1 phần nhỏ trong số chúng. Chương trình của Tesauro đã học cách chơi tốt hơn nhiều so với bất kỳ chương trình nào trước đó & cuối cùng là tốt hơn những người chơi giỏi nhất thế giới (xem Mục 16.1). ANN cung cấp cho chương trình khả năng khái quát hóa từ kinh nghiệm của chính nó, do đó, trong các trạng thái mới, nó sẽ chọn các nước đi dựa trên thông tin được lưu từ các trạng thái tương tự đã gặp phải trong quá khứ, do mạng xác định. Mức độ hoạt động tốt của 1 hệ thống RL trong các bài toán có tập trạng thái lớn như vậy gắn liền mật thiết với mức độ phù hợp mà nó có thể khái quát hóa từ các kinh nghiệm trong quá khứ. Trong vai trò này: chúng ta có nhu cầu lớn nhất về các phương pháp học có giám sát trong RL. ANN & DL (Mục 9.7) không phải là cách duy nhất hoặc tốt nhất để thực hiện điều này.

In this tic-tac-toe example, learning started with no prior knowledge beyond rules of game, but RL by no means entails a tabula rasa view of learning & intelligence. On contrary, prior information can be incorporated into RL in a variety of ways that can be critical for efficient learning (e.g., see Sects. 9.5, 17.4, & 13.1). Also have access to true state in tic-tac-toe example, whereas RL can also be applied when part of state is hidden, or when different states appear to learner to be same.

– Trong ví dụ cờ caro này, việc học bắt đầu mà không có kiến thức nền nào ngoài luật chơi, nhưng RL không hề hàm ý 1 cái nhìn phiến diện về học tập & trí tuệ. Ngược lại, thông tin nền tảng có thể được tích hợp vào RL theo nhiều cách khác nhau, rất quan trọng cho việc học tập hiệu quả (ví dụ: xem Mục 9.5, 17.4, & 13.1). Cũng có thể truy cập vào trạng thái thực trong ví dụ cờ caro, trong khi RL cũng có thể được áp dụng khi 1 phần trạng thái bị ẩn, hoặc khi các trạng thái khác nhau xuất hiện giống nhau đối với người học.

Finally, tic-tac-toe player was able to look ahead & know states that would result from each of its possible moves. To do this, it had to have a model of game that allowed it to foresee how its environment would change in response to moves that it might never make. Many problems are like this, but in others even a short-term model of effects of actions is lacking. RL can be applied in either case. A model is not required, but models can easily be used if they are available or can be learned (Chap. 8).

– Cuối cùng, người chơi cờ caro đã có thể nhìn trước & biết được các trạng thái sẽ xảy ra từ mỗi nước đi khả dĩ của mình. Để làm được điều này, người chơi phải có 1 mô hình trò chơi cho phép dự đoán môi trường xung quanh sẽ thay đổi như thế nào để phản ứng với những nước đi mà có thể họ sẽ không bao giờ thực hiện. Nhiều bài toán tương tự như vậy, nhưng ở 1 số bài toán khác, ngay cả 1 mô hình ngắn hạn về tác động của các hành động cũng còn thiếu. RL có thể được áp dụng trong cả 2 trường hợp. Không bắt buộc phải có mô hình, nhưng có thể dễ dàng sử dụng các mô hình nếu có sẵn hoặc có thể học được (Chương 8).

On other hand, there are RL methods that do not need any kind of environment model at all. Model-free systems cannot even think about how their environments will change in response to a single action. Tic-tac-toe player is model-free in this sense w.r.t. its opponent: it has no model of its opponent of any kind. Because models have to be reasonably accurate to be useful, model-free methods can have advantages over more complex methods when real bottleneck in solving a problem is difficulty of constructing a sufficiently accurate environment model. Model-free methods are also important building blocks for model-based methods. In this book, devote several chaps to model-free methods before discuss how they can be used as components of more complex model-based methods.

– Mặt khác, có những phương pháp RL không cần bất kỳ loại mô hình môi trường nào cả. Các hệ thống không mô hình thậm chí không thể nghĩ về cách môi trường của chúng sẽ thay đổi để đáp ứng với 1 hành động duy nhất. Người chơi Tic-tac-toe là không mô hình theo nghĩa này đối với đối thủ của nó: nó không có mô hình của đối thủ dưới bất kỳ hình thức nào. Vì các mô hình phải có độ chính xác hợp lý để hữu ích, các phương pháp không mô hình có thể có lợi thế hơn các phương pháp phức tạp hơn khi nút thắt thực sự trong việc giải quyết vấn đề là khó khăn trong việc xây dựng 1 mô hình môi trường đủ chính xác. Các phương pháp không mô hình cũng là những khối xây dựng quan trọng cho các phương pháp dựa trên mô hình. Trong cuốn sách này, hãy dành 1 số chương về các phương pháp không mô hình trước khi thảo luận về cách chúng có thể được sử dụng như các thành phần của các phương pháp dựa trên mô hình phức tạp hơn.

RL can be used at both high & low levels in a system. Although tic-tac-toe player learned only about basic moves of game, nothing prevents RL from working at higher levels where each of "actions" may itself be application of a possibly elaborate problem-solving method. In hierarchical learning systems, RL can work simultaneously on several levels.

– Học tăng cường có thể được sử dụng ở cả cấp độ cao & thấp trong 1 hệ thống. Mặc dù người chơi cờ caro chỉ học được những nước đi cơ bản của trò chơi, nhưng không có gì ngăn cản học tăng cường hoạt động ở các cấp độ cao hơn, nơi mỗi "hành động" có thể tự nó là ứng dụng của 1 phương pháp giải quyết vấn đề phức tạp. Trong các hệ thống học phân cấp, học tăng cường có thể hoạt động đồng thời ở nhiều cấp độ.

Problem 1 (Self-Play)**.** *Suppose, instead of playing against a random opponent, RL algorithm described above played against itself, with both sides learning. What do you think would happen in this case? Would it learn a different policy for selecting moves?*

– Giả sử, thay vì chơi với 1 đối thủ ngẫu nhiên, thuật toán RL được mô tả ở trên chơi với chính nó, cả 2 bên đều học. Bạn nghĩ điều gì sẽ xảy ra trong trường hợp này? Liệu nó có học được 1 chính sách khác để lựa chọn nước đi không?

Problem 2 (Symmetries)**.** *Many tic-tac-toe positions appear different but are really same because of symmetries. How might we amend learning process described above to take advantage of this? In what ways would this change improve learning process? Now think again. Suppose opponent did not take advantage of symmetries. In that case, should we? Is it true, then, that symmetrically equivalent positions should necessarily have same value?*

– Nhiều vị trí cờ caro trông có vẻ khác nhau nhưng thực chất lại giống nhau do tính đối xứng. Làm thế nào chúng ta có thể sửa đổi quy trình học tập được mô tả ở trên để tận dụng điều này? Sự thay đổi này sẽ cải thiện quy trình học tập theo những cách nào? Giờ hãy nghĩ lại. Giả sử đối thủ không tận dụng tính đối xứng. Trong trường hợp đó, chúng ta có nên làm vậy không? Vậy thì, liệu các vị trí tương đương đối xứng nhất thiết phải có cùng giá trị không?

Problem 3 (Greedy Play)**.** *Suppose RL player was greedy, i.e., it always played the move that brought it to position that it rated best. Might it learn to play better, or worse, than a nongreedy player? What problems might occur?*

– Giả sử người chơi RL tham lam, i.e., luôn đi nước đi đưa mình đến vị trí được đánh giá cao nhất. Liệu nó có thể học cách chơi tốt hơn, hay tệ hơn, 1 người chơi không tham lam? Những vấn đề nào có thể xảy ra?

Problem 4 (Learning from Exploration)**.** *Suppose learning updates occurred after* all *moves, including exploratory moves. If step-size parameter is appropriately reduced over time (but not tendency to explore), then state values would converge to a different set of probabilities. What (conceptually) are 2 sets of probabilities computed when we do, & when we do not, learn from exploratory moves? Assuming: we do continue to make exploratory moves, which set of probabilities might be better to learn? Which would result in more wins?*

– Giả sử các cập nhật học tập xảy ra sau tất cả *bước di chuyển, bao gồm cả các bước di chuyển thăm dò. Nếu tham số bước di chuyển được giảm phù hợp theo thời gian (nhưng không phải xu hướng khám phá), thì các giá trị trạng thái sẽ hội tụ về 1*

tập hợp xác suất khác. Vậy (về mặt khái niệm), 2 tập hợp xác suất nào được tính toán khi chúng ta thực hiện, & khi chúng ta không thực hiện, học từ các bước di chuyển thăm dò? Giả sử: chúng ta tiếp tục thực hiện các bước di chuyển thăm dò, tập hợp xác suất nào có thể học tốt hơn? Tập hợp nào sẽ mang lại nhiều chiến thắng hơn?

Problem 5 (Other Improvements)**.** *Can you think of other ways to improve RL player? Can you think of any better way to solve tic-tac-toe problem as posed?*

– Bạn có thể nghĩ ra cách nào khác để cải thiện trình chơi RL không? Bạn có thể nghĩ ra cách nào tốt hơn để giải bài toán tic-tac-toe như đã nêu không?

- **1.6. Summary.** RL is a computational approach to understanding & automating goal-directed learning & decision making. It is distinguished from other computational approaches by its emphasis on learning by an agent from direct interaction with its environment, without requiring exemplary supervision or complete models of environment. In our opinion, RL is 1st field to seriously address computational issues that arise when learning from interaction with an environment in order to achieve long-term goals.

  – RL là 1 phương pháp tính toán để hiểu & tự động hóa việc học tập hướng mục tiêu & ra quyết định. Phương pháp này khác biệt với các phương pháp tính toán khác ở chỗ nhấn mạnh vào việc học tập của tác nhân thông qua tương tác trực tiếp với môi trường, mà không cần giám sát mẫu mực hay mô hình môi trường hoàn chỉnh. Theo chúng tôi, RL là lĩnh vực đầu tiên nghiêm túc giải quyết các vấn đề tính toán phát sinh khi học tập từ tương tác với môi trường nhằm đạt được các mục tiêu dài hạn.

  RL uses formal framework of Markov decision processes to define interaction between a learning agent & its environment in terms of states, actions, & rewards. This framework is intended to be a simple way of representing essential features of AI problem. These features include a sense of cause & effect, a sense of uncertainty & nondeterminism, & existence of explicit goals.

  – RL sử dụng khuôn khổ chính thức của các quy trình quyết định Markov để xác định tương tác giữa tác nhân học tập & môi trường của nó dưới dạng trạng thái, hành động, & phần thưởng. Khuôn khổ này được thiết kế để biểu diễn các đặc điểm thiết yếu của bài toán AI 1 cách đơn giản. Các đặc điểm này bao gồm ý thức về nguyên nhân & kết quả, ý thức về sự không chắc chắn & tính không xác định, & sự tồn tại của các mục tiêu rõ ràng.

  Concepts of value & value function are key to most of RL methods considered in this book. Take position that value functions are important for efficient search in space of policies. Use of value functions distinguishes RL methods from evolutionary methods that search directly in policy space guided by evaluations of entire policies.

  – Khái niệm giá trị & hàm giá trị là chìa khóa cho hầu hết các phương pháp RL được đề cập trong cuốn sách này. Chúng tôi cho rằng hàm giá trị rất quan trọng để tìm kiếm hiệu quả trong không gian chính sách. Việc sử dụng hàm giá trị phân biệt các phương pháp RL với các phương pháp tiến hóa tìm kiếm trực tiếp trong không gian chính sách được hướng dẫn bởi việc đánh giá toàn bộ chính sách.

- **1.7. Early History of RL.**

PART I: TABULAR SOLUTION METHODS. In this part of book, describe almost all core ideas of RL algorithms in their simplest forms: that in which state & action spaces are small enough for approximate value functions to be represented as arrays, or *tables*. In this case, methods can often find exact solutions, i.e., they can often find exactly optimal value function & optimal policy. This contrasts with approximate methods described in next part of book, which only find approximate solutions, but which in return can be applied effectively to much larger problems.

– Trong phần này của sách, chúng tôi sẽ mô tả hầu hết các ý tưởng cốt lõi của thuật toán RL ở dạng đơn giản nhất: trong đó không gian trạng thái & hành động đủ nhỏ để các hàm giá trị xấp xỉ được biểu diễn dưới dạng mảng hoặc *table*. Trong trường hợp này, các phương pháp thường có thể tìm ra giải pháp chính xác, i.e., chúng thường có thể tìm ra chính xác hàm giá trị tối ưu & chính sách tối ưu. Điều này trái ngược với các phương pháp xấp xỉ được mô tả trong phần tiếp theo của sách, vốn chỉ tìm ra giải pháp xấp xỉ, nhưng đổi lại có thể được áp dụng hiệu quả cho các bài toán lớn hơn nhiều.

1st chap of this part of book describes solution methods for special case of RL problem in which there is only a single state, called *bandit problems*. 2nd chap describes general problem formulation that we treat throughout rest of book – finite Markov decision processes – & its main ideas including Bellman equations & value functions.

– Chương đầu tiên của phần này mô tả các phương pháp giải cho trường hợp đặc biệt của bài toán RL trong đó chỉ có 1 trạng thái duy nhất, được gọi là *bandit problems*. Chương thứ 2 mô tả cách xây dựng bài toán chung mà chúng tôi xử lý trong suốt phần còn lại của cuốn sách – quy trình quyết định Markov hữu hạn – & các ý tưởng chính bao gồm phương trình Bellman & hàm giá trị.

Next 3 chaps describe 3 fundamental classes of methods for solving finite Markov decision problems: dynamic programming, Monte Carlo methods, & temporal-difference learning. Each class of methods has its strengths & weaknesses. Dynamic programming methods are well developed mathematically, but require a complete & accurate model of environment. Monte Carlo methods don't require a model & are conceptually simple, but are not well suited for step-by-step incremental computation. Finally, temporal-difference methods require no model & are fully incremental, but are more complex to analyze. Methods also differ in several ways w.r.t. their efficiency & speed of convergence.

– 3 chương tiếp theo sẽ mô tả 3 lớp phương pháp cơ bản để giải quyết các bài toán quyết định Markov hữu hạn: quy hoạch động, phương pháp Monte Carlo, & học sai phân thời gian. Mỗi lớp phương pháp đều có những ưu điểm & nhược điểm riêng. Các phương pháp quy hoạch động được phát triển tốt về mặt toán học, nhưng đòi hỏi 1 mô hình môi trường hoàn chỉnh &

chính xác. Các phương pháp Monte Carlo không yêu cầu mô hình & đơn giản về mặt khái niệm, nhưng không phù hợp cho tính toán gia tăng từng bước. Cuối cùng, các phương pháp sai phân thời gian không yêu cầu mô hình & hoàn toàn gia tăng, nhưng phức tạp hơn để phân tích. Các phương pháp cũng khác nhau về hiệu quả & tốc độ hội tụ.

Remaining 2 chaps describe how these 3 classes of methods can be combined to obtain best features of each of them. In 1 chap, describe how strengths of Monte Carlo methods can be combined with strengths of temporal-difference methods via multi-step bootstrapping methods. In final chap of this part of book, show how temporal-difference learning methods can be combined with model learning & planning methods (e.g. dynamic programming) for a complete & unified solution to tabular RL problem.

– Hai chương còn lại mô tả cách kết hợp 3 lớp phương pháp này để thu được các đặc điểm tốt nhất của từng lớp. Trong chương 1, mô tả cách kết hợp điểm mạnh của phương pháp Monte Carlo với điểm mạnh của phương pháp sai phân thời gian thông qua phương pháp khởi động nhiều bước. Trong chương cuối cùng của phần này, trình bày cách kết hợp các phương pháp học sai phân thời gian với phương pháp học mô hình & phương pháp lập kế hoạch (ví dụ: quy hoạch động) để có được giải pháp & thống nhất hoàn chỉnh cho bài toán RL dạng bảng.

- **2. Multi-armed Bandits.** Most important feature distinguishing RL from other types of learning: RL uses training information that *evaluates* actions taken rather than *instructs* by giving correct actions. This is what creates need for active exploration, for an explicit search for good behavior. Purely evaluative feedback indicates how good action taken was, but not whether it was best or worst action possible. Purely instructive feedback, on other hand, indicates correct action to take, independently of action actually taken. This kind of feedback is basis of supervised learning, which includes large parts of pattern classification, ANNs, & system identification. In their pure forms, these 2 kinds of feedback are quite distinct: evaluative feedback depends entirely on action taken, whereas instructive feedback is independent of action taken.

  –Máy đánh bạc nhiều tay. Đặc điểm quan trọng nhất phân biệt RL với các loại hình học tập khác: RL sử dụng thông tin đào tạo để *đánh giá* các hành động được thực hiện thay vì *hướng dẫn* bằng cách đưa ra các hành động đúng. Đây là điều tạo ra nhu cầu khám phá tích cực, để tìm kiếm rõ ràng hành vi tốt. Phản hồi đánh giá thuần túy chỉ ra hành động tốt được thực hiện như thế nào, nhưng không phải là hành động tốt nhất hay tệ nhất có thể. Mặt khác, phản hồi hướng dẫn thuần túy chỉ ra hành động đúng cần thực hiện, độc lập với hành động thực sự được thực hiện. Loại phản hồi này là cơ sở của học tập có giám sát, bao gồm phần lớn phân loại mẫu, ANN & nhận dạng hệ thống. Ở dạng thuần túy, 2 loại phản hồi này khá khác biệt: phản hồi đánh giá hoàn toàn phụ thuộc vào hành động được thực hiện, trong khi phản hồi hướng dẫn độc lập với hành động được thực hiện.

  In this chap, study evaluative aspect of RL in a simplified setting, one that does not involve learning to act in more than 1 situation. This *nonassociative* setting is the one in which most prior work involving evaluative feedback has been done, & it avoids much of complexity of full RL problem. Studying this case enables us to see most clearly how evaluative feedback differs from, & yet can be combined with, instructive feedback.

  – Trong chương này, chúng ta sẽ nghiên cứu khía cạnh đánh giá của RL trong 1 bối cảnh đơn giản, không liên quan đến việc học cách hành động trong nhiều hơn 1 tình huống. Bối cảnh *nonassociative* này là bối cảnh mà hầu hết các nghiên cứu trước đây liên quan đến phản hồi đánh giá đã được thực hiện, & nó tránh được phần lớn sự phức tạp của bài toán RL đầy đủ. Nghiên cứu trường hợp này cho phép chúng ta thấy rõ nhất sự khác biệt giữa phản hồi đánh giá & & nhưng có thể kết hợp với phản hồi hướng dẫn.

  Particular nonassociative, evaluative feedback problem that we explore is a simple version of k-armed bandit problem. Use this problem to introduce a number of basic learning methods which we extend in later chaps to apply to full RL problem. At end of this chap, take a step closer to full RL problem by discussing what happens when bandit problem becomes associative, i.e., when best action depends on situation.

  – Bài toán phản hồi đánh giá, phi liên kết cụ thể mà chúng ta đang tìm hiểu là 1 phiên bản đơn giản của bài toán máy đánh bạc k. Sử dụng bài toán này để giới thiệu 1 số phương pháp học cơ bản mà chúng tôi sẽ mở rộng trong các chương sau để áp dụng cho bài toán thực tế ảo đầy đủ. Cuối chương này, hãy tiến gần hơn 1 bước đến bài toán thực tế ảo đầy đủ bằng cách thảo luận về điều gì xảy ra khi bài toán máy đánh bạc trở thành bài toán kết hợp, i.e., khi hành động tốt nhất phụ thuộc vào tình huống.

  - **2.1. A k-armed Bandit Problem.** Consider following learning problem. You are faced repeatedly with a choice among k different options, or actions. After each choice you receive a numerical reward chosen from a stationary probability distribution that depends on action you selected. Your objective: maximize expected total reward over some time period, e.g., > 1000 action selections, or *time steps*.

    – Bài toán Cướp có vũ trang k. Hãy xem xét bài toán học sau. Bạn liên tục phải đối mặt với việc lựa chọn giữa k phương án, hay hành động, khác nhau. Sau mỗi lựa chọn, bạn nhận được 1 phần thưởng số được chọn từ phân phối xác suất dừng phụ thuộc vào hành động bạn đã chọn. Mục tiêu của bạn: tối đa hóa tổng phần thưởng kỳ vọng trong 1 khoảng thời gian nhất định, ví dụ: > 1000 lựa chọn hành động, hay *bước thời gian*.

    This is original form of *k-armed bandit problem*, so named by analogy to a slot machine, or "1-armed bandit", except: it has k levers instead of 1. Each action selection is like a play of 1 of slot machine's levers, & rewards are payoffs for hitting jackpot. Through repeated action selections you are to maximize your winnings by concentrating your actions on best levers. Another analogy is that of a doctor choosing between experimental treatments for a series of seriously ill patients. Each action is selection of a treatment, & each reward is survival or well-being of patient. Today term "bandit problem" is sometimes used for a generalization of problem described above, but in this book we use it to refer just to this simple case.

– Đây là dạng gốc của *k-armed bandit problem*, được đặt tên tương tự như máy đánh bạc, hay "máy đánh bạc 1 tay", ngoại trừ: nó có k đòn bẩy thay vì 1. Mỗi lựa chọn hành động giống như 1 lần chơi 1 trong những đòn bẩy của máy đánh bạc, & phần thưởng là khoản tiền thưởng khi trúng giải độc đắc. Thông qua các lựa chọn hành động lặp lại, bạn phải tối đa hóa tiền thắng của mình bằng cách tập trung hành động của mình vào những đòn bẩy tốt nhất. 1 phép tương tự khác là 1 bác sĩ lựa chọn giữa các phương pháp điều trị thử nghiệm cho 1 loạt bệnh nhân bị bệnh nặng. Mỗi hành động là lựa chọn 1 phương pháp điều trị, & mỗi phần thưởng là sự sống sót hoặc sức khỏe của bệnh nhân. Ngày nay, thuật ngữ "bài toán máy đánh bạc" đôi khi được sử dụng để khái quát hóa vấn đề được mô tả ở trên, nhưng trong cuốn sách này, chúng tôi sử dụng nó để chỉ trường hợp đơn giản này.

In our k-armed bandit problem, each of k actions has an expected or mean reward given that that action is selected; call this *value* of that action. Denote action selected on time step t as A_t, & corresponding reward as R_t. Value then of an arbitrary action a, denoted $q_*(a)$, is expected reward given that a is selected:

$$q_*(a) \coloneqq \mathbb{E}[R_t | A_t = a].$$

If you knew value of each action, then it would be trivial to solve k-armed bandit problem: you would always select action with highest value. Assume: you do not know action values with certainty, although you may have estimates. Denote estimated value of action a at time step t as $Q_t(a)$. Would like $Q_t(a)$ to be close to $q_*(a)$.

– Trong bài toán máy đánh bạc k của chúng ta, mỗi hành động trong số k hành động đều có phần thưởng kỳ vọng hoặc phần thưởng trung bình nếu hành động đó được chọn; hãy gọi *giá trị* của hành động đó. Ký hiệu hành động được chọn tại bước thời gian t là A_t, & phần thưởng tương ứng là R_t. Khi đó, giá trị của 1 hành động a bất kỳ, ký hiệu là $q_*(a)$, là phần thưởng kỳ vọng nếu a được chọn:

$$q_*(a) \coloneqq \mathbb{E}[R_t | A_t = a].$$

Nếu bạn biết giá trị của từng hành động, thì việc giải bài toán máy đánh bạc k sẽ rất đơn giản: bạn sẽ luôn chọn hành động có giá trị cao nhất. Giả sử: bạn không biết chắc chắn giá trị của hành động, mặc dù bạn có thể có ước tính. Ký hiệu giá trị ước tính của hành động a tại bước thời gian t là $Q_t(a)$. Muốn $Q_t(a)$ gần với $q_*(a)$.

If maintain estimates of action values, then at any time step there is at least 1 action whose estimated value is greatest. Call these *greedy* actions. When select 1 of these actions, say: you are *exploring* your current knowledge of values of actions. If instead you select 1 of nongreedy actions, then we say you are *exploring*, because this enables you to improve your estimate of nongreedy action's value. Exploitation is right thing to do to maximize expected reward on 1 step, but exploration may produce greater total reward in long run. E.g., suppose a greedy action's value is known with certainty, while several other actions are estimated to be nearly as good but with substantial uncertainty. Uncertainty is s.t. at least 1 of these other actions probably is actually better than greedy actions, but you don't know which one. If you have many times steps ahead on which to make action selections, then it may be better to explore nongreedy actions & discover which of them are better than greedy action. Reward is lower in short run, during exploration, but higher in long run because after you have discovered better actions, you can exploit them many times. Because not possible both to explore & exploit with any single action selection, one often refers to "conflict" between exploration & exploitation.

– Nếu duy trì ước tính giá trị hành động, thì tại bất kỳ bước thời gian nào cũng có ít nhất 1 hành động có giá trị ước tính lớn nhất. Gọi những hành động này là *tham lam*. Khi chọn 1 trong những hành động này, hãy nói: bạn đang *khám phá* kiến thức hiện tại của mình về giá trị của các hành động. Thay vào đó, nếu bạn chọn 1 trong những hành động không tham lam, thì chúng ta nói bạn đang *khám phá*, vì điều này cho phép bạn cải thiện ước tính của mình về giá trị của hành động không tham lam. Khai thác là điều đúng đắn cần làm để tối đa hóa phần thưởng mong đợi ở 1 bước, nhưng khám phá có thể tạo ra tổng phần thưởng lớn hơn về lâu dài. Ví dụ: giả sử giá trị của 1 hành động tham lam được biết chắc chắn, trong khi 1 số hành động khác được ước tính là gần tốt nhưng có sự không chắc chắn đáng kể. Sự không chắc chắn là s.t. ít nhất 1 trong những hành động khác này có thể thực sự tốt hơn hành động tham lam, nhưng bạn không biết hành động nào. Nếu bạn có nhiều bước gấp nhiều lần để đưa ra lựa chọn hành động, thì tốt hơn là nên khám phá các hành động không tham lam & khám phá xem hành động nào trong số chúng tốt hơn hành động tham lam. Phần thưởng thấp hơn trong ngắn hạn, trong quá trình khám phá, nhưng cao hơn về lâu dài vì sau khi bạn khám phá ra những hành động tốt hơn, bạn có thể khai thác chúng nhiều lần. Vì không thể vừa khám phá & khai thác chỉ với 1 lựa chọn hành động duy nhất, nên người ta thường nói đến "xung đột" giữa khám phá & khai thác.

In any specific case, whether better to explore or exploit depends in a complex way on precise values of estimates, uncertainties, & number of remaining steps. There are many sophisticated methods for balancing exploration & exploitation for particular mathematical formulations of k-armed bandit & related problems. However, most of these methods make strong assumptions about stationary & prior knowledge that are either violated or impossible to verify in most applications & in full RL problem that we consider in subsequent chaps. Guarantees of optimality or bounded loss for these methods are of little comfort when assumptions of their theory do not apply.

– Trong bất kỳ trường hợp cụ thể nào, việc khám phá hay khai thác tốt hơn phụ thuộc rất nhiều vào các giá trị ước lượng chính xác, độ bất định, & số bước còn lại. Có nhiều phương pháp tinh vi để cân bằng giữa khám phá & khai thác cho các công thức toán học cụ thể của máy đánh bạc k-armed bandit & các bài toán liên quan. Tuy nhiên, hầu hết các phương pháp này đều đưa ra những giả định mạnh mẽ về kiến thức dừng & kiến thức đã biết, những giả định này bị vi phạm hoặc không thể xác minh trong hầu hết các ứng dụng & trong bài toán RL đầy đủ mà chúng ta sẽ xem xét trong các chương tiếp theo. Việc đảm bảo tính tối ưu hoặc tổn thất giới hạn cho các phương pháp này không mấy an toàn khi các giả định của lý thuyết của chúng không được áp dụng.

In this book, do not worry about balancing exploration & exploitation in a sophisticated way; worry only about balancing them at all. In this chap, present several simple balancing methods for k-armed bandit problem & show they work much better than methods that always exploit. Need to balance exploration & exploitation is a distinctive challenge that arises in RL; simplicity of our version of k-armed bandit problem enables us to show this in a particularly clear form.

– Trong cuốn sách này, đừng lo lắng về việc cân bằng giữa khám phá & khai thác 1 cách phức tạp; chỉ cần quan tâm đến việc cân bằng chúng. Trong chương này, chúng tôi trình bày 1 số phương pháp cân bằng đơn giản cho bài toán k-armed bandit & cho thấy chúng hiệu quả hơn nhiều so với các phương pháp luôn khai thác. Nhu cầu cân bằng giữa khám phá & khai thác là 1 thách thức đặc biệt phát sinh trong thực tế; sự đơn giản của phiên bản bài toán k-armed bandit của chúng tôi cho phép chúng tôi thể hiện điều này 1 cách đặc biệt rõ ràng.

- **2.2. Action-value Methods.** begin by looking more closely at methods for estimating values of actions & for using estimates to make action selection decisions, which we collectively call *action-value methods*. Recall: true value of an action is mean reward when that action is selected. 1 natural way to estimate this is by averaging rewards actually received: (2.1)

$$Q_t(a) \coloneqq \frac{\text{sum of rewards when } a \text{ taken prior to } t}{\text{number of times } a \text{ taken prior to } t} = \frac{\sum_{i=1}^{t-1} R_i \cdot \mathbf{1}_{A_i=a}}{\sum_{i=1}^{t-1} \cdot \mathbf{1}_{A_i=a}},$$

where $\mathbf{1}_{\text{predicate}}$ denotes random variable that is 1 if *predicate* is true & 0 if it is not. If denominator is 0, then instead define $Q_t(a)$ as some default value, e.g. 0. As denominator goes to ∞, by law of large numbers, $Q_t(a)$ converges to $q_*(a)$. Call this *sample-average* method for estimating action values because each estimate is an average of sample of relevant rewards. Of course this is just 1 way to estimate action values, & not necessarily best one. Nevertheless, for now, say with this simple estimation method & turn to question of how estimates might be used to select actions.

– Phương pháp Hành động-Giá trị. bắt đầu bằng cách xem xét kỹ hơn các phương pháp ước tính giá trị của hành động & sử dụng ước tính để đưa ra quyết định lựa chọn hành động, mà chúng tôi gọi chung là *phương pháp hành động-giá trị*. Nhắc lại: giá trị thực của 1 hành động là phần thưởng trung bình khi hành động đó được chọn. 1 cách tự nhiên để ước tính điều này là lấy trung bình phần thưởng thực tế nhận được: (2.1)

$$Q_t(a) \coloneqq \frac{\text{tổng phần thưởng khi } a \text{ được lấy trước } t}{\text{số lần } a \text{ được lấy trước } t} = \frac{\sum_{i=1}^{t-1} R_i \cdot \mathbf{1}_{A_i=a}}{\sum_{i=1}^{t-1} \cdot \mathbf{1}_{A_i=a}},$$

trong đó $\mathbf{1}_{\text{predicate}}$ biểu thị biến ngẫu nhiên có giá trị 1 nếu *predicate* là đúng & 0 nếu sai. Nếu mẫu số là 0, thì thay vào đó hãy định nghĩa $Q_t(a)$ là 1 giá trị mặc định, ví dụ: 0. Khi mẫu số tiến đến ∞, theo quy luật số lớn, $Q_t(a)$ hội tụ về $q_*(a)$. Gọi đây là phương pháp *sample-average* để ước tính giá trị hành động vì mỗi ước tính là trung bình của 1 mẫu phần thưởng có liên quan. Tất nhiên, đây chỉ là 1 cách để ước tính giá trị hành động, & không nhất thiết là cách tốt nhất. Tuy nhiên, hiện tại, hãy thử sử dụng phương pháp ước tính đơn giản này & chuyển sang câu hỏi về cách sử dụng ước tính để lựa chọn hành động.

Simplest action selection rule: select 1 of actions with highest estimated value, i.e., 1 of greedy actions as defined in prev sect. If there is > 1 greedy action, then a selection is made among them in some arbitrary way, perhaps randomly. Write this *greedy* action selection method as (2.2)

$$A_t \coloneqq \arg\max_a Q_t(a),$$

where $\arg\max_a$ denotes action a for which expression that follows is maximized (with ties broken arbitrarily). Greedy action selection always exploits current knowledge to maximize immediate reward; it spends no time at all sampling apparently inferior actions to see if they might really be better. A simple alternative: behave greedily most of time, but every once in a while, say with small probability ε, instead select randomly from among all actions with equal probability, independently of action-value estimates. Call methods using this near-greedy action selection rule *ε-greedy* methods. An advantage of these methods: in limit as number of steps increases, every action will be sampled an infinite number of times, thus ensuring that all $Q_t(a)$ converge to their respective $q_*(a)$. This of course implies: probability of selecting optimal action converges to $> 1 - \varepsilon$, i.e., to near certainty. These are just asymptotic guarantees, however, & say little about practical effectiveness of methods.

– Quy tắc lựa chọn hành động đơn giản nhất: chọn 1 trong các hành động có giá trị ước tính cao nhất, i.e., 1 trong các hành động tham lam như được định nghĩa trong phần trước. Nếu có > 1 hành động tham lam, thì việc lựa chọn được thực hiện giữa chúng theo 1 cách tùy ý, có thể là ngẫu nhiên. Viết phương pháp lựa chọn hành động *tham lam* này dưới dạng (2.2)

$$A_t \coloneqq \arg\max_a Q_t(a),$$

trong đó $\arg\max_a$ biểu thị hành động a mà biểu thức theo sau được tối đa hóa (với các ràng buộc bị phá vỡ 1 cách tùy ý). Lựa chọn hành động tham lam luôn khai thác kiến thức hiện có để tối đa hóa phần thưởng tức thời; nó không mất thời gian để lấy mẫu các hành động có vẻ kém hơn để xem liệu chúng có thực sự tốt hơn hay không. 1 giải pháp thay thế đơn giản: hành động tham lam hầu hết thời gian, nhưng thỉnh thoảng, với xác suất ε nhỏ, hãy chọn ngẫu nhiên từ tất cả các hành động có xác suất bằng nhau, độc lập với ước tính giá trị hành động. Gọi các phương thức sử dụng quy tắc lựa chọn hành động gần tham lam này *ε-tham lam*. 1 ưu điểm của các phương thức này: khi số bước tăng lên, mỗi hành động sẽ được lấy mẫu vô hạn lần, do đó đảm bảo rằng tất cả $Q_t(a)$ đều hội tụ về $q_*(a)$ tương ứng của chúng. Điều này tất nhiên ngụ ý: xác suất lựa chọn hành động tối ưu hội tụ về $> 1 - \varepsilon$, i.e., gần như chắc chắn. Tuy nhiên, đây chỉ là những đảm bảo tiệm cận, & không nói lên nhiều về hiệu quả thực tế của các phương thức.

Problem 6. *In ε-greedy action selection, for case of 2 actions & $\varepsilon = 0.5$, what is probability that greedy action is selected? – Trong lựa chọn hành động tham lam ε, đối với trường hợp 2 hành động & $\varepsilon = 0,5$, xác suất hành động tham lam được chọn là bao nhiêu?*

- **2.3. 10-armed Testbed.** To roughly assess relative effectiveness of greedy & ε-greedy action-value methods, compared them numerically on a suite of test problems. This was a set of 2000 randomly generated k-armed bandit problems with $k = 10$. For each bandit problem, e.g. one shown in Fig. 2.1: An example bandit problem from 10-armed testbed. True value $q_*(a)$ of esach of 10 actions was selected according to a normal distribution with mean 0 & unit variance, & then actual rewards were selected according to a mean $q_*(a)$, unit-variance normal distribution, as suggested by these gray distributions., action values, $q_*(a)$, $a \in [10]$, were selected according to a normal (Gaussian) distribution with mean 0 & variance 1. Then, when a learning method applied to that problem selected action A_t at time step t, actual reward R_t was selected from a normal distribution with mean $q_*(A_t)$ & variance 1. These distributions are shown in gray in Fig. 2.1. Call this suite of test tasks *10-armed testbed*. For any learning method, can measure its performance & behavior as it improves with experience > 1000 time steps when applied to 1 of bandit problems. This makes up 1 *run*. Repeating this for 2000 independent runs, each with a different bandit problem, obtained measures of learning algorithm's average behavior.

  – 10-armed Testbed. Để đánh giá sơ bộ hiệu quả tương đối của các phương pháp hành động-giá trị tham lam & ε-greedy, chúng tôi đã so sánh chúng bằng số trên 1 bộ bài toán thử nghiệm. Đây là 1 bộ gồm 2000 bài toán bandit k được tạo ngẫu nhiên với $k = 10$. Đối với mỗi bài toán bandit, ví dụ như bài toán được hiển thị trong Hình 2.1: 1 ví dụ về bài toán bandit từ testbed 10-armed. Giá trị thực $q_*(a)$ của mỗi 10 hành động được chọn theo phân phối chuẩn với trung bình 0 & phương sai đơn vị, & sau đó phần thưởng thực tế được chọn theo trung bình $q_*(a)$, phân phối chuẩn phương sai đơn vị, như được gợi ý bởi các phân phối xám này., các giá trị hành động, $q_*(a)$, $a \in [10]$, được chọn theo phân phối chuẩn (Gaussian) với trung bình 0 & phương sai 1. Sau đó, khi 1 phương pháp học được áp dụng cho vấn đề đó đã chọn hành động A_t tại bước thời gian t, phần thưởng thực tế R_t được chọn từ phân phối chuẩn với trung bình $q_*(A_t)$ & phương sai 1. Các phân phối này được hiển thị bằng màu xám trong Hình 2.1. Gọi bộ tác vụ kiểm tra này là *10-armed testbed*. Đối với bất kỳ phương pháp học nào, có thể đo lường hiệu suất & hành vi của nó khi nó cải thiện theo kinh nghiệm > 1000 bước thời gian khi được áp dụng cho 1 trong số các bài toán bandit. Điều này tạo nên 1 *run*. Lặp lại điều này trong 2000 lần chạy độc lập, mỗi lần chạy với 1 bài toán cướp khác nhau, thu được các phép đo về hành vi trung bình của thuật toán học.

  Fig. 2.2: Average performance of ε-greedy action-value methods on 10-armed testbed. These data are averages > 2000 runs with different bandit problems. All methods used sample averages as their action-value estimates. compares a greedy method with 2 ε-greedy methods ($\varepsilon = 0.01, \varepsilon = 0.1$), as described above, on 10-armed testbed. All methods formed their action-value estimates using sample-average technique (with an initial estimate of 0). Upper graph shows increase in expected reward with experience. Greedy method improved slightly faster than order methods at very beginning, but then leveled off at a lower level. It achieved a reward-per-step of only about 1, compared with best possible of abut 1.54 on this testbed. Greedy method performed significantly worse in long run because it often got struck performing suboptimal actions. Lower graph shows: greedy method found optimal action in only $\approx \frac{1}{3}$ of tasks. In other $\frac{2}{3}$, its initial samples of optimal action were disappointing, & it never returned to it. ε-greedy methods eventually performed better because they continued to explore & to improve their chances of recognizing optimal action. $\varepsilon = 0.1$ method explored more, & usually found optimal action earlier, but it never selected that action $> 91\%$ of time. $\varepsilon = 0.01$ method improved more slowly, but eventually would perform better than $\varepsilon = 0.1$ method on both performance measures shown in figure. Also possible to reduce ε over time to try to get best of both high & low values.

  – Hình 2.2: Hiệu suất trung bình của các phương pháp giá trị hành động ε-tham lam trên nền tảng thử nghiệm 10 cánh tay. Những dữ liệu này là trung bình > 2000 lần chạy với các vấn đề bandit khác nhau. Tất cả các phương pháp đều sử dụng trung bình mẫu làm ước tính giá trị hành động của chúng. so sánh 1 phương pháp tham lam với 2 phương pháp tham lam ε ($\varepsilon = 0,01, \varepsilon = 0,1$), như mô tả ở trên, trên nền tảng thử nghiệm 10 cánh tay. Tất cả các phương pháp đều hình thành ước tính giá trị hành động của chúng bằng kỹ thuật trung bình mẫu (với ước tính ban đầu là 0). Đồ thị trên cho thấy phần thưởng kỳ vọng tăng lên theo kinh nghiệm. Phương pháp tham lam cải thiện nhanh hơn 1 chút so với các phương pháp thứ tự ở giai đoạn đầu, nhưng sau đó ổn định ở mức thấp hơn. Nó đạt được phần thưởng cho mỗi bước chỉ khoảng 1, so với mức tốt nhất có thể là khoảng 1,54 trên nền tảng thử nghiệm này. Phương pháp tham lam hoạt động kém hơn đáng kể trong thời gian dài vì nó thường bị tấn công khi thực hiện các hành động không tối ưu. Biểu đồ bên dưới cho thấy: phương pháp tham lam chỉ tìm thấy hành động tối ưu trong $\approx \frac{1}{3}$ nhiệm vụ. Trong $\frac{2}{3}$ khác, các mẫu hành động tối ưu ban đầu của nó không được như mong đợi, & nó không bao giờ quay lại với nó. Các phương pháp ε-tham lam cuối cùng đã hoạt động tốt hơn vì chúng tiếp tục khám phá & để cải thiện cơ hội nhận ra hành động tối ưu. Phương pháp $\varepsilon = 0.1$ đã khám phá nhiều hơn, & thường tìm thấy hành động tối ưu sớm hơn, nhưng nó không bao giờ chọn hành động đó $> 91\%$ thời gian. Phương pháp $\varepsilon = 0.01$ cải thiện chậm hơn, nhưng cuối cùng sẽ hoạt động tốt hơn phương pháp $\varepsilon = 0.1$ trên cả 2 thước đo hiệu suất được hiển thị trong hình. Cũng có thể giảm ε theo thời gian để cố gắng đạt được kết quả tốt nhất ở cả giá trị cao & thấp.

  Advantage of ε-greedy over greedy methods depends on task. E.g., suppose reward variance had been larger, say 10 instead of 1. With noisier rewards it takes more exploration to find optimal action, & ε-greedy methods should fare even better relative to greedy method. On other hand, if reward variances were 0, then greedy method would know true value of each action after trying it once. In this case, greedy method might actually perform best because it would soon find optimal action & then never explore. But even in deterministic case there is a a large advantage to exploring if we weaken some of other assumptions. E.g., suppose bandit task were nonstationary, i.e., true values of actions changed over time. In this case exploration is needed even in deterministic case to make sure 1 of nongreedy actions has not changed to become better than greedy one. As will see in next few chaps, nonstationarity is case most commonly encountered in RL. Even if underlying

task is stationary & deterministic, learner faces a set of banditlike decision tasks each of which changes over time as learning proceeds & agent's decision-making policy changes. RL requires a balance between exploration & exploitation.

– Ưu điểm của phương pháp ε-tham lam so với phương pháp ε-tham lam phụ thuộc vào nhiệm vụ. Ví dụ, giả sử phương sai phần thưởng lớn hơn, chẳng hạn 10 thay vì 1. Với phần thưởng nhiễu hơn, cần nhiều lần khám phá hơn để tìm ra hành động tối ưu, & phương pháp ε-tham lam thậm chí còn tốt hơn phương pháp tham lam. Mặt khác, nếu phương sai phần thưởng bằng 0, thì phương pháp tham lam sẽ biết giá trị thực của mỗi hành động sau khi thử 1 lần. Trong trường hợp này, phương pháp tham lam thực sự có thể hoạt động tốt nhất vì nó sẽ sớm tìm ra hành động tối ưu & sau đó không bao giờ khám phá. Nhưng ngay cả trong trường hợp xác định, việc khám phá vẫn có 1 lợi thế lớn nếu chúng ta làm suy yếu 1 số giả định khác. Ví dụ, giả sử nhiệm vụ cướp không dừng, i.e., giá trị thực của các hành động thay đổi theo thời gian. Trong trường hợp này, việc khám phá là cần thiết ngay cả trong trường hợp xác định để đảm bảo rằng 1 trong các hành động không tham lam không thay đổi để trở nên tốt hơn hành động tham lam. Như sẽ thấy trong vài chương tiếp theo, tính không dừng là trường hợp thường gặp nhất trong RL. Ngay cả khi nhiệm vụ cơ bản là tĩnh & xác định, người học vẫn phải đối mặt với 1 loạt các nhiệm vụ quyết định giống như trò chơi cướp bóc, mỗi nhiệm vụ thay đổi theo thời gian khi quá trình học diễn ra & chính sách ra quyết định của tác nhân thay đổi. RL đòi hỏi sự cân bằng giữa khám phá & khai thác.

Problem 7. *In comparison shown in Fig. 2.2, which method will perform best in long run in terms of cumulative reward & probability of selecting best action? How much better will it be? Express your answer quantitatively.*

– So sánh với Hình 2.2, phương pháp nào sẽ hiệu quả hơn về lâu dài xét về phần thưởng tích lũy & xác suất lựa chọn hành động tốt nhất? Phương pháp đó sẽ tốt hơn bao nhiêu? Hãy thể hiện câu trả lời của bạn bằng định lượng.

- **2.4. Incremental Implementation.** Action-value methods discussed so far all estimate action values as sample averages of observed rewards. Now turn to question of how these averages can be computed in a computationally efficient manner, in particular, with constant memory & constant per-time-step computation.

  – Triển khai Gia tăng. Các phương pháp giá trị hành động đã thảo luận cho đến nay đều ước tính giá trị hành động dưới dạng trung bình mẫu của phần thưởng quan sát được. Bây giờ hãy chuyển sang câu hỏi làm thế nào để tính toán các giá trị trung bình này 1 cách hiệu quả về mặt tính toán, đặc biệt là với bộ nhớ không đổi & tính toán không đổi theo từng bước thời gian.

  To simplify notation, concentrate on a single action. Let R_i now denote reward received after ith selection of *this action*, & let Q_n denote estimate of its action value after it has been selected $n-1$ times, which we can now write simply as

$$Q_n \coloneqq \frac{\sum_{i=1}^{n-1} R_i}{n-1}.$$

  Obvious implementation would be maintain a record of all rewards & then perform this computation whenever estimated value was needed. However, if this is done, then memory & computational requirements would grow over time as more rewards are seen. Each additional reward would require additional memory to store it & additional computation to compute sum in numerator.

  – Để đơn giản hóa ký hiệu, hãy tập trung vào 1 hành động duy nhất. Giả sử R_i biểu thị phần thưởng nhận được sau lần chọn thứ i của *this action*, & giả sử Q_n biểu thị ước tính giá trị hành động của nó sau khi được chọn $n-1$ lần, mà giờ đây chúng ta có thể viết đơn giản là

$$Q_n \coloneqq \frac{\sum_{i=1}^{n-1} R_i}{n-1}.$$

  Cách triển khai rõ ràng sẽ là duy trì 1 bản ghi của tất cả phần thưởng & sau đó thực hiện phép tính này bất cứ khi nào cần giá trị ước tính. Tuy nhiên, nếu thực hiện điều này, thì bộ nhớ & yêu cầu tính toán sẽ tăng theo thời gian khi có thêm phần thưởng. Mỗi phần thưởng bổ sung sẽ yêu cầu thêm bộ nhớ để lưu trữ & thêm phép tính để tính tổng trong tử số.

  As you might suspect, this is not really necessary. Easy to devise incremental formulas for updating averages with small, constant computation required to process each new reward. Given Q_n & nth reward R_n, new average of all n rewards can be computed by (2.3)

$$Q_{n+1} = \frac{1}{n}\sum_{i=1}^{n} R_i = \cdots = Q_n + \frac{R_n - Q_n}{n},$$

  which holds even for $n=1$, obtaining $Q_2 = R_1$ for arbitrary Q_1. This implementation requires memory only for Q_n & n, & only small computation (2.3) for each new reward.

  – Như bạn có thể nghi ngờ, điều này thực sự không cần thiết. Dễ dàng thiết kế các công thức gia tăng để cập nhật giá trị trung bình với các phép tính nhỏ, không đổi cần thiết để xử lý mỗi phần thưởng mới. Với Q_n & n phần thưởng thứ R_n, giá trị trung bình mới của tất cả n phần thưởng có thể được tính theo (2.3)

$$Q_{n+1} = \frac{1}{n}\sum_{i=1}^{n} R_i = \cdots = Q_n + \frac{R_n - Q_n}{n},$$

  điều này đúng ngay cả với $n=1$, thu được $Q_2 = R_1$ cho Q_1 bất kỳ. Việc triển khai này chỉ yêu cầu bộ nhớ cho Q_n & n, & chỉ cần phép tính nhỏ (2.3) cho mỗi phần thưởng mới.

  This update rule (2.3) is of a form that occurs frequently throughout this book. General form is (2.4)

$$\text{NewEstimate} \leftarrow \text{OldEstimate} + \text{StepSize}[\text{Target} - \text{OldEstimate}].$$

Expression [Target − OldEstimate] is an *error* in estimate. It is reduced by taking a step toward "Target". Target is presumed to indicate a desirable direction in which to move, thought it may be noisy. In case above, e.g., target is nth reward.

– Quy tắc cập nhật (2.3) này có dạng thường xuyên xuất hiện trong suốt cuốn sách này. Dạng tổng quát là (2.4)

$$\text{NewEstimate} \leftarrow \text{OldEstimate} + \text{StepSize}[\text{Target} - \text{OldEstimate}].$$

Biểu thức [Target − OldEstimate] là 1 *error* trong ước tính. Nó được rút gọn bằng cách thực hiện 1 bước tiến về phía "Target". Target được cho là chỉ ra 1 hướng mong muốn để di chuyển, mặc dù nó có thể bị nhiễu. Trong trường hợp trên, ví dụ, target là phần thưởng thứ n.

Note: step-size parameter StepSize used in incremental method (2.3) changes from time step to time step. In processing nth reward for action a, method uses step-size parameter $\frac{1}{n}$. In this book, denote step-size parameter by α or, more generally, by $\alpha_t(a)$.

– Lưu ý: tham số kích thước bước StepSize được sử dụng trong phương pháp gia tăng (2.3) thay đổi theo từng bước thời gian. Khi xử lý phần thưởng thứ n cho hành động a, phương pháp sử dụng tham số kích thước bước $\frac{1}{n}$. Trong sách này, hãy ký hiệu tham số kích thước bước là α hoặc, tổng quát hơn, là $\alpha_t(a)$.

Pseucode for a complete bandit algorithm using incrementally computed sample averages & ε-greedy action selection is shown in box below. Function bandit(a) is assumed to take an action & return a corresponding reward.

– Mã giả cho 1 thuật toán cướp hoàn chỉnh sử dụng trung bình mẫu được tính toán gia tăng & lựa chọn hành động tham lam ε được hiển thị trong hộp bên dưới. Hàm bandit(a) được giả định thực hiện 1 hành động & trả về phần thưởng tương ứng.

- **2.5. Tracking a Nonstationary Problem.** Averaging methods discussed so far are appropriate for stationary bandit problems, i.e., for bandit problems in which reward probabilities do not change over time. As noted earlier, often encounter RL problems that are effectively nonstationary. In such cases it makes sense to give more weight to recent rewards than to long-past rewards. 1 of most popular ways of doing this: use a constant step-size parameter. E.g., incremental update rule (2.3) for updating an average Q_n of $n-1$ past rewards is modified to be

$$Q_{n+1} \coloneqq Q_n + \alpha[R_n - Q_n],$$

where step-size parameter $\alpha \in (0, 1]$ is constant. This results in Q_{n+1} being a weighted average of past rewards & initial estimate Q_1: (2.6)

$$Q_{n+1} = \cdots = (1-\alpha)^n Q_1 + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} R_i.$$

Call this a weighted average because sum of weights is $(1-\alpha)^n + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} = 1$. Note: weight $\alpha(1-\alpha)^{n-i}$, given to reward R_i depends on how many rewards ago, $n-i$, it was observed. Quantity $1-\alpha < 1$, & thus weight given to R_i decreases as number of intervening rewards increases. In fact, weight decays exponentially according to exponent on $1-\alpha$. (If $1-\alpha = 0$, then all weight goes on very last reward R_n, because of convention that $O^0 = 1$.) Accordingly, this is sometimes called an *exponential recency-weighted average*.

– Theo dõi 1 Bài toán Không dừng. Các phương pháp tính trung bình đã thảo luận cho đến nay phù hợp với các bài toán bandit dừng, i.e., các bài toán bandit trong đó xác suất phần thưởng không thay đổi theo thời gian. Như đã lưu ý trước đó, thường gặp phải các bài toán RL về cơ bản là không dừng. Trong những trường hợp như vậy, việc chú trọng hơn vào phần thưởng gần đây hơn là phần thưởng đã qua lâu là hợp lý. 1 trong những cách phổ biến nhất để thực hiện việc này: sử dụng tham số kích thước bước không đổi. Ví dụ, quy tắc cập nhật gia tăng (2.3) để cập nhật Q_n trung bình của $n-1$ phần thưởng trong quá khứ được sửa đổi thành

$$Q_{n+1} \coloneqq Q_n + \alpha[R_n - Q_n],$$

trong đó tham số kích thước bước $\alpha \in (0, 1]$ là hằng số. Điều này dẫn đến Q_{n+1} là trung bình có trọng số của phần thưởng trong quá khứ & ước lượng ban đầu Q_1: (2.6)

$$Q_{n+1} = \cdots = (1-\alpha)^n Q_1 + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} R_i.$$

Gọi đây là trung bình có trọng số vì tổng trọng số là $(1-\alpha)^n + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} = 1$. Lưu ý: trọng số $\alpha(1-\alpha)^{n-i}$, được trao cho phần thưởng R_i phụ thuộc vào số phần thưởng trước đó, $n-i$, đã được quan sát. Số lượng $1-\alpha < 1$, & do đó trọng số được trao cho R_i giảm khi số phần thưởng xen kẽ tăng. Trên thực tế, trọng số giảm theo cấp số nhân theo số mũ của $1-\alpha$. (Nếu $1-\alpha = 0$, thì toàn bộ trọng số sẽ được chuyển sang phần thưởng cuối cùng R_n, do quy ước $O^0 = 1$.) Do đó, điều này đôi khi được gọi là *trung bình có trọng số gần đây theo cấp số nhân*.

Sometimes convenient to vary step-size parameter from step to step. Let $\alpha_n(a)$ denote step-size parameter used to process reward received after nth selection of action a. As have noted, choice $\alpha_n(a) = \frac{1}{n}$ results in sample-average method, which is guaranteed to converge to true action values by law of large numbers. But of course convergence is not guaranteed for all choices of sequence $\{\alpha_n(a)\}$. A well-known result in stochastic approximation theory gives us conditions required to assure convergence with probability 1: (2.7)

$$\sum_{n=1}^{\infty} \alpha_n(a) = \infty, \ \sum_{n=1}^{\infty} \alpha_n^2(a) < \infty.$$

1st condition is required to guarantee: steps are large enough to eventually overcome any initial conditions or random fluctuations. 2nd condition guarantees that eventually steps become small enough to assure convergence.

– Đôi khi thuận tiện để thay đổi tham số kích thước bước từ bước này sang bước khác. Giả sử $\alpha_n(a)$ biểu thị tham số kích thước bước được sử dụng để xử lý phần thưởng nhận được sau lần lựa chọn hành động a thứ n. Như đã lưu ý, lựa chọn $\alpha_n(a) = \frac{1}{n}$ dẫn đến phương pháp trung bình mẫu, được đảm bảo hội tụ về giá trị hành động thực theo quy luật số lớn. Nhưng tất nhiên, sự hội tụ không được đảm bảo cho mọi lựa chọn của chuỗi $\{\alpha_n(a)\}$. 1 kết quả nổi tiếng trong lý thuyết xấp xỉ ngẫu nhiên cung cấp cho chúng ta các điều kiện cần thiết để đảm bảo sự hội tụ với xác suất 1: (2.7)

$$\sum_{n=1}^{\infty} \alpha_n(a) = \infty, \sum_{n=1}^{\infty} \alpha_n^2(a) < \infty.$$

Điều kiện thứ nhất cần đảm bảo: các bước đủ lớn để cuối cùng vượt qua mọi điều kiện ban đầu hoặc biến động ngẫu nhiên. Điều kiện thứ 2 đảm bảo rằng cuối cùng các bước sẽ đủ nhỏ để đảm bảo sự hội tụ.

Note both convergence conditions are met for sample-average case, $\alpha_n(a) = \frac{1}{n}$, but not for case of constant step-size parameter, $\alpha_n(a)$. In latter case, 2nd condition is not met, indicating: estimates never completely converge but2 continue to vary in response to most recently received rewards. As mentioned above, this is actually desirable in a nonstationary environment, & problems that are effectively nonstationary are most common in RL. In addition, sequences of step-size parameters that meet conditions (2.7) often converge very slowly or need considerable tuning in order to obtain a satisfactory convergence rate. Although sequences of step-size parameters that meet these convergence conditions are often used in theoretical work, they are seldom used in applications & empirical research.

– Lưu ý cả 2 điều kiện hội tụ đều được đáp ứng đối với trường hợp trung bình mẫu, $\alpha_n(a) = \frac{1}{n}$, nhưng không đáp ứng đối với trường hợp tham số kích thước bước không đổi, $\alpha_n(a)$. Trong trường hợp sau, điều kiện thứ 2 không được đáp ứng, cho thấy: các ước tính không bao giờ hội tụ hoàn toàn mà2 tiếp tục thay đổi để đáp ứng với phần thưởng nhận được gần đây nhất. Như đã đề cập ở trên, điều này thực sự mong muốn trong môi trường không dừng, & các vấn đề thực sự không dừng là phổ biến nhất trong RL. Ngoài ra, các chuỗi tham số kích thước bước đáp ứng các điều kiện (2.7) thường hội tụ rất chậm hoặc cần điều chỉnh đáng kể để có được tốc độ hội tụ thỏa đáng. Mặc dù các chuỗi tham số kích thước bước đáp ứng các điều kiện hội tụ này thường được sử dụng trong công việc lý thuyết, nhưng chúng hiếm khi được sử dụng trong các ứng dụng & nghiên cứu thực nghiệm.

Problem 8. *If step-size parameters α_n are not constant, then estimate Q_n is a weighted average of previously received rewards with a weighting different from that given by (2.6). What is weighting on each prior reward for general case, analogous to (2.6), in terms of sequence of step-size parameters?*

– Nếu các tham số kích thước bước α_n không phải là hằng số, thì ước tính Q_n là giá trị trung bình có trọng số của các phần thưởng đã nhận trước đó với trọng số khác với trọng số được cho bởi (2.6). Trọng số của mỗi phần thưởng trước đó trong trường hợp tổng quát, tương tự như (2.6), là bao nhiêu về mặt trình tự các tham số kích thước bước?

Problem 9 (Programming)**.** *Design & conduct an experiment to demonstrate difficulties that sample-average methods have for nonstationary problems: Use a modified version of 10-armed testbed in which all $q_*(a)$ start out equal & then take independent random walks (say by adding a normally distributed increment with mean* 0 *& standard deviation* 0.01 *to all $q_*(a)$ on each step). Prepare plots like Fig. 2.2 for an action-value method using sample averages, incrementally computed, & another action-value method using a constant step-size parameter, $\alpha = 0.1$. Use $\varepsilon = 0.1$ & longer runs, say of* 10000 *steps.*

– Thiết kế & tiến hành 1 thí nghiệm để chứng minh những khó khăn mà các phương pháp trung bình mẫu gặp phải đối với các vấn đề không dừng: Sử dụng phiên bản đã sửa đổi của nền tảng thử nghiệm 10 cánh tay trong đó tất cả $q_*(a)$ đều bắt đầu bằng nhau & sau đó thực hiện các bước đi ngẫu nhiên độc lập (ví dụ bằng cách thêm 1 gia số phân phối chuẩn với trung bình 0 *& độ lệch chuẩn* 0,01 *vào tất cả $q_*(a)$ ở mỗi bước). Chuẩn bị các biểu đồ như Hình 2.2 cho phương pháp giá trị hành động sử dụng trung bình mẫu, được tính toán gia tăng, & 1 phương pháp giá trị hành động khác sử dụng tham số kích thước bước không đổi, $\alpha = 0,1$. Sử dụng $\varepsilon = 0,1$ & các lần chạy dài hơn, ví dụ* 10000 *bước.*

- **2.6. Optimistic Initial Values.** All methods we have discussed so far are dependent to some extent on initial action-value estimates, $Q_1(a)$. In language of statistics, these methods are *biased* by their initial estimates. For sample-average methods, bias disappears once all actions have been selected at least once, but for methods with constant α, bias is permanent, though decreasing over time as given by (2.6). In practice, this kind of bias is usually not a problem & can sometimes be very helpful. Downside: initial estimates become, in effect, a set of parameters that must be picked by user, if only to set them all to 0. Upside: they provide an easy way to supply some prior knowledge about what level of rewards can be expected.

– Giá trị ban đầu lạc quan. Tất cả các phương pháp chúng ta đã thảo luận cho đến nay đều phụ thuộc ở 1 mức độ nào đó vào ước tính giá trị hành động ban đầu, $Q_1(a)$. Theo ngôn ngữ thống kê, các phương pháp này *bị sai lệch* bởi các ước tính ban đầu của chúng. Đối với các phương pháp trung bình mẫu, sai lệch biến mất khi tất cả các hành động đã được chọn ít nhất 1 lần, nhưng đối với các phương pháp có α không đổi, sai lệch là vĩnh viễn, mặc dù giảm dần theo thời gian như được cho bởi (2.6). Trong thực tế, loại sai lệch này thường không phải là vấn đề & đôi khi có thể rất hữu ích. Nhược điểm: các ước tính ban đầu, trên thực tế, trở thành 1 tập hợp các tham số phải được người dùng chọn, nếu chỉ để đặt tất cả chúng thành 0. Ưu điểm: chúng cung cấp 1 cách dễ dàng để cung cấp 1 số kiến thức trước đó về mức phần thưởng có thể mong đợi.

Initial action values can also be used as a simple way to encourage exploration. Suppose: instead of setting initial action values to 0, as we did in 10-armed testbed, set them all to +5. Recall: $q_*(a)$ in this problem are selected from a normal distribution with mean 0 & variance 1. An initial estimate of +5 is thus wildly optimistic. But this optimism encourages

action-value methods to explore. Whichever actions are initially selected, reward is less than starting estimates; learner switches to other actions, being "disappointed" with rewards it is receiving. Result: all actions are tried several times before value estimates converge. System does a fair amount of exploration even if greedy actions are selected all time.

– Giá trị hành động ban đầu cũng có thể được sử dụng như 1 cách đơn giản để khuyến khích khám phá. Giả sử: thay vì đặt giá trị hành động ban đầu thành 0, như chúng ta đã làm trong thử nghiệm 10 cánh tay, hãy đặt tất cả chúng thành +5. Nhớ lại: $q_*(a)$ trong bài toán này được chọn từ phân phối chuẩn với trung bình 0 & phương sai 1. Do đó, ước tính ban đầu là +5 là quá lạc quan. Nhưng sự lạc quan này khuyến khích các phương pháp giá trị hành động khám phá. Bất kỳ hành động nào được chọn ban đầu, phần thưởng sẽ thấp hơn ước tính ban đầu; người học chuyển sang các hành động khác, "thất vọng" với phần thưởng mà nó nhận được. Kết quả: tất cả các hành động được thử nhiều lần trước khi ước tính giá trị hội tụ. Hệ thống thực hiện 1 lượng khám phá kha khá ngay cả khi các hành động tham lam được chọn mọi lúc.

Fig. 2.3: Effect of optimistic initial action-value estimates on 10-armed testbed. Both methods used a constant step-size parameter $\alpha = 0.1$. shows performance on 10-armed bandit testbed of a greedy method using $Q_1(a) = +5$, $\forall a$. For comparison, also shown is an ε-greedy method with $Q_1(a) = 0$. Initially, optimistic method performs worse because it explores more, but eventually it performs better because its exploration decreases with time. Call this technique for encouraging exploration *optimistic initial values*. Regard it as a simple trick that can be quite effective on stationary problems, but it is far from being a generally useful approach to encouraging exploration. E.g., not well suited to nonstationary problem because its derive for exploration is inherently temporary. If task changes, creating a renewed need for exploration, this method cannot help. Indeed, any method that focuses on initial conditions in any special way is unlikely to help with general nonstationary case. Beginning of time occurs only once, & thus we should not focus on it too much. This criticism applies as well to sample-average methods, which also treat beginning of time as a special event, averaging all subsequent rewards with equal weights. Nevertheless, all of these methods are very simple, & 1 of them – or some simple combination of them – is often adequate in practice. In rest of this book, make frequent use of several of these simple exploration techniques.

– Hình 2.3: Ảnh hưởng của ước tính giá trị hành động ban đầu lạc quan trên nền tảng thử nghiệm 10 cánh tay. Cả 2 phương pháp đều sử dụng tham số kích thước bước không đổi $\alpha = 0, 1$. cho thấy hiệu suất trên nền tảng thử nghiệm 10 cánh tay của phương pháp tham lam sử dụng $Q_1(a) = +5$, $\forall a$. Để so sánh, cũng được hiển thị là phương pháp tham lam ε với $Q_1(a) = 0$. Ban đầu, phương pháp lạc quan hoạt động kém hơn vì nó khám phá nhiều hơn, nhưng cuối cùng nó hoạt động tốt hơn vì khả năng khám phá của nó giảm theo thời gian. Gọi kỹ thuật này là để khuyến khích khám phá *giá trị ban đầu lạc quan*. Hãy coi nó như 1 mẹo đơn giản có thể khá hiệu quả đối với các vấn đề dừng, nhưng nó còn lâu mới là 1 cách tiếp cận hữu ích để khuyến khích khám phá. Ví dụ: không phù hợp với vấn đề không dừng vì đạo hàm của nó để khám phá về cơ bản là tạm thời. Nếu nhiệm vụ thay đổi, tạo ra nhu cầu khám phá mới, phương pháp này không thể giúp ích. Thật vậy, bất kỳ phương pháp nào tập trung vào các điều kiện ban đầu theo bất kỳ cách đặc biệt nào đều khó có thể giúp ích cho trường hợp phi dừng tổng quát. Sự khởi đầu của thời gian chỉ xảy ra 1 lần, & do đó chúng ta không nên tập trung quá nhiều vào nó. Lời chỉ trích này cũng áp dụng cho các phương pháp trung bình mẫu, vốn cũng coi sự khởi đầu của thời gian là 1 sự kiện đặc biệt, tính trung bình tất cả các phần thưởng tiếp theo với trọng số bằng nhau. Tuy nhiên, tất cả các phương pháp này đều rất đơn giản, & 1 trong số chúng – hoặc 1 sự kết hợp đơn giản nào đó – thường là đủ trong thực tế. Trong phần còn lại của cuốn sách này, hãy thường xuyên sử dụng 1 số kỹ thuật khám phá đơn giản này.

Problem 10 (Mysterious Spikes)**.** *Results shown in Fig. 2.3 should be quite reliable because they are averages* > 2000 *individual, randomly chosen 10-armed bandit tasks. Why, then, are there oscillations & spikes in early part of curve for optimistic method? I.e., what might make this method perform particularly better or worse, on average, on particular early steps?*

– Kết quả thể hiện trong Hình 2.3 khá đáng tin cậy vì chúng là các giá trị trung bình > 2000 *của các tác vụ máy đánh bạc 10 tay được chọn ngẫu nhiên. Vậy tại sao lại có dao động & gai ở phần đầu đường cong đối với phương pháp lạc quan? Nghĩa là, điều gì có thể khiến phương pháp này hoạt động tốt hơn hoặc kém hơn, xét về mặt trung bình, ở các bước đầu cụ thể?*

Problem 11 (Unbiased Constant-Step-Size Trick.)**.** *In most of this chap, have used sample averages to estimate action values because sample averages do not produce initial bias that constant step sizes do (see analysis leading to (2.6)). However, sample averages are not a completely satisfactory solution because they may perform poorly on nonstationary problems. Is it possible to avoid bias of constant step sizes while retaining their advantages on nonstationary problems? 1 way: use a step size of* $\beta_n \coloneqq \frac{\alpha}{\overline{o}_n}$*, to process nth reward for a particular action, where* $\alpha > 0$ *is a conventional constant step size, &* $\overline{o}_n$ *is a trace of one that starts at 0:*

$$\overline{o}_n \coloneqq \overline{o}_{n-1} + \alpha(1 - \overline{o}_{n-1}), \ \forall n \in \mathbb{N}^\star, \ \textit{with } \overline{o}_n \coloneqq 0.$$

Carry out an analysis like that in (2.6) to show: Q_n *is an exponential recency-weighted average* without initial bias.

– Trong hầu hết chương này, chúng tôi đã sử dụng trung bình mẫu để ước tính giá trị hành động vì trung bình mẫu không tạo ra độ lệch ban đầu như kích thước bước không đổi (xem phân tích dẫn đến (2.6)). Tuy nhiên, trung bình mẫu không phải là 1 giải pháp hoàn toàn thỏa đáng vì chúng có thể hoạt động kém trong các bài toán không dừng. Liệu có thể tránh được độ lệch của kích thước bước không đổi trong khi vẫn giữ được ưu điểm của chúng trong các bài toán không dừng không? Cách 1: sử dụng kích thước bước $\beta_n \coloneqq \frac{\alpha}{\overline{o}_n}$*, để xử lý phần thưởng thứ n cho 1 hành động cụ thể, trong đó* $\alpha > 0$ *là kích thước bước không đổi thông thường, &* $\overline{o}_n$ *là dấu vết của 1 bắt đầu từ 0:*

$$\overline{o}_n \coloneqq \overline{o}_{n-1} + \alpha(1 - \overline{o}_{n-1}), \ \forall n \in \mathbb{N}^\star, \ \textit{với } \overline{o}_n \coloneqq 0.$$

Thực hiện phân tích tương tự như trong (2.6) để chứng minh: Q_n *là trung bình trọng số mũ gần đây* không có độ lệch ban đầu.

◦ **2.7. Upper-Confidence-Bound Action Selection.** Exploration is needed because there is always uncertainty about accuracy of action-value estimates. Greedy actions are those that look best at present, but some of other actions may actually be better. ε-greedy action selection forces non-greedy actions to be tried, but indiscriminately, with no preference for those that are nearly greedy or particularly uncertain. It would be better to select among non-greedy actions according to their potential for actually being optimal, taking into account both how close their estimates are to being maximal & uncertainties in those estimates. 1 effective way of doing this is to select actions according to (2.10)

$$A_t \coloneqq \arg\max_a \left[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right],$$

where $\ln t$ denotes natural logarithm of t, $N_t(a)$ denotes number of times that action a has been selected prior to time t (denominator in (2.1)), & number $c > 0$ controls degree of exploration. If $N_t(a) = 0$, then a is considered to be a maximizing action.

– Lựa chọn Hành động Giới hạn Độ tin cậy Trên. Việc khám phá là cần thiết vì luôn có sự không chắc chắn về độ chính xác của các ước tính giá trị hành động. Các hành động tham lam là những hành động có vẻ tốt nhất hiện tại, nhưng 1 số hành động khác thực sự có thể tốt hơn. Lựa chọn hành động tham lam ε buộc phải thử các hành động không tham lam, nhưng 1 cách bừa bãi, không ưu tiên những hành động gần tham lam hoặc đặc biệt không chắc chắn. Sẽ tốt hơn nếu lựa chọn giữa các hành động không tham lam theo tiềm năng tối ưu thực sự của chúng, đồng thời tính đến cả mức độ gần với ước tính tối đa của chúng & mức độ không chắc chắn trong các ước tính đó. 1 cách hiệu quả để thực hiện điều này là chọn các hành động theo (2.10)

$$A_t \coloneqq \arg\max_a \left[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right],$$

trong đó $\ln t$ biểu thị logarit tự nhiên của t, $N_t(a)$ biểu thị số lần hành động a được chọn trước thời điểm t (mẫu số trong (2.1)), & số $c > 0$ kiểm soát mức độ khám phá. Nếu $N_t(a) = 0$, thì a được coi là 1 hành động tối đa hóa.

Idea of this *upper confidence bound* (UCB) action selection: square-root term is a measure of uncertainty or variance in estimate of a's value. Quantity being maxed over is thus a sort of upper bound on possible true value of action a, with c determining confidence level. Each time a is selected, uncertainty is presumably reduced: $N_t(a)$ increments, &, as it appears in denominator, uncertainty term decreases. On other hand, each time an action other than a is selected, t increases but $N_t(a)$ does not; because t appears in numerator, uncertainty estimate increases. Use of natural logarithm means: increases get smaller over time, but are unbounded; all actions will eventually be selected, but actions with lower value estimates, or that have already been selected frequently, will be selected with decreasing frequency over time.

– Ý tưởng về lựa chọn hành động *giới hạn tin cậy trên* (UCB): hạng tử căn bậc 2 là thước đo mức độ không chắc chắn hoặc phương sai trong ước tính giá trị của a. Do đó, số lượng được tối đa hóa là 1 loại giới hạn trên đối với giá trị thực có thể có của hành động a, với c xác định mức độ tin cậy. Mỗi lần a được chọn, mức độ không chắc chắn được cho là giảm đi: $N_t(a)$ tăng lên, &, khi xuất hiện ở mẫu số, hạng tử không chắc chắn giảm đi. Mặt khác, mỗi lần chọn 1 hành động khác với a, t tăng nhưng $N_t(a)$ thì không; vì t xuất hiện ở tử số, ước tính không chắc chắn tăng lên. Sử dụng logarit tự nhiên có nghĩa là: mức tăng giảm dần theo thời gian nhưng không bị giới hạn; tất cả các hành động cuối cùng sẽ được chọn, nhưng các hành động có ước tính giá trị thấp hơn hoặc đã được chọn thường xuyên sẽ được chọn với tần suất giảm dần theo thời gian.

Results with UCB on 10-armed testbed are shown in Fig. 2.4: Average performance of UCB action selection on the 10-armed testbed. As shown, UCB generally performs better than ε-greedy action selection, except in 1st k steps, when it selects randomly among as-yet-untried actions. UCB often performs well, as shown here, but is more difficult than ε-greedy to extend beyond bandits to more general RL settings considered in rest of this book. 1 difficulty is in dealing with nonstationary problems; methods more complex than those presented in Sect. 2.5 would be needed. Another difficulty is dealing with large state spaces, particularly when using function approximation as developed in Part II of this book. In these more advanced settings, idea of UCB action selection is usually not practical.

– Kết quả với UCB trên nền tảng thử nghiệm có 10 cánh tay được hiển thị trong Hình 2.4: Hiệu suất trung bình của lựa chọn hành động UCB trên nền tảng thử nghiệm có 10 cánh tay. Như đã hiển thị, UCB thường hoạt động tốt hơn lựa chọn hành động ε-tham lam, ngoại trừ trong k bước đầu tiên, khi nó chọn ngẫu nhiên trong số các hành động chưa được thử. UCB thường hoạt động tốt, như được hiển thị ở đây, nhưng khó hơn ε-tham lam để mở rộng ra ngoài các nhóm cướp đến các thiết lập RL tổng quát hơn được xem xét trong phần còn lại của cuốn sách này. 1 khó khăn là giải quyết các vấn đề không dừng; cần các phương pháp phức tạp hơn những phương pháp được trình bày trong Phần 2.5. 1 khó khăn khác là xử lý các không gian trạng thái lớn, đặc biệt là khi sử dụng xấp xỉ hàm như đã phát triển trong Phần II của cuốn sách này. Trong các thiết lập nâng cao hơn này, ý tưởng về lựa chọn hành động UCB thường không thực tế.

Problem 12 (UCB Spikes)**.** *In Fig. 2.4, UCB algorithm shows a distinct spike in performance on 11th step. Why is this? Note: for your answer to be fully satisfactory it must explain both why reward increases on 11th step & why it decreases on subsequent steps. If $c = 1$, then spike is less prominent.*

– Trong Hình 2.4, thuật toán UCB cho thấy hiệu suất tăng đột biến rõ rệt ở bước thứ 11. Tại sao vậy? Lưu ý: để câu trả lời của bạn được coi là hoàn toàn thỏa đáng, bạn phải giải thích được cả lý do tại sao phần thưởng tăng ở bước thứ 11 & tại sao nó giảm ở các bước tiếp theo. Nếu $c = 1$, thì sự tăng đột biến sẽ ít rõ rệt hơn.

◦ **2.8. Gradient Bandit Algorithms.** So far in this chap, have considered methods that estimate action values & use those estimates to select actions. This is often a good approach, but not the only 1 possible. In this sect, consider learning a

numerical *preference* for each action a, which denote $H_t(a) \in \mathbb{R}$. Larger preference, more often that action is taken, but preference has no interpretation in terms of reward. Only relative preference of 1 action over another is important; if add 1000 to all action preferences there is no effect on action probabilities, which are determined according to a *soft-max distribution* (i.e., Gibbs or Boltzmann distribution) as follows: (2.11)

$$\Pr\{A_t = a\} \coloneqq \frac{e^{H-t(a)}}{\sum_{b=1}^{k} e^{H_t(b)}} \coloneqq \pi_t(a),$$

where here have also introduced a useful new notation, $\pi_t(a)$, for probability of taking action a at time t. Initially all action preferences are same (e.g., $H_1(a) = 0$, $\forall a$) so that all actions have an equal probability of being selected.

– Thuật toán Gradient Bandit. Cho đến nay trong chương này, chúng ta đã xem xét các phương pháp ước tính giá trị hành động & sử dụng các ước tính đó để chọn hành động. Đây thường là 1 cách tiếp cận tốt, nhưng không phải là cách tiếp cận duy nhất. Trong phần này, hãy xem xét việc học 1 *preference* dạng số cho mỗi hành động a, biểu thị $H_t(a) \in \mathbb{R}$. Ưu tiên càng lớn, hành động đó càng được thực hiện thường xuyên, nhưng ưu tiên không được diễn giải theo nghĩa phần thưởng. Chỉ có sự ưu tiên tương đối của 1 hành động so với hành động khác là quan trọng; nếu cộng 1000 vào tất cả các sở thích hành động thì không ảnh hưởng đến xác suất hành động, được xác định theo phân phối *soft-max* (i.e., phân phối Gibbs hoặc Boltzmann) như sau: (2.11)

$$\Pr\{A_t = a\} \coloneqq \frac{e^{H-t(a)}}{\sum_{b=1}^{k} e^{H_t(b)}} \coloneqq \pi_t(a),$$

trong đó ở đây cũng đã giới thiệu 1 ký hiệu mới hữu ích, $\pi_t(a)$, cho xác suất thực hiện hành động a tại thời điểm t. Ban đầu, tất cả các sở thích hành động đều giống nhau (ví dụ: $H_1(a) = 0$, $\forall a$) nên tất cả các hành động đều có xác suất được chọn bằng nhau.

Problem 13. *Show: in case of 2 actions, soft-max distribution is same as that given by logistic, or sigmoid, function often used in statistics & ANNs.*

– Trong trường hợp có 2 hành động, phân phối soft-max giống với phân phối được đưa ra bởi hàm logistic hoặc hàm sigmoid thường được sử dụng trong thống kê & ANN.

There is a natural learning algorithm for soft-max action preferences based on idea of stochastic gradient ascent. On each step, after selecting action A_t & receiving reward R_t, action preferences are updated by (2.12)

$$\begin{aligned} H_{t+1}(A_t) &\coloneqq H_t(A_t) + \alpha(R_t - \overline{R}_t)(1 - \pi_t(A_t)), \\ H_{t+1}(a) &\coloneqq H_t(a) - \alpha(R_t - \overline{R}_t)\pi_t(a), \ \forall a \neq A_t, \end{aligned}$$

where $\alpha > 0$ is a step-size parameter, & $\overline{R}_t \in \mathbb{R}$ is average of rewards up to but not including time t (with $\overline{R}_1 \coloneqq R_1$), which can be computed incrementally as described in Sect. 2.4 (or Sect. 2.5 if problem is nonstationary). [In empirical results in this chap, baseline $\overline{R}_t$ also included R_t.] $\overline{R}_t$ term serves as a baseline with which reward is compared. If reward is higher than baseline, then probability of taking A_t in future is increased, & if reward is below baseline, then probability is decreased. Non-selected actions move in opposite direction.

– Có 1 thuật toán học tự nhiên cho các tùy chọn hành động soft-max dựa trên ý tưởng về sự tăng dần của độ dốc ngẫu nhiên. Ở mỗi bước, sau khi chọn hành động A_t & nhận phần thưởng R_t, các tùy chọn hành động được cập nhật theo (2.12)

$$\begin{aligned} H_{t+1}(A_t) &\coloneqq H_t(A_t) + \alpha(R_t - \overline{R}_t)(1 - \pi_t(A_t)), \\ H_{t+1}(a) &\coloneqq H_t(a) - \alpha(R_t - \overline{R}_t)\pi_t(a), \ \forall a \neq A_t, \end{aligned}$$

trong đó $\alpha > 0$ là tham số kích thước bước, & $\overline{R}_t \in \mathbb{R}$ là giá trị trung bình của phần thưởng lên đến nhưng không bao gồm thời gian t (với $\overline{R}_1 \coloneqq R_1$), có thể được tính toán gia tăng như mô tả trong Phần. 2.4 (hoặc Mục 2.5 nếu bài toán không dừng). [Trong các kết quả thực nghiệm trong chương này, đường cơ sở $\overline{R}_t$ cũng bao gồm R_t.] Số hạng $\overline{R}_t$ đóng vai trò là đường cơ sở để so sánh phần thưởng. Nếu phần thưởng cao hơn đường cơ sở, thì xác suất nhận được A_t trong tương lai sẽ tăng lên, & nếu phần thưởng thấp hơn đường cơ sở, thì xác suất giảm xuống. Các hành động không được chọn sẽ di chuyển theo hướng ngược lại.

Fig. 2.5: Average performance of gradient bandit algorithm with & without a reward baseline on 10-armed testbed when $q_*(a)$ are chosen to be near +4 rather than near 0. shows results with gradient bandit algorithm on a variant of 10-armed testbed in which true expected rewards were selected according to a normal distribution with a mean of +4 instead of 0 (& with unit variance as before). This shifting up of all rewards has absolutely no effect on gradient bandit algorithm because of reward baseline term, which instantaneously adapts to new level. But if baseline were omitted (i.e., if $\overline{R}_t$ was taken to constant 0 in (2.12)), then performance would be significantly degraded, as shown in figure.

– Hình 2.5: Hiệu suất trung bình của thuật toán gradient bandit với & không có đường cơ sở phần thưởng trên nền tảng thử nghiệm 10 cánh tay khi $q_*(a)$ được chọn gần +4 thay vì gần 0. cho thấy kết quả với thuật toán gradient bandit trên 1 biến thể của nền tảng thử nghiệm 10 cánh tay trong đó phần thưởng kỳ vọng thực sự được chọn theo phân phối chuẩn với giá trị trung bình là +4 thay vì 0 (& với phương sai đơn vị như trước). Sự dịch chuyển lên của tất cả các phần thưởng này hoàn toàn không ảnh hưởng đến thuật toán gradient bandit do phần cơ sở phần thưởng, điều này ngay lập tức thích ứng với mức mới. Nhưng nếu đường cơ sở bị bỏ qua (i.e., nếu $\overline{R}_t$ được đưa đến hằng số 0 trong (2.12)), thì hiệu suất sẽ giảm đáng kể, như thể hiện trong hình.

Example 4 (Bandit Gradient Algorithm as Stochastic Gradient Ascent)**.** *One can gain a deeper insight into gradient bandit algorithm by understanding it as a stochastic approximation to gradient ascent. In exact* gradient ascent*, each action preference $H_t(a)$ would be incremented in proportion to increment's effect on performance: (2.13)*

$$H_{t+1}(a) \coloneqq H_t(a) + \alpha \frac{\partial \mathbb{E}[R_t]}{\partial H_t(a)},$$

where measure of performance here is expected reward:

$$\mathbb{E}[R_t] = \sum_x \pi_t(x) q_*(x),$$

& measure of increment's effect is partial derivative *of this performance measure w.r.t. action preference.*

- **2.9. Associative Search (Contextual Bandits).**

- **3. Finite Markov Decision Processes.**
- **4. Dynamic Programming.**
- **5. Monte Carlo Methods.**
- **6. Temporal-Difference Learning.**
- **7. n-step Boostrapping.**
- **8. Planning & Learning with Tabular Methods.**

  PART II: APPROXIMATE SOLUTION METHODS.
- **9. On-policy Prediction with Approximation.**
- **10. On-policy Control with Approximation.**
- **11. Off-policy Methods with Approximation.**
- **12. Eligibility Traces.**
- **13. Policy Gradient Methods.**

  PART III: LOOKING DEEPER.
- **14. Psychology.**
- **15. Neuroscience.**
- **16. Applications & Case Studies.**
- **17. Frontiers.**

2 Reinforcement Learning for Optimal Job Scheduling

2.1 Xinquan Wu, Xuefeng Yan, Mingqiang Wei, Donghai Guan. An Efficient Deep Reinforcement Learning Environment for Flexible Job-Shop Scheduling. Sep 10, 2025

- **Abstract.** Flexible Job-shop Scheduling Problem (FJSP) is a classical combinatorial optimization problem that has a wide-range of applications in real world. In order to generate fast & accurate scheduling solutions for FJSP, various deep reinforcement learning (DRL) scheduling methods have been developed. However, these methods are mainly focused on design of DRL scheduling Agent, overlooking modeling of DRL environment. This paper presents a simple chronological DRL environment for FJSP based on discrete event simulation & an end-to-end DRL scheduling model is proposed based on proximal policy optimization (PPO). Furthermore, a short novel state representation of FJSP is proposed based on 2 state variables in scheduling environment & a novel comprehensive reward function is designed based on scheduling area of machines. Experimental results on public benchmark instances show: performance of simple priority dispatching rules (PDR) is improved in our scheduling environment & our DRL scheduling model obtains competing performance compared with OR-Tools, meta-heuristic, DRL & PDR scheduling methods.

  – Bài toán Lập lịch Xưởng Linh hoạt (FJSP) là 1 bài toán tối ưu hóa tổ hợp cổ điển có phạm vi ứng dụng rộng rãi trong thực tế. Để tạo ra các giải pháp lập lịch nhanh & chính xác cho FJSP, nhiều phương pháp lập lịch học tăng cường sâu (DRL) đã được phát triển. Tuy nhiên, các phương pháp này chủ yếu tập trung vào thiết kế Tác nhân lập lịch DRL, bỏ qua việc mô hình hóa môi trường DRL. Bài báo này trình bày 1 môi trường DRL theo trình tự thời gian đơn giản cho FJSP dựa trên mô phỏng sự kiện rời rạc & 1 mô hình lập lịch DRL đầu cuối được đề xuất dựa trên tối ưu hóa chính sách gần đúng (PPO). Hơn nữa, 1 biểu diễn trạng thái mới ngắn gọn của FJSP được đề xuất dựa trên 2 biến trạng thái trong môi trường lập lịch & 1 hàm

thưởng toàn diện mới được thiết kế dựa trên vùng lập lịch của máy. Kết quả thử nghiệm trên các trường hợp chuẩn công khai cho thấy: hiệu suất của các quy tắc phân bổ ưu tiên đơn giản (PDR) được cải thiện trong môi trường lập lịch của chúng tôi & mô hình lập lịch DRL của chúng tôi đạt được hiệu suất cạnh tranh so với các phương pháp lập lịch OR-Tools, meta-heuristic, DRL & PDR.

- **1. Introduction.** Job shop scheduling problem (JSSP) is a classical NP-hard combinatorial optimization problem. It has been studied for decades & has been applied in a wide-range of areas including semiconductor, machine manufacturing, metallurgy, automobile manufacturing, supply chain & other fields [25]. Flexible Job-shop Scheduling Problem (FJSP) is a generalization of JSSP, which allows each operation to be processed on multiple candidate machines [6]. This makes it a more challenging problem with more complex topology & larger solution space.

  – Bài toán lập lịch xưởng gia công (JSSP) là 1 bài toán tối ưu tổ hợp NP-khó kinh điển. Nó đã được nghiên cứu trong nhiều thập kỷ & đã được ứng dụng trong nhiều lĩnh vực bao gồm bán dẫn, chế tạo máy, luyện kim, sản xuất ô tô, chuỗi cung ứng & các lĩnh vực khác [25]. Bài toán lập lịch xưởng gia công linh hoạt (FJSP) là 1 dạng tổng quát của JSSP, cho phép mỗi thao tác được xử lý trên nhiều máy ứng viên [6]. Điều này làm cho nó trở thành 1 bài toán khó hơn với cấu trúc phức tạp hơn & không gian nghiệm lớn hơn.

  Extensive researches have been widely explored to solve FJSP in recent years, including exact, heuristic, meta-heuristic, hyper-heuristic & RL methods. Exact approaches e.g. mixed integer linear programming (MILP) [20] may suffer from curse of dimension. So it is intractable to find exact scheduling solutions in a given time limit. Heuristic dispatching rules are widely used in real world manufacturing due to their simple & fast nature. Simple priority dispatching rules (PDR)[21], e.g. most work remaining (MWKR) possess advantages of high flexibility & easy implementation, but performance of PDR methods is not stable for different optimization objectives. Hyper-heuristic methods [29] e.g. genetic programming-based hyper-heuristic (GPHH), provide a way of automatically designing dispatching rules [28]. However, GP-evolved rules often has a large number of features in terminal set, making it difficult to identify promising search areas & determine optimal features.

  – Các nghiên cứu sâu rộng đã được khai thác rộng rãi để giải quyết FJSP trong những năm gần đây, bao gồm các phương pháp chính xác, heuristic, meta-heuristic, hyper-heuristic & RL. Các phương pháp chính xác, ví dụ như lập trình tuyến tính số nguyên hỗn hợp (MILP) [20] có thể bị lời nguyền về chiều. Do đó, việc tìm ra các giải pháp lập lịch chính xác trong 1 giới hạn thời gian nhất định là khó khăn. Các quy tắc điều phối heuristic được sử dụng rộng rãi trong sản xuất thực tế do bản chất đơn giản & nhanh chóng của chúng. Các quy tắc điều phối ưu tiên đơn giản (PDR) [21], ví dụ như hầu hết công việc còn lại (MWKR) có ưu điểm là tính linh hoạt cao & dễ triển khai, nhưng hiệu suất của các phương pháp PDR không ổn định đối với các mục tiêu tối ưu khác nhau. Các phương pháp hyper-heuristic [29], ví dụ như hyper-heuristic dựa trên lập trình di truyền (GPHH), cung cấp 1 cách để tự động thiết kế các quy tắc điều phối [28]. Tuy nhiên, các quy tắc phát triển GP thường có 1 số lượng lớn các tính năng trong tập đầu cuối, khiến việc xác định các khu vực tìm kiếm đầy hứa hẹn & xác định các tính năng tối ưu trở nên khó khăn.

  Different from PDR, meta-heuristic methods e.g. Tabu Search [12], Genetic Algorithm [19], Differential Evolution [7] & Particle Swarm Optimization [1], can produce higher solution quality than PDR due to introduction of iterative, randomized search strategies. Nevertheless, these methods also have shortcomings e.g. slow convergence speed, difficulty in obtaining global optimal solutions for large-scale problems & are difficult to apply to dynamic scenarios that need real-time decisions.

  – Khác với PDR, các phương pháp siêu heuristic như Tìm kiếm Tabu [12], Thuật toán Di truyền [19], Tiến hóa Vi phân [7] & Tối ưu hóa Bầy đàn Hạt [1] có thể tạo ra chất lượng giải pháp cao hơn PDR nhờ áp dụng các chiến lược tìm kiếm lặp lại, ngẫu nhiên. Tuy nhiên, các phương pháp này cũng có những hạn chế, ví dụ như tốc độ hội tụ chậm, khó đạt được giải pháp tối ưu toàn cục cho các bài toán quy mô lớn & khó áp dụng cho các tình huống động cần quyết định theo thời gian thực.

  RL is 1 of most important branches of ML & attracts interests of researchers from numerous fields especially for scheduling. Early RL scheduling methods represent scheduling policies using arrays or tables [2], which are only suitable for small-scale scheduling problems because of curse of dimension. However, scheduling tasks in real world usually have a higher dimensional state space & large action space, which limits applications of RL. Since deep neural networks have a strong fitting capability to process high-dimensional data, deep reinforcement learning (DRL) has been used to solve scheduling tasks with large or continuous state space & shows great potential to solve various scheduling problems. In DRL scheduling methods, scheduling policy is usually designed based on convolutional neural network (CNN) in [9], multi-layer perception (MLP) in [9, 10], RNN in [14], GNN in [15, 22, 26], attention networks in [23] & other deep neural networks in [16]. Scheduling agent is often trained by RL algorithms e.g. deep Q-learning (DQN), proximal policy optimization (PPO), Deep Deterministic Policy Gradient (DDPG). However, above DRL scheduling methods obtained low accurate solutions & some are even inferior to simple PDR. Even though recent GMAS model [13] whose policy is designed based on Graph Convolutional Network (GCN), obtained SOTA results on standard benchmark instances, it is a decentralized Multi-agent rRL (MARL) model. Besides, current DRL scheduling methods focus mainly on design of scheduling agents, overlooking modeling of scheduling environment which is of vital important for improving performance of DRL scheduling methods.

  – RL là 1 trong những nhánh quan trọng nhất của ML & thu hút sự quan tâm của các nhà nghiên cứu từ nhiều lĩnh vực, đặc biệt là đối với việc lập lịch. Các phương pháp lập lịch RL ban đầu biểu diễn các chính sách lập lịch sử dụng mảng hoặc bảng [2], chỉ phù hợp với các vấn đề lập lịch quy mô nhỏ do giới hạn chiều. Tuy nhiên, các tác vụ lập lịch trong thế giới thực thường có không gian trạng thái nhiều chiều & không gian hành động lớn, điều này hạn chế các ứng dụng của RL. Vì mạng nơ-ron sâu có khả năng khớp mạnh để xử lý dữ liệu nhiều chiều, nên học tăng cường sâu (DRL) đã được sử dụng để giải quyết các tác vụ lập lịch với không gian trạng thái lớn hoặc liên tục & cho thấy tiềm năng to lớn trong việc giải quyết nhiều vấn đề lập lịch khác nhau. Trong các phương pháp lập lịch DRL, chính sách lập lịch thường được thiết kế dựa trên mạng nơ-ron

tích chập (CNN) trong [9], nhận thức đa lớp (MLP) trong [9, 10], RNN trong [14], GNN trong [15, 22, 26], mạng chú ý trong [23] & các mạng nơ-ron sâu khác trong [16]. Tác nhân lập lịch thường được đào tạo bằng các thuật toán RL, ví dụ: Học sâu Q (DQN), tối ưu hóa chính sách gần đúng (PPO), Gradient Chính sách Xác định Sâu (DDPG). Tuy nhiên, các phương pháp lập lịch DRL trên cho kết quả độ chính xác thấp & 1 số thậm chí còn kém hơn PDR đơn giản. Mặc dù mô hình GMAS gần đây [13], với chính sách được thiết kế dựa trên Mạng Tích chập Đồ thị (GCN), đã đạt được kết quả SOTA trên các trường hợp chuẩn mực, nhưng nó là 1 mô hình rRL Đa tác tử phi tập trung (MARL). Bên cạnh đó, các phương pháp lập lịch DRL hiện tại chủ yếu tập trung vào thiết kế các tác nhân lập lịch, bỏ qua việc mô hình hóa môi trường lập lịch, vốn rất quan trọng để cải thiện hiệu suất của các phương pháp lập lịch DRL.

There are mainly 4 modeling methods for FJSP: mathematical [20], Petri Nets (PN)-based [17], Disjunctive Graph (DG)-based [3] & simulation-based modeling [24]. Mathematical modeling methods usually formulates FJSP as a MILP & formulated problem is then be solved by optimization methods e.g. Genetic Algorithm. PN is a versatile modeling tool that can be used to model scheduling problems as they can model parallel activities, resource sharing & synchronization. Both modeling methods are not suitable for DRL scheduling methods. DG is another alternative to model FJSP which integrates machine status & operation dependencies, & provides critical structural information for scheduling decisions. However, DG fails to incorporate dynamically changing state information of FJSP, requiring extra handcrafted node features. Besides, allocation order provided by DG model is not strictly chronological. Simulation-based modeling method: develop algorithms to simulate laws of activities in real world. In current simulation environment of DRL scheduling methods, scheduling process is promoted by events in candidate queue, ignoring occurrence temporal order of these events.

– Có chủ yếu 4 phương pháp mô hình hóa cho FJSP: toán học [20], dựa trên Petri Nets (PN) [17], dựa trên Disjunctive Graph (DG) [3] & mô hình hóa dựa trên mô phỏng [24]. Các phương pháp mô hình hóa toán học thường xây dựng FJSP như 1 bài toán MILP & sau đó được giải quyết bằng các phương pháp tối ưu hóa, ví dụ như thuật toán di truyền. PN là 1 công cụ mô hình hóa đa năng có thể được sử dụng để mô hình hóa các vấn đề lập lịch vì chúng có thể mô hình hóa các hoạt động song song, chia sẻ tài nguyên & đồng bộ hóa. Cả 2 phương pháp mô hình hóa đều không phù hợp với các phương pháp lập lịch DRL. DG là 1 giải pháp thay thế khác cho mô hình FJSP tích hợp trạng thái máy & phụ thuộc hoạt động, & cung cấp thông tin cấu trúc quan trọng cho các quyết định lập lịch. Tuy nhiên, DG không kết hợp thông tin trạng thái thay đổi động của FJSP, yêu cầu các tính năng nút thủ công bổ sung. Bên cạnh đó, thứ tự phân bổ do mô hình DG cung cấp không hoàn toàn theo thứ tự thời gian. Phương pháp mô hình hóa dựa trên mô phỏng: phát triển các thuật toán để mô phỏng các quy luật hoạt động trong thế giới thực. Trong môi trường mô phỏng hiện tại của các phương pháp lập lịch DRL, quá trình lập lịch được thúc đẩy bởi các sự kiện trong hàng đợi ứng viên, bỏ qua thứ tự thời gian xảy ra của các sự kiện này.

In this paper, a chronological DRL scheduling environment for FJSP is presented based on discrete event simulation algorithm where at each decision step, accurate state changes of scheduling process are recorded by state variables & a novel comprehensive reward function is designed based on scheduling area. Furthermore, an end to end DRL model for FJSP is proposed based on actor-critic PPO. Specially, a very short state representation is expressed by 2 state variables which are derived directly from necessary variables in environment simulation algorithm & action space is constructed using 6 PDRs from literature. Besides, in order to reduce computation time for RL agent, a simple scheduling policy is designed based on MLP with only 1 hidden layer & Softmax function.

– Bài báo này trình bày 1 môi trường lập lịch DRL theo thời gian cho FJSP dựa trên thuật toán mô phỏng sự kiện rời rạc, trong đó tại mỗi bước quyết định, các thay đổi trạng thái chính xác của quy trình lập lịch được ghi lại bởi các biến trạng thái & 1 hàm thưởng toàn diện mới được thiết kế dựa trên vùng lập lịch. Hơn nữa, 1 mô hình DRL đầu cuối cho FJSP được đề xuất dựa trên PPO tác nhân-phê bình. Cụ thể, 1 biểu diễn trạng thái rất ngắn được thể hiện bằng 2 biến trạng thái được suy ra trực tiếp từ các biến cần thiết trong thuật toán mô phỏng môi trường & không gian hành động được xây dựng bằng cách sử dụng 6 PDR từ tài liệu. Bên cạnh đó, để giảm thời gian tính toán cho tác nhân RL, 1 chính sách lập lịch đơn giản được thiết kế dựa trên MLP với chỉ 1 lớp ẩn & hàm Softmax.

Contributions of this work are listed as follows.

1. A basic simulation algorithm based on chronological discrete event is designed for FJSP, providing a general environment for DRL scheduling methods.
2. A simple DRL model for FJSP is proposed based on PPO where a very short state representation for FJSP is presented, avoiding handcrafted state features & massive experiments for feature selection & a novel comprehensive reward function is proposed based on scheduling area.
3. Experimental results show: performance of single PDR is improved in our environment, even better than some DRL scheduling methods & proposed DRL scheduling model obtains competing performance compared with OR-Tools, meta-heuristic, DRL & PDR scheduling methods.

– Những đóng góp của công trình này được liệt kê như sau.

1. 1 thuật toán mô phỏng cơ bản dựa trên sự kiện rời rạc theo trình tự thời gian được thiết kế cho FJSP, cung cấp 1 môi trường chung cho các phương pháp lập lịch DRL.
2. 1 mô hình DRL đơn giản cho FJSP được đề xuất dựa trên PPO, trong đó 1 biểu diễn trạng thái rất ngắn cho FJSP được trình bày, tránh các đặc trưng trạng thái thủ công & các thử nghiệm hàng loạt để lựa chọn đặc trưng & 1 hàm thưởng toàn diện mới được đề xuất dựa trên khu vực lập lịch.

3. Kết quả thử nghiệm cho thấy: hiệu suất của PDR đơn lẻ được cải thiện trong môi trường của chúng tôi, thậm chí còn tốt hơn 1 số phương pháp lập lịch DRL & mô hình lập lịch DRL được đề xuất đạt được hiệu suất cạnh tranh so với các phương pháp lập lịch OR-Tools, meta-heuristic, DRL & PDR.

Rest of paper is organized as follows. A chronological discrete event simulation-based scheduling environment for FJSP is introduced in Sect. II. In Sect. III, proposed DRL scheduling model for FJSP are constructed. Sect. IV demonstrates detailed experiments on standard benchmarks with different sizes & results are compared & discussed, while conclusions & future work reside in Sect. V.

– Phần còn lại của bài báo được tổ chức như sau. 1 môi trường lập lịch dựa trên mô phỏng sự kiện rời rạc theo thời gian cho FJSP được giới thiệu trong Phần II. Trong Phần III, mô hình lập lịch DRL đề xuất cho FJSP đã được xây dựng. Phần IV trình bày các thí nghiệm chi tiết trên các chuẩn mực chuẩn với các kích thước khác nhau & kết quả được so sánh & thảo luận, trong khi kết luận & các nghiên cứu trong tương lai nằm trong Phần V.

- **2. Simulation environment for FJSP.** In this sect, 1st introduced basics of FJSPs. Then defined storage structure of FJSP & rules of state changing when scheduling. Finally, detailed simulation algorithm for FJSP is presented.

  – Môi trường mô phỏng cho FJSP. Trong phần này, trước tiên chúng tôi giới thiệu những kiến thức cơ bản về FJSP. Sau đó, chúng tôi định nghĩa cấu trúc lưu trữ của FJSP & các quy tắc thay đổi trạng thái khi lập lịch. Cuối cùng, thuật toán mô phỏng chi tiết cho FJSP được trình bày.

  - **2.1. Flexible Job-shop Scheduling Problems.** FJSP differs from JSSP in that it allows an operation to be processed by any machine from a given set & different machines may have different processing times. Problem: assign each operation to eligible machine s.t. maximal completion time (makespan) of all jobs is minimized. It can be presented as $FJ||Cmax$ by 3-field notations [6].

    – FJSP khác với JSSP ở chỗ nó cho phép 1 thao tác được xử lý bởi bất kỳ máy nào trong 1 tập hợp nhất định & các máy khác nhau có thể có thời gian xử lý khác nhau. Vấn đề: gán mỗi thao tác cho máy đủ điều kiện, thời gian hoàn thành tối đa (makespan) của tất cả các công việc được giảm thiểu. Nó có thể được biểu diễn dưới dạng $FJ||Cmax$ theo ký hiệu 3 trường [6].

    In public benchmark of FJSP, environment information is passed through an instance file. As is depicted in Fig. 1: An example of flexible job-shop scheduling problem (MK1 [4]), 1st line in instance file consists of 3 integers representing number of jobs, number of machines & average number of machines per operation (optional), resp. Each of following lines describes information of a job where 1st integer is number of operations & followed integers depicts operation information. Operation information includes 2 integers: 1st represents index of a machine & 2nd is processing time of operation on this machine. Processing operation order in a job is advanced from left to right.

    – Trong chuẩn mực công khai của FJSP, thông tin môi trường được truyền qua 1 tệp thể hiện. Như được mô tả trong Hình 1: Ví dụ về bài toán lập lịch xưởng công việc linh hoạt (MK1 [4]), dòng đầu tiên trong tệp thể hiện bao gồm 3 số nguyên biểu diễn số lượng công việc, số lượng máy & số lượng máy trung bình trên mỗi thao tác (tùy chọn), tương ứng. Mỗi dòng sau mô tả thông tin của 1 công việc, trong đó số nguyên đầu tiên là số lượng thao tác & các số nguyên tiếp theo mô tả thông tin thao tác. Thông tin thao tác bao gồm 2 số nguyên: số 1 biểu diễn chỉ số của 1 máy & số 2 là thời gian xử lý thao tác trên máy này. Thứ tự thao tác trong 1 công việc được sắp xếp theo thứ tự từ trái sang phải.

  - **2.2. Data structure for FJSP.** In this paper, based on benchmark instance file, propose a new & efficient storage structure for FJSP instances. Scheduling information is represented by a 2D table where job information is described in rows & column records processing stage information. Each element in this table includes 2 sets: machine set & remaining processing time set. Fig. 2: Storage structure of FJSP instance. demonstrates an example of a FJSP instance where storage structure is marked with a red box. It is efficient to retrieve any operation information using job index & processing stage index.

    – Cấu trúc dữ liệu cho FJSP. Trong bài báo này, dựa trên tệp phiên bản chuẩn, chúng tôi đề xuất 1 cấu trúc lưu trữ mới & hiệu quả cho các phiên bản FJSP. Thông tin lập lịch được biểu diễn bằng 1 bảng 2D, trong đó thông tin công việc được mô tả theo hàng & cột ghi lại thông tin giai đoạn xử lý. Mỗi phần tử trong bảng này bao gồm 2 tập hợp: tập máy & tập thời gian xử lý còn lại. Hình 2: Cấu trúc lưu trữ của phiên bản FJSP. minh họa 1 ví dụ về phiên bản FJSP, trong đó cấu trúc lưu trữ được đánh dấu bằng hộp đỏ. Việc truy xuất bất kỳ thông tin thao tác nào bằng chỉ mục công việc & chỉ mục giai đoạn xử lý đều hiệu quả.

  - **2.3. Rules of state updating.** In order to accurately record state changes of each scheduling step, 4 rules of state updating are defined based on type of operations. Job operations are divided into 4 categories: operations on processing, waiting operations without predecessors, completed operations & operations whose predecessors have not been completed. 4 rules are listed as follows & Fig. 3: Use of state update rules on a FJSP instance at time 4 provides an example to demonstrate use of these rules.

    – Quy tắc cập nhật trạng thái. Để ghi lại chính xác các thay đổi trạng thái của mỗi bước lập lịch, 4 quy tắc cập nhật trạng thái được xác định dựa trên loại thao tác. Các thao tác công việc được chia thành 4 loại: thao tác đang xử lý, thao tác chờ không có thao tác tiền nhiệm, thao tác đã hoàn thành & thao tác có thao tác tiền nhiệm chưa hoàn thành. 4 quy tắc được liệt kê như sau & Hình 3: Sử dụng quy tắc cập nhật trạng thái trên 1 thể hiện FJSP tại thời điểm 4 cung cấp 1 ví dụ để minh họa việc sử dụng các quy tắc này.

    * Rule 1: For an operation on processing, machine set has only 1 element which is negative index of selected machine while remaining time is recorded in remaining processing time set & its value decreases as time advances until completion of this operation.

– Quy tắc 1: Đối với 1 thao tác xử lý, tập hợp máy chỉ có 1 phần tử là chỉ số âm của máy được chọn trong khi thời gian còn lại được ghi vào tập hợp thời gian xử lý còn lại & giá trị của nó giảm dần theo thời gian cho đến khi hoàn thành thao tác này.

* Rule 2: For a waiting operation without predecessors whose needed machine is occupied, elements in machine set are all negative index of machines & elements in remaining processing time set stay unchanged. When needed machine is released, value of machine index is restored to be positive.
  – Quy tắc 2: Đối với 1 thao tác chờ không có tiền nhiệm mà máy cần thiết đang bị chiếm dụng, các phần tử trong tập máy đều có chỉ số máy âm & các phần tử trong tập thời gian xử lý còn lại không đổi. Khi máy cần thiết được giải phóng, giá trị chỉ số máy sẽ được khôi phục thành dương.
* Rule 3: For a completed operation, values in machine set & remaining processing time set equal to negative value of machine index & 0, resp.
  – Quy tắc 3: Đối với 1 hoạt động đã hoàn thành, các giá trị trong bộ máy & thời gian xử lý còn lại được đặt bằng giá trị âm của chỉ số máy & 0, tương ứng.
* Rule 4: For operations whose predecesors have not been proposed, values in machine set & remaining processing time set remain unchanged.
  – Quy tắc 4: Đối với các hoạt động mà hoạt động trước đó chưa được đề xuất, các giá trị trong tập hợp máy & thời gian xử lý còn lại vẫn không thay đổi.

This representation for FJSP instance can be recorded in a instance file & reloaded to new environment, i.e., it can be applied to nonzero or re-entrant scenarios.

– Biểu diễn này cho phiên bản FJSP có thể được ghi lại trong tệp phiên bản & tải lại vào môi trường mới, i.e., có thể áp dụng cho các trường hợp khác không hoặc có thể nhập lại.

- **2.4. Simulation algorithm for FJSP.** In this paper, proposed scheduling environment model is a simulation model based on chronological discrete events. There is a timer used to record current time & triggering time of events are recorded in a state variable. Once current time reaches triggering time of any events, this event will occur & corresponding event response program will be executed. Detailed process is illustrated in Algorithm 1: Simulation algorithm for FJSP.

  – Thuật toán mô phỏng cho FJSP. Trong bài báo này, mô hình môi trường lập lịch được đề xuất là 1 mô hình mô phỏng dựa trên các sự kiện rời rạc theo trình tự thời gian. Có 1 bộ đếm thời gian được sử dụng để ghi lại thời gian hiện tại & thời gian kích hoạt của các sự kiện được ghi lại trong 1 biến trạng thái. Khi thời gian hiện tại đạt đến thời gian kích hoạt của bất kỳ sự kiện nào, sự kiện này sẽ xảy ra & chương trình phản hồi sự kiện tương ứng sẽ được thực thi. Quy trình chi tiết được minh họa trong Thuật toán 1: Thuật toán mô phỏng cho FJSP.

  Proposed algorithm is mainly composed of 4 parts: selection of jobs & machines, state update of jobs & machines, time advance (3rd while statement) & machine release (4th while statement). In 1st part, decision action is divided into PDRs for job & machine selection & then specific job number & machine number are derived from PDRs. After allocating jobs & machines, their states are updated using state variables & table dictionaries. E.g., completion time of this job operation (or release time of machine) is recorded in `next_time_on_machine` & job-machine allocation information is recorded in `job_on_machine`. Once a job is allocated on a machine, state of this job is `assignable_job` changes to 0 & states of jobs which need that machine, are updated. If candidate machines of a job are all occupied by other jobs this job will become not assignable.

  – Thuật toán đề xuất chủ yếu bao gồm 4 phần: lựa chọn công việc & máy, cập nhật trạng thái của công việc & máy, tiến độ thời gian (câu lệnh while thứ 3) & giải phóng máy (câu lệnh while thứ 4). Trong phần 1, hành động quyết định được chia thành các PDR để lựa chọn công việc & máy & sau đó số công việc cụ thể & số máy được lấy từ PDR. Sau khi phân bổ công việc & máy, trạng thái của chúng được cập nhật bằng các biến trạng thái & từ điển bảng. Ví dụ: thời gian hoàn thành của hoạt động công việc này (hoặc thời gian giải phóng máy) được ghi lại trong `next_time_on_machine` & thông tin phân bổ công việc-máy được ghi lại trong `job_on_machine`. Sau khi 1 công việc được phân bổ trên 1 máy, trạng thái của công việc này là `assignable_job` thay đổi thành 0 & trạng thái của các công việc cần máy đó được cập nhật. Nếu tất cả các máy ứng viên của 1 công việc đều bị các công việc khác chiếm giữ thì công việc này sẽ không thể gán được.

  When there are no assignable jobs, time advance stage is coming. In this part, current time is 1st updated based on values in `next_time_on_machine` (if current time is smaller than any value in `next_time_on_machine`, take smallest value, otherwise take next smallest of them). Length of time step that needs to advance is then calculated based on `next_time_on_machine` & current time. Finally, next time of machines whose next time is slower than current time is advanced to current time. Machine release part releases machines whose release time reaches current time & updates status of jobs & machines. When current job operation is completed, occupied machine becomes idle & current operation of this job move to next operation. States of jobs & machines are updated in `assignable_job, job_on_machine`. If all operation of current job is completed, this job become not assignable & if machine needed for next operation is occupied, this job is still not assignable.

  – Khi không có công việc nào có thể gán được, giai đoạn tiến độ thời gian sẽ diễn ra. Trong phần này, thời gian hiện tại được cập nhật lần đầu tiên dựa trên các giá trị trong `next_time_on_machine` (nếu thời gian hiện tại nhỏ hơn bất kỳ giá trị nào trong `next_time_on_machine`, hãy lấy giá trị nhỏ nhất, nếu không thì lấy giá trị nhỏ tiếp theo trong số chúng). Sau đó, độ dài bước thời gian cần tiến lên được tính toán dựa trên `next_time_on_machine` & thời gian hiện tại. Cuối cùng, thời gian tiếp theo của các máy có thời gian tiếp theo chậm hơn thời gian hiện tại sẽ được tiến lên thời gian hiện tại. Bộ phận nhả máy sẽ nhả các máy có thời gian nhả đạt đến thời gian hiện tại & cập nhật trạng thái của công việc & máy. Khi thao tác công việc hiện tại hoàn thành, máy đang được chiếm dụng sẽ trở nên nhàn rỗi & thao tác hiện tại của công việc này chuyển sang thao tác tiếp theo. Trạng thái của công việc & máy được cập nhật trong `assignable_job, job_on_machine`.

Nếu tất cả thao tác của công việc hiện tại đã hoàn thành, công việc này sẽ không thể gán được & nếu máy cần cho thao tác tiếp theo bị chiếm dụng, công việc này vẫn không thể gán được.

- **3. A DRL scheduling framework for FJSP.** In this sect, introduce overall DRL framework for FJSP, illustrated in Fig. 4: DRl framework for flexible job-shop scheduling problems. It is composed of:

1. state representation based on 2 state variables
2. PDR action space
3. reward function based on scheduling area
4. scheduling policy based on deep neural networks & a Softmax function
5. PPO agent with an actor-critic learning architecture.

– Khung lập lịch DRL cho FJSP. Trong phần này, giới thiệu khung DRL tổng thể cho FJSP, được minh họa trong Hình 4: Khung DRL cho các bài toán lập lịch xưởng linh hoạt. Khung này bao gồm:

1. biểu diễn trạng thái của mục dựa trên 2 biến trạng thái
2. Không gian hành động PDR
3. Hàm thưởng dựa trên vùng lập lịch
4. Chính sách lập lịch dựa trên mạng nơ-ron sâu & 1 hàm Softmax
5. Tác tử PPO với kiến trúc học tập tác nhân-phê bình.

- **3.1. State representation based on state variables.** State representation depicts features of scheduling environment & determines size of state space, which is very crucial in RL scheduling methods. Various state representations for FJSP are presented in literature & are mainly focused on disjunctive graph [15, 22], feature matrix [24] & handcrafted state variables [8, 10, 14]. However, whether it is node features of disjunctive graph or matrix, or variable features, state features of job shop are mainly manually designed, which requires a large amount of processional domain knowledge, computing resources & time.

  – Biểu diễn trạng thái dựa trên biến trạng thái. Biểu diễn trạng thái mô tả các đặc điểm của môi trường lập lịch & xác định kích thước của không gian trạng thái, điều này rất quan trọng trong các phương pháp lập lịch RL. Nhiều biểu diễn trạng thái cho FJSP được trình bày trong tài liệu & chủ yếu tập trung vào đồ thị rời rạc [15, 22], ma trận đặc trưng [24] & các biến trạng thái thủ công [8, 10, 14]. Tuy nhiên, cho dù là đặc trưng nút của đồ thị rời rạc hay ma trận, hay đặc trưng biến, đặc trưng trạng thái của job shop chủ yếu được thiết kế thủ công, đòi hỏi lượng lớn kiến thức về miền xử lý, tài nguyên tính toán & thời gian.

  In this paper, a novel short state representation is proposed to reduce feature redundancy & to avoid handcrafted feature design & feature selection, which requires less computation time of environment state variables & scheduling policy networks. 2 state variables: `assignable_job, completed_op_of_job` are selected from Algorithm 1 as state features of job shop scheduling environment. Variable `assignable_job` is a Boolean vector to represent whether a job can be allocated or not. `completed_op_of_job` represents number of completed operation of a job & this value is then scaled by maximum number of operations in jobs to be in range $[0, 1]$. Length of both variables equals number of jobs. Finally, in order to represent environment state, scaled variables are simply concatenated to a vector whose length equals 2 times of number of jobs.

  – Trong bài báo này, 1 biểu diễn trạng thái ngắn mới được đề xuất để giảm sự dư thừa tính năng & để tránh thiết kế tính năng thủ công & lựa chọn tính năng, đòi hỏi ít thời gian tính toán hơn của các biến trạng thái môi trường & mạng chính sách lập lịch. 2 biến trạng thái: `assignable_job, completed_op_of_job` được chọn từ Thuật toán 1 làm các tính năng trạng thái của môi trường lập lịch job shop. Biến `assignable_job` là 1 vectơ Boolean để biểu diễn liệu 1 công việc có thể được phân bổ hay không. `completed_op_of_job` biểu diễn số lượng hoạt động đã hoàn thành của 1 công việc & giá trị này sau đó được chia tỷ lệ theo số lượng hoạt động tối đa trong các công việc nằm trong phạm vi $[0, 1]$. Độ dài của cả 2 biến bằng số lượng công việc. Cuối cùng, để biểu diễn trạng thái môi trường, các biến được chia tỷ lệ chỉ cần được nối thành 1 vectơ có độ dài bằng 2 lần số lượng công việc.

  There are many advantages of our state representation:

  1. length of state features is much less than that in previous research which means less computation time of environment state variables & scheduling policy networks
  2. state features are unique in a scheduling solution, which suggests that state features can be easily distinguished by scheduling agent
  3. state features are directly derived from 2 state variables in Algorithm 1, avoiding massive experiments on feature design & selection.

  – Biểu diễn trạng thái của chúng tôi có nhiều ưu điểm:

  1. Độ dài của các đặc trưng trạng thái ngắn hơn nhiều so với nghiên cứu trước đây, đồng nghĩa với việc giảm thời gian tính toán các biến trạng thái môi trường & mạng lưới chính sách lập lịch.
  2. Các đặc trưng trạng thái của mục là duy nhất trong 1 giải pháp lập lịch, điều này cho thấy các đặc trưng trạng thái có thể dễ dàng được phân biệt bởi tác nhân lập lịch.

3. Các đặc trưng trạng thái của mục được suy ra trực tiếp từ 2 biến trạng thái trong Thuật toán 1, tránh được các thử nghiệm lớn về thiết kế & lựa chọn đặc trưng.

- **3.2. Action space based on PDR.** In DRl scheduling methods based on single agent, action is output of scheduling policy networks, which is usually an integer. Since FJSP needs to select a job & a machine at each decision, decompose this integer into 2 parts by dividing it by number of PDRs for machine selection where quotient is index of PDR for jobs assignment & remainder represents index of PDR for machine selection.

  – Không gian hành động dựa trên PDR. Trong các phương pháp lập lịch DRl dựa trên tác nhân đơn lẻ, hành động là đầu ra của mạng chính sách lập lịch, thường là 1 số nguyên. Vì FJSP cần chọn 1 công việc & 1 máy tại mỗi quyết định, hãy phân tích số nguyên này thành 2 phần bằng cách chia nó cho số PDR để chọn máy, trong đó thương là chỉ số của PDR để gán công việc & phần dư biểu thị chỉ số của PDR để chọn máy.

  In this paper, 6 PDRs are selected to construct action space for simplicity of implementation & ease of generalization. For selecting a job, 4 PDRs are selected directly from literature [27] including Shortest Processing (SPT), Most Work Remaining (MWRK), Most Operations Remaining (MOR) & Minimum ratio of Flow Due Date to Most Work Remaining (FDD/MWRK). Longest Remaining Machine time not including current operation processing time (LRM) is selected from [11] due to its excellent performance. First In First Out (FIFO) is selected because it is widely used in various scheduling problems. In addition, action space for selecting machine given a job is composed of SPT & Longest Processing Time (LPT). So that total size of action space equals number of PDRs for selecting jobs multiplied by number of PDRs for selecting machines. Definitions of these PDR are listed as follows:

  * SPT: $\min Z_{ij} = p_{ij}$
  * LPT: $\max Z_{ij} = p_{ij}$
  * FDD/MWKR: $\min Z_{ij} = \frac{\sum_1^j p_{ij}}{\sum_j^{n_i} p_{ij}}$
  * MOR: $\max Z_{ij} = n_i - j + 1$
  * LRM: $\max Z_{ij} = \sum_{j+1}^{n_i} p_{ij}$
  * FIFO: $\max Z_{ij} = t - Rt_i$

  where Z_{ij} is priority index of operation O_{ij}. p_{ij} is processing time of operation O_{ij}. n_i: number of operations for job J_i. j: number of completed operations. t: current time & Rt_i: release time of J_i.

  – Trong bài báo này, 6 PDR được chọn để xây dựng không gian hành động nhằm đơn giản hóa việc triển khai & dễ dàng khái quát hóa. Để lựa chọn 1 công việc, 4 PDR được chọn trực tiếp từ tài liệu [27] bao gồm Xử lý ngắn nhất (SPT), Công việc còn lại nhiều nhất (MWRK), Hoạt động còn lại nhiều nhất (MOR) & Tỷ lệ tối thiểu giữa Ngày đến hạn luồng & Công việc còn lại nhiều nhất (FDD/MWRK). Thời gian máy còn lại dài nhất không bao gồm thời gian xử lý hoạt động hiện tại (LRM) được chọn từ [11] do hiệu suất tuyệt vời của nó. Vào trước ra trước (FIFO) được chọn vì nó được sử dụng rộng rãi trong nhiều vấn đề lập lịch khác nhau. Ngoài ra, không gian hành động để lựa chọn máy cho 1 công việc bao gồm SPT & Thời gian xử lý dài nhất (LPT). Do đó, tổng kích thước của không gian hành động bằng số PDR để lựa chọn công việc nhân với số PDR để lựa chọn máy. Định nghĩa của các PDR này được liệt kê như sau:

  * SPT: $\min Z_{ij} = p_{ij}$
  * LPT: $\max Z_{ij} = p_{ij}$
  * FDD/MWKR: $\min Z_{ij} = \frac{\sum_1^j p_{ij}}{\sum_j^{n_i} p_{ij}}$
  * MOR: $\max Z_{ij} = n_i - j + 1$
  * LRM: $\max Z_{ij} = \sum_{j+1}^{n_i} p_{ij}$
  * FIFO: $\max Z_{ij} = t - Rt_i$

  trong đó Z_{ij} là chỉ số ưu tiên của phép toán O_{ij}. p_{ij} là thời gian xử lý của thao tác O_{ij}. n_i: số thao tác cho công việc J_i. j: số thao tác đã hoàn thành. t: thời gian hiện tại & Rt_i: thời gian giải phóng J_i.

- **3.3. Reward function based on scheduling area.** In this paper, propose a comprehensible reward function based on scheduling area (1)

$$\text{reward}(s_t, a_t) = -p_{a,j} - \sum_M \text{vacancy}_m(s_t, s_{t+1})$$

  where processing time of allocated operations & time vacancy of all machines is computed after each dispatching action, where s_t: current state & s_{t+1} is next state after applying action a_t; a_t is jth operation of a job with processing time $p_{a,j}$; $\text{vacancy}(s_t, s_{t+1})$ is a function returning total time vacancy on machine set M while transitioning from state s_t to s_{t+1}.

  – Hàm thưởng dựa trên vùng lập lịch. Trong bài báo này, chúng tôi đề xuất 1 hàm thưởng dễ hiểu dựa trên vùng lập lịch (1)

$$\text{reward}(s_t, a_t) = -p_{a,j} - \sum_M \text{vacancy}_m(s_t, s_{t+1})$$

  trong đó thời gian xử lý của các tác vụ được phân bổ & thời gian trống của tất cả các máy được tính sau mỗi hành động điều phối, trong đó s_t: trạng thái hiện tại & s_{t+1} là trạng thái tiếp theo sau khi áp dụng tác vụ a_t; a_t là tác vụ thứ j của 1 tác vụ có thời gian xử lý $p_{a,j}$; $\text{vacancy}(s_t, s_{t+1})$ là 1 hàm trả về tổng thời gian trống trên tập máy M khi chuyển từ trạng thái s_t sang s_{t+1}.

Proposed reward function is motivated by fact: total processing time of all jobs & time vacancy on all machines constructs scheduling area of all machines which equals maximum make-span multiplied by number of machines. As is demonstrated in Fig. 5: An example to show the scheduling area., scheduling area is composed of total processing time of all jobs (shaded area) & time vacancy on all machines (white color area). Relationship between accumulated reward & maximum scheduling makespan is derived from (2)

$$R = -a - b = -(a + b) = -S = -|M| * \text{makespan}$$

where R is accumulated reward, a: total processing time of all jobs, b: total time vacancy on all machines, S: scheduling time area & $|M|$: number of machines.

– Hàm phần thưởng được đề xuất dựa trên thực tế: tổng thời gian xử lý của tất cả các công việc & thời gian trống trên tất cả các máy tạo nên vùng lập lịch của tất cả các máy, bằng khoảng thời gian hoàn thành tối đa nhân với số máy. Như được minh họa trong Hình 5: Ví dụ minh họa vùng lập lịch., vùng lập lịch bao gồm tổng thời gian xử lý của tất cả các công việc (vùng tô bóng) & thời gian trống trên tất cả các máy (vùng màu trắng). Mối quan hệ giữa phần thưởng tích lũy & khoảng thời gian hoàn thành tối đa được suy ra từ (2)

$$R = -a - b = -(a + b) = -S = -|M| * \text{khongthigianhonthnh}$$

trong đó R là phần thưởng tích lũy, a: tổng thời gian xử lý của tất cả các công việc, b: tổng thời gian trống trên tất cả các máy, S: vùng thời gian lập lịch & $|M|$: số máy.

Obviously shown from (2): total reward & maximum makespan are negatively linearly dependent & coefficient is number of machines. I.e., minimizing maximum scheduling makespan is equivalent to maximizing total reward.

– Hiển nhiên từ (2): tổng phần thưởng & thời gian hoàn thành tối đa phụ thuộc tuyến tính âm & hệ số là số máy. Nghĩa là, việc giảm thiểu thời gian hoàn thành tối đa tương đương với việc tối đa hóa tổng phần thưởng.

- **3.4. Model training method based on PPO.** In order to strengthen representation ability of RL scheduling method, scheduling policy is usually represented by DNNs e.g. CNN, RNN, & MLP. In our method, state feature vector is 1st fed to MLP to obtain a scalar score for each action & a Softmax function is then applied to output a distribution over computed score, which is shown in (3)

$$p(a_t|s_t) = \text{Softmax}(\text{MLP}_{\pi_\theta}(s_t)),$$

where $p(a_t|s_t)$ is selection probability of action a_t at time t in state s_t & θ is parameter of scheduling policy π. Structure of our scheduling policy is demonstrate in Fig. 4, which constructs actor network of PPO agent. Similar to actor network, critic network is implemented by MLP with 1 hidden layer. Parameters of our scheduling policy are learned by a clipped PPO whose loss function is expressed in (4)

$$L(\theta) = E_t[\min(r_t(\theta)A_t, \text{clip}(r_t(\theta), 1 \pm \epsilon)A_t)]$$

where $r_t(\theta) = \dfrac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$, A_t is an estimator of advantage function at time step t, clip is a clipping function & ϵ is a hyper parameter which is used to limit boundary of objective function.

– Phương pháp huấn luyện mô hình dựa trên PPO. Để tăng cường khả năng biểu diễn của phương pháp lập lịch RL, chính sách lập lịch thường được biểu diễn bằng các DNN, ví dụ: CNN, RNN, & MLP. Trong phương pháp của chúng tôi, vectơ đặc trưng trạng thái đầu tiên được đưa vào MLP để thu được điểm số vô hướng cho mỗi hành động & sau đó áp dụng hàm Softmax để đưa ra phân phối trên điểm số đã tính toán, được thể hiện trong (3)

$$p(a_t|s_t) = \text{Softmax}(\text{MLP}_{\pi_\theta}(s_t)),$$

trong đó $p(a_t|s_t)$ là xác suất lựa chọn hành động a_t tại thời điểm t trong trạng thái s_t & θ là tham số của chính sách lập lịch π. Cấu trúc của chính sách lập lịch của chúng tôi được minh họa trong Hình 4, trong đó xây dựng mạng lưới tác nhân của tác tử PPO. Tương tự như mạng diễn viên, mạng phê bình được triển khai bằng MLP với 1 lớp ẩn. Các tham số của chính sách lập lịch của chúng tôi được học bởi 1 PPO bị cắt xén có hàm mất mát được biểu thị trong (4)

$$L(\theta) = E_t[\min(r_t(\theta)A_t, \text{clip}(r_t(\theta), 1 \pm \epsilon)A_t)]$$

trong đó $r_t(\theta) = \dfrac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$, A_t là 1 ước lượng của hàm lợi thế tại bước thời gian t, clip là 1 hàm cắt xén & ϵ là 1 siêu tham số được sử dụng để giới hạn biên của hàm mục tiêu.

Detailed training process is provided by Algorithm 2: Model training method based on PPO. Training process includes 2 aspects: data collection & policy learning. When collecting training data, T independent complete trajectories of scheduling are generated & they are collected in memory buffer M. On policy learning stage, training data are used for K times. At each time, these data are randomly divided into batches whose length is related to scale of instances & scheduling agent learns from each batch data. After replaying these experience samples evenly, prioritized experience replay is performed for C times. Training process will stop until episode reaches maximum iterations or result is convergent or scheduling is time out. Define: result is thought as convergent if makespan values are same in 30 decision steps & training time is limited in an hour.

– Quá trình đào tạo chi tiết được cung cấp bởi Thuật toán 2: Phương pháp đào tạo mô hình dựa trên PPO. Quá trình đào tạo bao gồm 2 khía cạnh: thu thập dữ liệu & học chính sách. Khi thu thập dữ liệu đào tạo, T quỹ đạo hoàn chỉnh độc lập

của lập lịch được tạo & chúng được thu thập trong bộ đệm M. Ở giai đoạn học chính sách, dữ liệu đào tạo được sử dụng trong K lần. Tại mỗi thời điểm, những dữ liệu này được chia ngẫu nhiên thành các lô có độ dài liên quan đến quy mô của các trường hợp & tác nhân lập lịch học từ mỗi dữ liệu lô. Sau khi phát lại các mẫu trải nghiệm này 1 cách đồng đều, việc phát lại trải nghiệm được ưu tiên sẽ được thực hiện trong C lần. Quá trình đào tạo sẽ dừng lại cho đến khi tập đạt số lần lặp tối đa hoặc kết quả hội tụ hoặc lập lịch hết thời gian. Định nghĩa: kết quả được coi là hội tụ nếu các giá trị makespan giống nhau trong 30 bước quyết định & thời gian đào tạo bị giới hạn trong 1 giờ.

- **4. Experiments.** In this sect, in order to show effectiveness of our DRL scheduling environment for FJSP (Algorithm 1) & to evaluate performance of proposed DRL scheduling methods, a group of experiments are performed on public FJSP benchmarks with various sizes & results are compared with different types of scheduling methods from recent literature. Finally, training details e.g. training time are demonstrated.

  – Trong phần này, để chứng minh tính hiệu quả của môi trường lập lịch DRL cho FJSP (Thuật toán 1) & đánh giá hiệu suất của các phương pháp lập lịch DRL được đề xuất, 1 nhóm các thử nghiệm được thực hiện trên các chuẩn FJSP công khai với nhiều kích thước khác nhau & kết quả được so sánh với các loại phương pháp lập lịch khác nhau từ các tài liệu gần đây. Cuối cùng, các chi tiết đào tạo, ví dụ như thời gian đào tạo, sẽ được trình bày.

  - **4.1. Benchmark instances & baseline models.** In this paper, 2 well-known benchmarks of FJSP are used to evaluate our proposed methods including MK instances (MK01-MK10) in [4] & 3 group of LA instances (edata, rdata, & vdata each with 40 instances) in [12].

    – Trong bài báo này, 2 chuẩn mực nổi tiếng của FJSP được sử dụng để đánh giá các phương pháp đề xuất của chúng tôi bao gồm các thể hiện MK (MK01-MK10) trong [4] & 3 nhóm thể hiện LA (edata, rdata, & vdata mỗi nhóm có 40 thể hiện) trong [12].

    This paper compared with PDR, exact solver, meta-heuristic & DRL models. 6 PDRs are used to compare scheduling results where 4 out of 6 PDRs are compared in old scheduling environment & our proposed environment. Also compare with well-known Google OR-Tools which is a powerful constraint programming solver showing strong performance in solving industrial scheduling problems. For meta-heuristic scheduling methods, 2 recent improved Genetic Algorithms are selected from [18] & [5]. For DRL scheduling methods, compared with competing models in recent 3 years, including 4 DRl methods proposed by Feng et al.[9], Zeng et al.[26], Song et al.[22] & Lei et al.[15] resp., GSMA model [13] which is a MARL scheduling method & obtains state-of-art results, DANILE model [23] which is based on attention mechanism.

    – Bài báo này so sánh với PDR, bộ giải chính xác, mô hình meta-heuristic & DRL. 6 PDR được sử dụng để so sánh kết quả lập lịch trong đó 4 trong số 6 PDR được so sánh trong môi trường lập lịch cũ & môi trường đề xuất của chúng tôi. Cũng so sánh với Google OR-Tools nổi tiếng, 1 bộ giải lập trình ràng buộc mạnh mẽ cho thấy hiệu suất cao trong việc giải quyết các vấn đề lập lịch công nghiệp. Đối với các phương pháp lập lịch meta-heuristic, 2 Thuật toán di truyền được cải tiến gần đây được chọn từ [18] & [5]. Đối với các phương pháp lập lịch DRL, so sánh với các mô hình cạnh tranh trong 3 năm gần đây, bao gồm 4 phương pháp DRl do Feng et al.[9], Zeng et al.[26], Song et al.[22] & Lei et al.[15] đề xuất, tương ứng, mô hình GSMA [13] là 1 phương pháp lập lịch MARL & thu được kết quả tiên tiến, mô hình DANILE [23] dựa trên cơ chế chú ý.

  - **4.2. Model configurations.** PPO agent adopts actor-critic architecture where actor networks represent scheduling policy & critic networks calculate state-value of policy networks. Actor network is implemented by MLP & Softmax function while critic network is only represented by MLP. Both networks are optimized by Adam optimizer, use ReLU as activation function & have only 1 hidden layer with dynamic hidden dimension which equals length of state features. For each problem size, train policy network for ≤ 8000 iterations, each of which contains 9 independent trajectories (i.e., complete scheduling process of instances) & use a dynamic batch size which equals 2 times of scale (total number of operations of all jobs) of an instance. For PPO, set epochs of updating network to 10 & clipping parameter epsilon to 0.2. Set discount factor γ to 0.999 & learning rate are $1e-3, 3e-3$ for actor & critic network resp. For prioritized experience replay, parameter α is set to 0.6, value of β anneals from 0.4 to 1, number of prioritized experience replay C is set to 1, & number of convergence training steps is 2000.

    – Tác nhân PPO áp dụng kiến trúc actor-critic trong đó các mạng actor biểu diễn chính sách lập lịch & các mạng critic tính toán giá trị trạng thái của các mạng policy. Mạng actor được triển khai bởi hàm MLP & Softmax trong khi mạng critic chỉ được biểu diễn bởi MLP. Cả 2 mạng đều được tối ưu hóa bởi trình tối ưu hóa Adam, sử dụng ReLU làm hàm kích hoạt & chỉ có 1 lớp ẩn với chiều ẩn động bằng với độ dài của các đặc trưng trạng thái. Đối với mỗi kích thước vấn đề, hãy huấn luyện mạng policy cho ≤ 8000 lần lặp, mỗi lần lặp chứa 9 quỹ đạo độc lập (i.e., quy trình lập lịch hoàn chỉnh của các trường hợp) & sử dụng kích thước lô động bằng 2 lần quy mô (tổng số thao tác của tất cả các công việc) của 1 trường hợp. Đối với PPO, đặt các kỷ nguyên cập nhật mạng thành 10 & tham số cắt epsilon thành 0,2. Đặt hệ số chiết khấu γ thành $0,999$ & tốc độ học là $1e-3, 3e-3$ tương ứng với mạng actor & critic. Đối với phát lại trải nghiệm được ưu tiên, tham số α được đặt thành 0,6, giá trị của β được ủ từ 0,4 đến 1, số lần phát lại trải nghiệm được ưu tiên C được đặt thành 1, & số bước đào tạo hội tụ là 2000.

  - **4.3. Results analysis.** In order to evaluate our proposed scheduling environment, 1st test performance of 6 PDRs in our proposed scheduling environment on MK benchmark instances where 6 PDRs are used to select a job while SPT is used for selection of machines. Results are shown in Table 1: Scheduling results of PDRs on MK benchmark instances. Widely-used 4 PDRs (SPT, MWKR, FIFO, MOR) are performed in our environment & results are compared with that in old scheduling environment where their best results are selected from literature [22]. Symbol "-" means: specific results are not recorded in literature. Best scheduling result of all 6 PDRs on each instance are recorded in minPDR row. Besides, results of 2 DRL

scheduling methods proposed resp. by Feng et al.[9] & Zeng et al.[26] are also compared in Table 1 & their results are directly from their literature instead of our implementation environment.

– Để đánh giá môi trường lập lịch đề xuất của chúng tôi, hiệu suất thử nghiệm đầu tiên của 6 PDR trong môi trường lập lịch đề xuất của chúng tôi trên các phiên bản chuẩn MK trong đó 6 PDR được sử dụng để chọn 1 công việc trong khi SPT được sử dụng để chọn máy. Kết quả được hiển thị trong Bảng 1: Kết quả lập lịch của PDR trên các phiên bản chuẩn MK. 4 PDR được sử dụng rộng rãi (SPT, MWKR, FIFO, MOR) được thực hiện trong môi trường của chúng tôi & kết quả được so sánh với kết quả trong môi trường lập lịch cũ, trong đó kết quả tốt nhất của chúng được chọn từ tài liệu [22]. Ký hiệu "-" có nghĩa là: kết quả cụ thể không được ghi lại trong tài liệu. Kết quả lập lịch tốt nhất của tất cả 6 PDR trên mỗi phiên bản được ghi lại trong hàng minPDR. Bên cạnh đó, kết quả của 2 phương pháp lập lịch DRL do Feng et al.[9] & Zeng et al.[26] đề xuất cũng được so sánh trong Bảng 1 & kết quả của chúng được lấy trực tiếp từ tài liệu của họ thay vì môi trường triển khai của chúng tôi.

As shown in table 1, results indicate: performance of SPT, FIFO, MOR in our environment is improved while performance of MWKR is suddenly worse than before. LRM obtained best average results among all 6 PDRs & even better than some DRL scheduling methods e.g. models proposed by Feng et al. & Zeng et al., which is surprisingly interesting. Best result on different instances is distributed in different PDRs, e.g. MWKR obtained best results on MK1 & FIFO performs best on MK3. So that, minPDR can get better results than LRM.

– Như thể hiện trong bảng 1, kết quả cho thấy: hiệu suất của SPT, FIFO, MOR trong môi trường của chúng tôi được cải thiện trong khi hiệu suất của MWKR đột nhiên kém hơn trước. LRM đạt được kết quả trung bình tốt nhất trong số tất cả 6 PDR & thậm chí còn tốt hơn 1 số phương pháp lập lịch DRL, ví dụ như các mô hình do Feng & cộng sự đề xuất & Zeng & cộng sự, điều này thật đáng ngạc nhiên. Kết quả tốt nhất trên các trường hợp khác nhau được phân phối trong các PDR khác nhau, ví dụ: MWKR đạt được kết quả tốt nhất trên MK1 & FIFO hoạt động tốt nhất trên MK3. Do đó, minPDR có thể đạt được kết quả tốt hơn LRM.

2nd group of experiments are also performed on MK benchmark instances to evaluate generality of our proposed DRL scheduling agent in our environment. Train these instances independently for 5 times & average results are recorded in "Ours" column. As shown in Table 2, compare performance of our proposed DRL scheduling model with exact solver (OR-Tools), meta-heuristic scheduling methods (2SGA[18] & SLGA [5]), DRL scheduling methods (GMAS[13], DANILE[23], & models proposed by Song et al.[22]) & minPDR method. LB & UB columns resp. record lower bound & upper bound scheduling results in literature. Results in these compared methods are directly from literature except results of PDRs which are performed in our environment.

– Nhóm thí nghiệm thứ 2 cũng được thực hiện trên các trường hợp chuẩn MK để đánh giá tính tổng quát của tác nhân lập lịch DRL được đề xuất của chúng tôi trong môi trường của chúng tôi. Đào tạo các trường hợp này độc lập trong 5 lần & kết quả trung bình được ghi lại trong cột "Của chúng tôi". Như thể hiện trong Bảng 2, hãy so sánh hiệu suất của mô hình lập lịch DRL được đề xuất của chúng tôi với bộ giải chính xác (OR-Tools), các phương pháp lập lịch meta-heuristic (2SGA[18] & SLGA [5]), các phương pháp lập lịch DRL (GMAS[13], DANILE[23], & các mô hình do Song et al.[22] đề xuất) & phương pháp minPDR. Các cột LB & UB tương ứng ghi lại kết quả lập lịch giới hạn dưới & giới hạn trên trong tài liệu. Kết quả trong các phương pháp được so sánh này được lấy trực tiếp từ tài liệu, ngoại trừ kết quả của PDR được thực hiện trong môi trường của chúng tôi.

Average makespan is usually used to evaluate accuracy of scheduling solutions. As demonstrated in Table 2: Makespan performance of our DRL model on MK benchmark instances, GMAS which is a DRL scheduling method based on MARL, obtained best average result. Nevertheless, they used optimal results of each instance rather than average results. Average makespan of our proposed DRL model is smaller than SLGA, DANILE, Song, & minPDR methods & is larger than OR-Tools, 2SGA & GMAS methods, which shows: performance of DRL scheduling methods are getting close to meta-heuristic & exact solver. However, different from complex scheduling networks of GMAS, our model is simple & easy to train to converge, which is stable.

– Makespan trung bình thường được sử dụng để đánh giá độ chính xác của các giải pháp lập lịch. Như được minh họa trong Bảng 2: Hiệu suất Makespan của mô hình DRL của chúng tôi trên các trường hợp chuẩn MK, GMAS, 1 phương pháp lập lịch DRL dựa trên MARL, đã thu được kết quả trung bình tốt nhất. Tuy nhiên, họ đã sử dụng kết quả tối ưu của từng trường hợp thay vì kết quả trung bình. Makespan trung bình của mô hình DRL đề xuất của chúng tôi nhỏ hơn các phương pháp SLGA, DANILE, Song, & minPDR & lớn hơn các phương pháp OR-Tools, 2SGA & GMAS, điều này cho thấy: hiệu suất của các phương pháp lập lịch DRL đang tiến gần đến trình giải meta-heuristic & chính xác. Tuy nhiên, khác với các mạng lập lịch phức tạp của GMAS, mô hình của chúng tôi đơn giản & dễ huấn luyện để hội tụ, điều này rất ổn định.

Beside, evaluate convergence performance of our DRL scheduling method. As is demonstrated in Fig. 6: Taining trajectories & training time on MK benchmarks., our DRL model on all MK instances is convergent in half an hour on average & number of needed trajectories is < 900. Convergence time of 8 out of 10 instances is < 600 seconds, which is close to industrial time limitation.

– Ngoài ra, hãy đánh giá hiệu suất hội tụ của phương pháp lập lịch DRL của chúng tôi. Như được minh họa trong Hình 6: Định vị quỹ đạo & thời gian huấn luyện trên chuẩn MK., mô hình DRL của chúng tôi trên tất cả các trường hợp MK đều hội tụ trung bình trong nửa giờ & số quỹ đạo cần thiết là < 900. Thời gian hội tụ của 8 trên 10 trường hợp là < 600 giây, gần với giới hạn thời gian công nghiệp.

In order to show stability & generality of our proposed DRL scheduling agent in our environment, more experiments are performed on LA benchmark instances, including edta, rdata, & vdata, whose flexibility is getting increased. As shown in Table 3: Average makespan comparison LA instances, compare with various scheduling methods in literature e.g. exact solver

OR-Tools, meta-heuristic 2SGA, DRL scheduling methods: DANILE, GMAS, Lei [15] & Song as well as PDR methods. Results of these methods are directly from literature. Results of PDR methods are from running in our proposed environment & only minimum result of 6 PDRs on each instance is recorded in minPDR column.

– Để chứng minh tính ổn định & tính tổng quát của tác nhân lập lịch DRL được đề xuất trong môi trường của chúng tôi, nhiều thử nghiệm hơn đã được thực hiện trên các phiên bản chuẩn LA, bao gồm edta, rdata, & vdata, với tính linh hoạt ngày càng tăng. Như được thể hiện trong Bảng 3: So sánh makespan trung bình của các phiên bản LA, hãy so sánh với các phương pháp lập lịch khác nhau trong tài liệu, ví dụ: bộ giải chính xác OR-Tools, meta-heuristic 2SGA, các phương pháp lập lịch DRL: DANILE, GMAS, Lei [15] & Song cũng như các phương pháp PDR. Kết quả của các phương pháp này được trích dẫn trực tiếp từ tài liệu. Kết quả của các phương pháp PDR được thực hiện trong môi trường được đề xuất của chúng tôi & chỉ có kết quả tối thiểu là 6 PDR trên mỗi phiên bản được ghi lại trong cột minPDR.

As is demonstrated in Table 3, our proposed DRL scheduling agent got smaller average makespan than DANILE, Lei, Song, & PDR models & obtained larger average makespan than OR-Tools, GMAS, & 2SGA, which shows competing performance of our proposed DRL scheduling agent in our environment. More interestingly, minPDR obtained smaller makespan than Lei model which is used to solve large-scale dynamic FJSP. That shows efficiency of our proposed environment again.

– Như được minh họa trong Bảng 3, tác nhân lập lịch DRL đề xuất của chúng tôi có makespan trung bình nhỏ hơn các mô hình DANILE, Lei, Song, & PDR & đạt makespan trung bình lớn hơn OR-Tools, GMAS, & 2SGA, điều này cho thấy hiệu suất cạnh tranh của tác nhân lập lịch DRL đề xuất trong môi trường của chúng tôi. Thú vị hơn, minPDR đạt makespan nhỏ hơn mô hình Lei, được sử dụng để giải các bài toán FJSP động quy mô lớn. Điều này 1 lần nữa cho thấy hiệu quả của môi trường chúng tôi đề xuất.

Finally, training time of our proposed DRL scheduling agent on edata, rdata, & vdata are depicted in Fig. 7: Training time on edata, rdata, & vdata instances. Our proposed DRL model can be trained to converge in an hour on all benchmark instances & total training time on edta, rdata, & vdata are 22064, 31650 & 40137 seconds, resp., which shows training time rises as increase of complicity of scheduling instances.

– Cuối cùng, thời gian đào tạo của tác nhân lập lịch DRL được đề xuất của chúng tôi trên edata, rdata, & vdata được mô tả trong Hình 7: Thời gian đào tạo trên các phiên bản edata, rdata, & vdata. Mô hình DRL được đề xuất của chúng tôi có thể được đào tạo để hội tụ trong 1 giờ trên tất cả các phiên bản chuẩn & tổng thời gian đào tạo trên edta, rdata, & vdata tương ứng là 22064, 31650 & 40137 giây, cho thấy thời gian đào tạo tăng lên khi mức độ phức tạp của các phiên bản lập lịch tăng lên.

- **5. Conclusion.** In this paper, proposed a chronological discrete event simulation based DRL environment for FJSP. Scheduling process is described by a simulation algorithm where state changes are captured by state variables & reward function is calculated based on scheduling area of machines at each decision step. In this simulaiton environment, an end-to-end DRL framework is proposed based on actor-critic PPO, providing a flexible infrastructure for design of state representation, action space & scheduling policy networks. Besides, a simple DRL model for FJSP is presented by defining state representation of very short state features based on 2 state variables in simulation environment, action space composed of widely-used priority dispatching rules in literature & scheduling policy baesd on MLP with only 1 hidden layer. Various experiments are performed on classic benchmark instances with different sizes & results show: performance of PDR is improved in our environment, even better than some DRL methods. Besides, our DRL scheduling method provides competing scheduling performance compared with DRL, meta-heuristic & PDR methods.

  – Bài báo này đề xuất 1 môi trường DRL dựa trên mô phỏng sự kiện rời rạc theo trình tự thời gian cho FJSP. Quá trình lập lịch được mô tả bằng 1 thuật toán mô phỏng, trong đó các thay đổi trạng thái được ghi lại bởi các biến trạng thái & hàm thưởng được tính toán dựa trên diện tích lập lịch của máy tại mỗi bước quyết định. Trong môi trường mô phỏng này, 1 khuôn khổ DRL đầu cuối được đề xuất dựa trên PPO tác nhân-phê bình, cung cấp 1 cơ sở hạ tầng linh hoạt cho việc thiết kế biểu diễn trạng thái, không gian hành động & mạng lưới chính sách lập lịch. Bên cạnh đó, 1 mô hình DRL đơn giản cho FJSP được trình bày bằng cách định nghĩa biểu diễn trạng thái của các đặc trưng trạng thái rất ngắn dựa trên 2 biến trạng thái trong môi trường mô phỏng, không gian hành động bao gồm các quy tắc phân bổ ưu tiên được sử dụng rộng rãi trong tài liệu & chính sách lập lịch dựa trên MLP với chỉ 1 lớp ẩn. Nhiều thử nghiệm khác nhau được thực hiện trên các phiên bản chuẩn cổ điển với các kích thước khác nhau & kết quả cho thấy: hiệu suất của PDR được cải thiện trong môi trường của chúng tôi, thậm chí còn tốt hơn 1 số phương pháp DRL. Hơn nữa, phương pháp lập lịch DRL của chúng tôi cung cấp hiệu suất lập lịch cạnh tranh so với các phương pháp DRL, meta-heuristic & PDR.

  Future research will mainly focus on design of scheduling policy networks as well as state representation. State features can be thought as texts or images so that various networks in NLP & CV fields can be used in scheduling policy networks, e.g. SPP networks, TextCNN, & Transformer.

  – Nghiên cứu trong tương lai sẽ chủ yếu tập trung vào thiết kế mạng lưới chính sách lập lịch cũng như biểu diễn trạng thái. Các đặc điểm trạng thái có thể được hiểu dưới dạng văn bản hoặc hình ảnh, do đó, các mạng lưới khác nhau trong trường NLP & CV có thể được sử dụng trong mạng lưới chính sách lập lịch, ví dụ: mạng SPP, TextCNN, & Transformer.

3 Miscellaneous