MỤC LỤC

DANH MỤC HÌNH ẢNH

DANH MỤC BẢNG

Chương 1

Giới thiệu

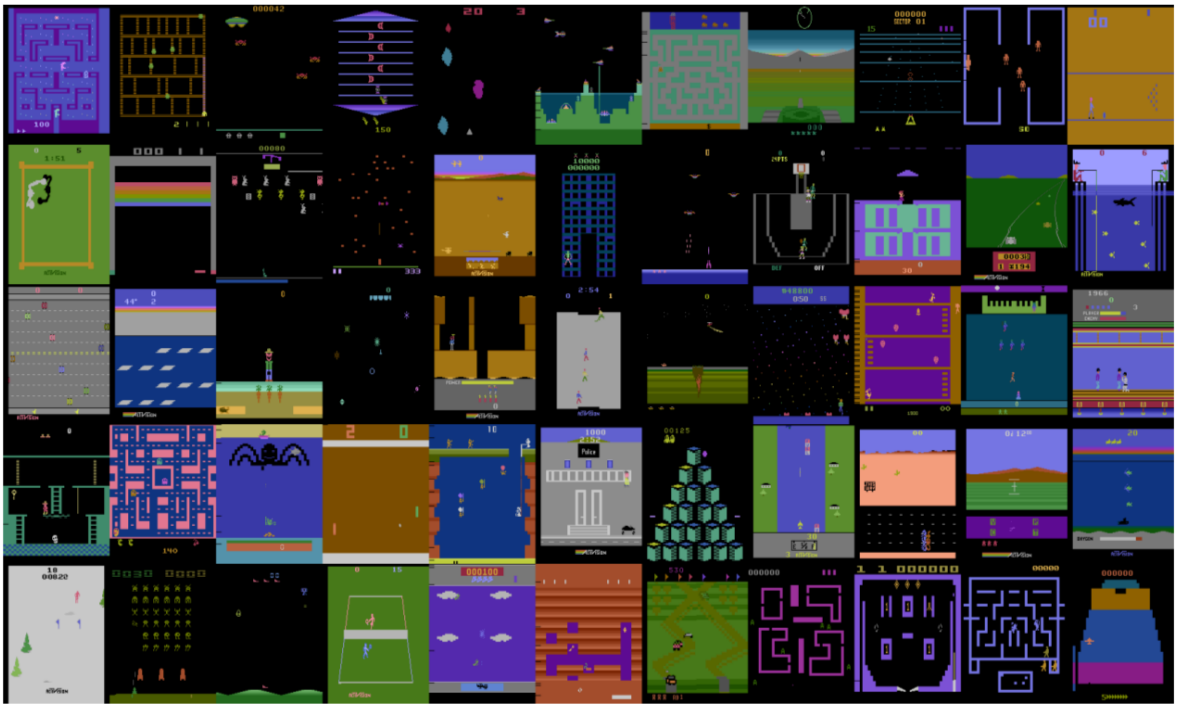
Những năm gần đây, **học tăng cường** (reinforcement learning) liên tục đạt được những thành tựu quan trọng trong lĩnh vực Trí tuệ nhân tạo. Những đóng góp nổi bật của học tăng cường bao gồm: tự động điều khiển robot di chuyển, điều khiển mô hình máy bay trực thăng, hệ thống chơi cờ vây ... Trong số các thành tựu này, hệ thống chơi cờ vây với khả năng chiến thắng những kỳ thủ hàng đầu thế giới là một cột mốc quan trọng của lĩnh vực Trí tuệ nhân tạo. Dù vậy, học tăng cường không phải là một phương pháp mới được phát triển gần đây. Nền tảng lý thuyết của học tăng cường đã được xây dựng từ những năm 1980 [16].

Được xây dựng nhằm mô phỏng quá trình học của con người, ý tưởng chính của học tăng cường là tìm cách lựa chọn hành động *tối ưu* để nhận được **nhiều nhất giá trị điểm thưởng** (reward). Giá trị điểm thưởng này có ý nghĩa tương tự cảm nhận của con người về môi trường. Khi một đứa trẻ bắt đầu "học" về thế giới xung quanh của mình, những cảm giác như đau đớn (ứng với điểm thưởng thấp) hay vui sướng (điểm thưởng cao) chính là mục tiêu cần tối ưu của việc học. Việc đứa trẻ thực hiện các hành động để tăng cảm giác vui sướng (và giảm đau đớn) cũng tương đồng với việc hệ thống học tăng cường tối đa hoá giá trị điểm thưởng. Một điểm quan trọng của học tăng cường là các thuật toán được xây dựng với ít giả định nhất có thể về môi trường xung quanh. Hệ thống sử dụng học tăng cường (agent) không cần biết cách thức hoạt động của môi trường để hoạt động. Ví dụ như để điều khiển robot tìm được đi trong mê cung, hệ thống không cần biết mê cung được xây dựng thế nào hay kích thước là bao nhiêu.

Việc hạn chế tối đa những ràng buộc về dữ liệu đầu giúp cho thuật toán học tăng cường có thể áp dụng vào nhiều bài toán thực tế.

Học tăng cường được xem là một nhánh trong lĩnh vực máy học ngoài hai nhánh: *học có giám sát* và *học không có giám sát*. Trong bài toán học có giám sát, dữ liệu học thường được gán nhãn thủ công sẵn (hand-labeled); hệ thống cần tìm cách mối liên hệ giữa dữ liệu và nhãn tương ứng. Mối liên hệ tìm được sẽ dùng để dự đoán nhãn của dữ liệu mới. Các nhãn này có thể xem như là sự hướng dẫn trong quá trình học; tính đúng sai của việc học lúc này có thể được xác định dựa vào kết quả dự đoán của hệ thống và nhãn đúng của dữ liệu. Tiếp theo đối với những bài toán học không có giám sát, dữ liệu học thường không được gán nhãn nên công việc của việc học là phải tự tìm ra được cấu trúc "ẩn" bên dưới dữ liệu đó. Khác với hai loại bài toán vừa nêu, trong bài toán học tăng cường, hệ thống *không nhận được nhãn thực sự* (tức hành động tối ưu của tình huống hiện tại) mà chỉ nhận được điểm thưởng từ môi trường. Điểm thưởng lúc này chỉ thể hiện mức độ "tốt/xấu" của hành động vừa chọn chứ không nói lên hành động đó có phải là hành động tối ưu hay không. Điểm thưởng này thông thường rất **thưa**: ta có thể chỉ nhận được điểm thưởng có ý nghĩa (khác không) sau hàng trăm hành động. Ngoài ra, giá trị điểm thưởng thường không đơn định và rất **nhiễu**: cùng một hành động tại cùng một trạng thái, ta có thể nhận được điểm thưởng khác nhau vào hai thời điểm khác nhau. Hai vấn đề này (tính thưa và nhiễu của điểm thưởng) cũng chính là những khó khăn cơ bản của bài toán học tăng cường.

Các trò chơi điện tử thường hay có điểm số mà người chơi cần phải tối ưu hoá. Đặc điểm này trùng với yêu cầu của bài toán học tăng cường, vì vậy các trò chơi này cũng chính là những ứng dụng tự nhiên nhất của phương pháp học tăng cường. Trong khoá luận này, chúng em áp dụng phương pháp học tăng cường nhằm xây dựng **hệ thống tự động chơi các game** trên hệ máy Atari. Dữ liệu đầu vào của hệ thống chỉ bao gồm các frame ảnh RGB cùng với điểm số. Từ hình ảnh thô này, hệ thống cần tìm cách chơi sao cho điểm số cuối màn chơi (episode) là lớn nhất có thể. Lưu ý rằng điểm số được game cung cấp hệ thống dưới dạng số thực (chứ không cần phải nhìn hình ảnh thô để "đọc" điểm số). Điểm khó khăn của bài toán này là hệ thống hoàn toàn không biết quy luật



Hình 1.1: Hình ảnh các game trên hệ máy Atari. Hình được chỉnh sửa từ [5]

của game trước khi bắt đầu quá trình học mà phải tự tìm hiểu quy luật và chiến thuật chơi tối ưu. Lý do khoá luận sử dụng game của máy Atari là vì các game này có quy luật chơi tương đối đơn giản nhưng lại rất đa dạng. Mỗi màn chơi thường có độ dài vừa phải (từ 2 - 15 phút) và số hành động có ý nghĩa không quá nhiều (18 hành động). Ngoài ra, các trò chơi này có thể được giả lập trên máy vi tính với tốc độ cao, giúp quá trình học được tăng tốc.

Một số khó khăn trước mắt có thể thấy ở bài toán tự động chơi game bao gồm:

- **Hệ thống biết luật chơi của game**. Chính vì thế nó cũng không thể biết được hành động nào nên làm hoặc không nên làm ứng với từng tình huống cụ thể.
- **Dữ liệu đầu vào là hình ảnh thô** (ảnh RGB có kích thước 210×160). Để học được một chiến thuật chơi đơn giản thì hệ thống cũng phải chơi "thử và sai" một số lượng lớn màn chơi (có thể lên đến 10000 frame). Vì vậy, lượng dữ liệu đầu vào cần phải xử lý là rất lớn.
- **Các game rất khác nhau**. Khác biệt này là về cả hình ảnh lẫn nội dung

của game. Để có thể học cách chơi của nhiều game khác nhau thì thuật toán học phải mang tính tổng quát cao, không sử dụng các tính chất riêng biệt của từng game.

- **Cần có chiến thuật chơi riêng biệt cho từng game**. Để đạt được điểm số cao (ngang hoặc hơn điểm số của con người) thì phải tìm được chiến thuật chơi mang tính lâu dài. Những phương pháp tham lam, lựa chọn hành động để đạt điểm tối đa trong tương lai gần thường không tối ưu.

Một trong những tiếp cận đầu tiên cho bài toán tự động chơi game là cuộc thi AAAI General Game Playing được đề xuất từ năm 2005 bởi đại học Stanford [6]. Trong cuộc thi này, các đội thi nhận được mô tả sơ bộ cho những game được sử dụng để thi. Các mô hình chi tiết được thiết kế bên dưới các game này không được mô tả cho các đội chơi. Dựa vào những thông tin nhận được, các đội chơi phải thiết kế một mô hình tổng quát để có thể chơi những game này; trong cuộc thi, họ sẽ chơi ngẫu nhiên một trong những game đã được mô tả [6]. Đội thắng cuộc trong cuộc thi này đã sử dụng mô hình "Monte Carlo Tree Search" (một mô hình phổ biến của học tăng cường) để tìm chiến lược chơi trong những game này.

Trong khoảng ba năm trở lại, tập những game trên hệ máy Atari 2600 trở nên phổ biến trong việc đánh giá khả năng của những phương pháp tiếp cận cho bài toán tự động chơi game. Những game này bao gồm nhiều thể loại khác nhau như: đối kháng, chiến thuật... Sự đa dạng trong cách chơi của những game Atari 2600 cùng giao diện lập trình đơn giản làm cho chúng gây được sự chú ý lớn trong cộng đồng Trí tuệ nhân tạo. Nhiều phương pháp đã được đề xuất liên quan tới bài toán tự động chơi game trên hệ máy Atari 2600. [1, 2] thực hiện chia những frame ảnh trong game thành nhiều phần; sau đó xây dựng cấu trúc cây và áp dụng mô hình học Bayesian để mô phỏng lại mô hình của "môi trường" đã được xây dựng trong các game này cho hệ thống. Tuy nhiên, hướng tiếp cận này giả định những phần ảnh lân cận là đủ thông tin để dự đoán những phần ảnh trung tâm, do đó không phù hợp trong một vài game Atari có sự tương tác phức tạp. [9] sử dụng mạng nơ-ron với cấu trúc nông để tạo ra một chính

sách có thể tự động chơi một số game Atari. Phương pháp sử dụng thuật toán lập trình tiến hóa để tìm trọng số liên kết tối ưu trong mạng nơ-ron, khác với phương pháp truyền thống sử dụng lan truyền ngược. Mặc dù phương pháp này vượt qua điểm số của con người trong ba game Atari nhưng còn nhiều hạn chế. Một trong những hạn chế đó là đặc tính của lập trình tiến hóa hoạt động giống như một "hộp đen", nghĩa là chúng ta không thể dễ dàng quan sát, cũng như kiểm soát cách tìm kiếm những trọng số của mạng nơ-ron như đã làm với lan truyền ngược [14].

Trong những năm gần đây, học sâu đạt đươc nhiều bước đột phá trong nhiều lĩnh vực như Thị giác máy tính, Nhận diện giọng nói... Việc kết hợp học tăng cường với học sâu đã dẫn đến một hướng tiếp cận mới cho bài toán tự động chơi game; hướng tiếp cận này giúp hệ thống có thể học cách chơi nhiều game khác nhau từ hình ảnh đầu vào RGB [13]. Với học sâu, ta có thể học được những đặc trưng cấp cao (high level features) từ hình ảnh thô mà không cần phải tự thiết kế đặc trưng bằng tay (hand-designed features). Khi kết hợp với học tăng cường, ta có một hình "**End-to-end**": việc học đặc trưng và học chiến thuật chơi được liên kết chặt chẽ với nhau. Trong khoá luận này, chúng em thực hiện việc cài đặt lại phương pháp học tăng cường sâu và thử nghiệm mô hình với những tham số khác nhau. Cùng với đó, khoá luận tìm hiểu những khó khăn cơ bản của bài toán tự động chơi game cũng như thử nghiệm lại các đề xuất nhằm giải quyết những khó khăn này.

Chương 2

Kiến thức nền tảng

Chương này trình bày những kiến thức nền tảng của học tăng cường. Trong phần đầu tiên, chúng em trình bày định nghĩa của các thành phần cơ bản trong học tăng cường. Tiếp đó, mô hình "Markov Decision Processes" được giới thiệu; mô hình này dùng để mô hình hoá các bài toán học tăng cường khác nhau về chung một "framework" cố định. Từ đó, ta có thể giải các bài toán học tăng cường bằng cách giải bài toán trên mô hình này. Cuối cùng, chúng em trình bày thuật toán "Q-learning" - thuật toán học tăng cường được chúng em sử dụng trong bài toán tự động chơi game.

2.1 Các thành phần cơ bản của học tăng cường

2.1.1 Hệ thống và môi trường

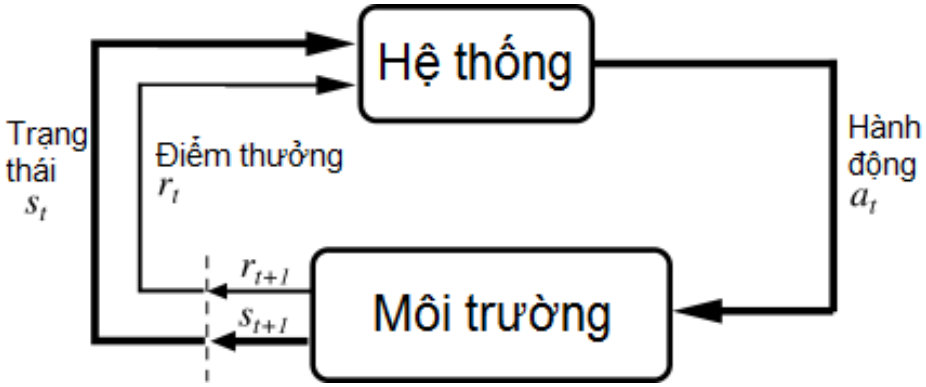
Trong học tăng cường, thành phần cần "học" và đưa ra quyết định được gọi chung là *hệ thống* (agent). Thành phần này tương tác trực tiếp tới các đối tượng "bên ngoài" được gọi là *môi trường* (environment). Một cách đơn giản, môi trường ở đây chính là những thành phần "ẩn" mà hệ thống không kiểm soát trực tiếp được; để thay đổi môi trường, hệ thống phải tương tác với môi trường bằng cách thực hiện hành động. Sự tương tác này diễn ra tại các mốc thời gian nhất định. Tại mỗi thời điểm, hệ thống lựa chọn hành động dựa trên những thông tin nhận được từ môi trường. Những thông tin này bao gồm:

- **Trạng thái** (state): những thông tin về môi trường xung quanh mà hệ thống nhận được. Ví dụ trong đánh cờ, trạng thái có thể là vị trí những quân cờ đang có trên bàn cờ. Môi trường trong bài toán học tăng cường có thể mang yếu tố ngẫu nhiên (stochastic environment): một hành động có thể có dẫn đến những kết quả khác nhau. Ví dụ như trong trò chơi đổ xúc xắc, giá trị của xúc xắc là hoàn toàn ngẫu nhiên. Vì vậy, nếu môi trường thay đổi dựa vào kết quả của xúc xắc thì trạng thái cũng thay đổi một cách ngẫu nhiên. Ký hiệu của trạng thái là s.

- **Điểm thưởng** (reward): giá trị số thực mà môi trường trả về cho hệ thống; giá trị điểm thưởng này nói lên độ tốt của một hành động tại một trạng thái nào đó. Với ví dụ đánh cờ, điểm thưởng mà hệ thống có thể nhận được từ môi trường ở cuối ván cờ là +1 nếu hệ thống thắng, -1 nếu hệ thống thua. Trong quá trình đánh cờ, điểm thưởng có thể là 0 cho mỗi nước cờ mà hệ thống thực hiện. Có thể thấy trong ví dụ này, điểm thưởng không nói rõ nước cờ nào là tốt mà chỉ đánh giá tổng quát cho cả ván đấu. Chính vì vậy, điểm thưởng có ý nghĩa khác biệt so với nhãn của phương pháp học có giám sát. Tương tự như trạng thái, điểm thưởng cũng có thể ngẫu nhiên. Với ví dụ xúc xắc, nếu điểm thưởng là giá trị của xúc xắc thì điểm thưởng là ngẫu nhiên. Nếu ta chơi hai lần và thực hiện cùng một hành động tại cùng một trạng thái thì điểm thưởng vẫn có thể khác nhau. Điểm thưởng được ký hiệu là r.

Từ trạng thái và điểm thưởng nhận được, hệ thống chọn hành động phù hợp để đạt được điểm thưởng cao nhất.

Hệ thống và môi trường tương tác với nhau một cách tuần tự. Thời điểm tương tác (được ký hiệu là t) được đánh số tăng dần: $t = 0, 1, 2, ..., T$ (với T là thời điểm kết thúc). Tại thời điểm t, hệ thống nhận được trạng thái hiện tại $S_t \in \mathcal{S}$, với $\mathcal{S}$ là tập các trạng thái có thể có. Dựa vào trạng thái nhận được, hệ thống thực hiện một hành động $A_t \in \mathcal{A}$, trong đó $\mathcal{A}$ là tập các hành động có thể thực hiện. Hành động này làm cho môi trường thay đổi. Tại thời điểm $t+1$ sau đó, hệ thống nhận được giá trị điểm thưởng $R_{t+1} \in \mathbb{R}$ cùng với trạng thái tiếp theo S_{t+1}. Giá trị R_{t+1} là điểm thưởng nhận được do hành động A_t vừa làm



Hình 2.1: Quá trình tương tác giữa hệ thống và môi trường

thay đổi môi trường (chứ không phải tổng điểm thưởng từ lúc bắt đầu). Quá trình tương tác giữa hệ thống và môi trường cứ tiếp tục lặp lại cho đến khi môi trường trả về trạng thái kết thúc. Quá trình tương tác này được mô tả trong hình 2.1.

Các thành phần của hệ thống gồm có:

- **Chính sách** π là cách hệ thống lựa chọn hành động dựa vào trạng thái hiện tại. Tại thời điểm t, khả năng một hành động a được chọn ở trạng thái s là $\pi(a \mid s) = \mathbb{P}[A_t = a \mid S_t = s]$. Nếu chính sách luôn chọn một hành động với xác suất là 1 thì chính sách đó gọi là *đơn định* (deterministic). Chính sách đơn định lúc này có thể coi là một ánh xạ từ tập trạng thái sang tập hành động: $a = \pi(s)$. Để đạt nhiều điểm thưởng nhất, hệ thống cần có một chính sách chọn lựa hành động phù hợp. Những phương pháp học tăng cường thường tập trung thay đổi dần dần các chính sách của hệ thống để đạt được mục tiêu trên.

- **Hàm giá trị** dùng để đánh giá trạng thái hoặc hành động tại trạng thái là "tốt" đến mức nào. Để đánh giá một trạng thái, ta có thể thử tương tác với môi trường và tính tổng điểm thưởng nhận được từ khi gặp trạng thái đó cho đến khi kết thúc. Tuy nhiên, do môi trường có thể mang tính ngẫu nhiên, tổng điểm thưởng này có thể khác nhau ở những lần tương tác khác nhau. Vì vậy, giá trị trạng thái được định nghĩa là **kỳ vọng** tổng điểm thưởng (lấy trung bình vô số lần tương tác). Lúc này, giá trị của một trạng thái s theo chính sách π được ký hiệu $v_\pi(s)$ là kỳ vọng tổng điểm

thưởng mà hệ thống có thể nhận được bắt đầu từ trạng thái s về sau. Lý do có chính sách π trong định nghĩa này là do điểm thưởng nhận được sẽ phụ thuộc vào cách hệ thống chọn hành động. Tương tự, để đánh giá một hành động tại một trạng thái, ta định nghĩa hàm giá trị hành động. Giá trị của việc thực hiện hành động a tại trạng thái s rồi tương tác theo chính sách π được ký hiệu $q_\pi(s, a)$. Giá trị này thể hiện là kỳ vọng tổng điểm thưởng mà hệ thống có thể nhận được kể từ sau khi thực hiện hành động a tại trạng thái s và thực hiện các hành động tiếp theo dựa vào chính sách π.

- **Mô hình**. Trong một số bài toán học tăng cường, hệ thống có thể xây dựng mô hình cho riêng mình để mô phỏng lại môi trường. Qua đó cho phép hệ thống có thể suy luận hoặc dự đoán những thông tin của môi trường trong tương lai.

2.1.2 Tổng điểm thưởng

Tổng điểm thưởng (return) G_t xác định lượng điểm thưởng mà hệ thống nhận được kể từ thời điểm t trở đi. Ta có thể định nghĩa giá trị này bằng:

$$G_t = R_{t+1} + R_{t+2} + ... + R_T \tag{2.1}$$

ở đây T là thời điểm cuối cuối cùng hệ thống tương tác với môi trường. Tuy nhiên, tổng này có thể không hữu hạn nếu như độ dài một lần tương tác không được giới hạn.

Từ đây, ta có thể sử dụng một định nghĩa tổng điểm thưởng luôn hữu hạn:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... + \gamma^{T-t-1} R_T = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.2}$$

Trong đó γ là hệ số "giảm điểm thưởng" (discount factor) với giá trị $0 \leqslant \gamma \leqslant 1$. Đây là định nghĩa tổng điểm thưởng được dùng trong các bài toán học tăng cường. Lý do chính để các nhà nghiên cứu chọn định nghĩa này là để đơn giản hoá về mặt toán học. Tuy vậy, định nghĩa này cũng có một số lợi ích khác. Nếu

như cần dự đoán giá trị tổng điểm thưởng tại thời điểm t (một việc mà ta sẽ phải làm thường xuyên), các giá trị điểm thưởng ở càng xa thì khả năng ta dự đoán đúng càng thấp. Vì vậy, trọng số của các điểm thưởng ở càng xa thời điểm hiện tại cần phải giảm dần. Với $\gamma = 0$, hệ thống chỉ quan tâm điểm thưởng ngay lập tức mà không quan tâm về lâu dài. Ngược lại, γ càng gần 1 thì hệ thống càng quan tâm một cách dài hạn hơn về điểm thưởng.

2.2 Mô hình Markov Decision Processes

2.2.1 Định nghĩa mô hình Markov Decision Processes

Mô hình Markov Decision Processes (MDP) được sử dụng để mô hình hóa bài toán học tăng cường một cách hình thức. Cụ thể, MDP là một bộ bao gồm 5 thành phần $< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma >$:

- $\mathcal{S}$: tập trạng thái hữu hạn có thể có của môi trường.
- $\mathcal{A}$: tập hữu hạn những hành động mà hệ thống có thể thực hiện để tương tác với môi trường. Tổng quát hơn, tại mỗi trạng thái, hệ thống có thể thực hiện những hành động khác nhau. Khi đó, tập hành động tại trạng thái s được ký hiệu là $\mathcal{A}(s)$.
- γ: hệ số giảm điểm thưởng có giá trị thỏa $0 \leqslant \gamma \leqslant 1$ thể hiện mức độ tin tưởng về giá trị điểm thưởng nhận được ở tương lai.
- $\mathcal{P}$: ma trận xác suất chuyển trạng thái. Trong đó $\mathcal{P}^a_{ss'}$ là xác suất chuyển đến trạng thái s' khi hệ thống đang ở trạng thái s và thực hiện hành động a.

$$\mathcal{P}^a_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a] \tag{2.3}$$

- $\mathcal{R}$: ma trận điểm thưởng của từng *bộ (trạng thái, hành động)*. $\mathcal{R}^a_s$ là giá trị kỳ vọng điểm thưởng nhận được **ngay lập tức** khi hệ thống thực hiện hành động a ở trạng thái s.

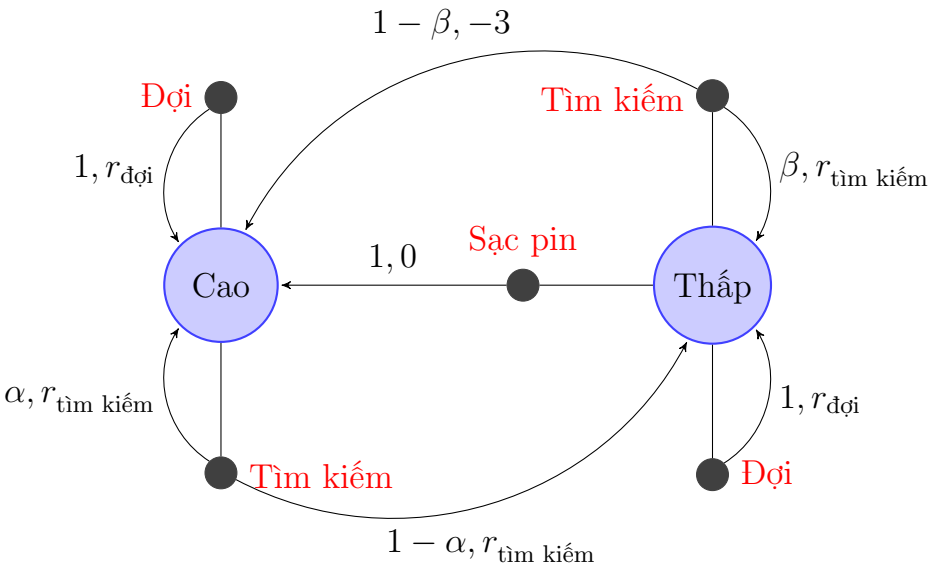
$$\mathcal{R}^a_s = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a] \tag{2.4}$$

Ví dụ mô hình MDP của bài toán robot thu gom soda (ví dụ chỉnh sửa từ [16]): công việc của robot này là thu lượm những lon soda đã được uống hết trong văn phòng. Robot có những cảm biến để xác định những lon soda này, bánh xe và cánh tay để di chuyển và gắp nhặt những lon này bỏ vào thùng. Robot hoạt động bằng pin sạc. Hệ thống điều khiển của robot có chức năng tiếp nhận những thông tin từ cảm biến để điều khiển bánh xe và cánh tay. Trong ví dụ, chúng em chỉ xét dựa trên mức độ pin hiện tại robot nên quyết định tìm kiếm những lon soda như thế nào. Tại mỗi thời điểm, robot có thể thực hiện một trong ba hành động: (1) thực hiện tìm kiếm một lon soda, (2) đứng yên và đợi người khác mang lon soda đến cho nó, (3) quay trở lại nơi sạc pin. Trạng thái của môi trường được xác định là lượng pin hiện tại của robot. Cách tốt nhất để tìm kiếm những lon soda là robot thực hiện hành động tìm kiếm, nhưng việc này sẽ làm giảm dung lượng của pin. Ngược lại nếu robot đứng yên và đợi thì dung lượng pin không giảm. Mỗi khi dung lượng pin ở mức thấp thì robot sẽ quay lại chỗ sạc pin. Trường hợp xấu nhất có thể xảy ra là robot không đủ dung lượng pin để quay lại nơi sạc; khi đó robot sẽ đứng yên và đợi ai đó mang nó đến chỗ sạc. Do đó robot cần có một chiến lược phù hợp để đạt được hiệu năng cao nhất có thể. Hệ thống đưa ra những quyết định dựa trên mức năng lượng pin. Mức năng lượng này có thể được xác định hai mức *cao* và *thấp*. Khi đó tập trạng thái mà hệ thống có thể nhận được là $\mathcal{S} = \{\text{cao}, \text{thấp}\}$. Những hành động của hệ thống trong ví dụ này được xét đơn giản gồm ba hành động *đợi*, *tìm kiếm* và *sạc pin*. Khi dung lượng pin ở trạng thái cao, hệ thống chỉ thực hiện hai hành động: tìm kiếm và đợi. Ngược lại khi ở trạng thái thấp, hệ thống có thể thực hiện ba hành động: tìm kiếm, đợi, và sạc pin.

$$\mathcal{A}(\text{cao}) = \{\text{tìm kiếm}, \text{đợi}\}$$

$$\mathcal{A}(\text{thấp}) = \{\text{tìm kiếm}, \text{đợi}, \text{sạc pin}\}$$

Khi mức năng lượng pin ở mức cao, nếu robot thực hiện tìm kiếm sẽ có xác suất α năng lượng pin vẫn ở mức cao, và $1-\alpha$ năng lượng của pin sẽ chuyển về mức thấp. Mặt khác, khi mức năng lượng ở mức thấp, nếu robot thực hiện tìm kiếm sẽ có xác suất β năng lượng pin ở mức thấp, và $1-\beta$ chuyển đến mức cao



Hình 2.2: Đồ thị minh họa việc chuyển trạng thái cho robot thu gom. Trong đồ thị có hai loại node: node trạng thái và node hành động. Node trạng thái minh họa những trạng thái mà hệ thống có thể nhận được và được ký hiệu là một vòng tròn lớn với tên của trạng thái bên trong. Node hành động tương ứng với cặp (trạng thái, hành động). Việc thực hiện hành động a tại trạng thái s tương ứng trên đồ thị là một cạnh bắt đầu từ node trạng thái s tới node hành động a. Khi đó môi trường sẽ trả ra trạng thái tiếp theo s' ứng với đích của mũi tên đi từ node hành động a. Giá trị được ký hiệu trên mỗi mũi tên lần lượt là xác suất chuyển tới trạng thái mới và giá trị điểm thưởng kỳ vọng nhận được. Ví dụ: khi mức năng lượng pin đang ở trạng thái *thấp*, hệ thống quyết định thực hiện hành động *sạc pin* thì trạng thái tiếp theo mà hệ thống nhận được sẽ là mức năng lượng pin *cao* (với xác suất là 1) và giá trị kỳ vọng điểm thưởng là 0.

(trường hợp này xảy ra khi dung lượng pin cạn kiệt và cần ai đó mang robot đến chỗ sạc). Ngoài ra, mỗi lon soda nhặt được tương ứng với $+1$ điểm thưởng và sẽ bị -3 điểm thưởng mỗi khi robot phải cần ai đó mang đến chỗ sạc. $r_{\text{đợi}}$, $r_{\text{tìm kiếm}}$ là số lượng lon soda kỳ vọng mà robot có thể thu gom được trong khi đợi và tìm kiếm. Hình 2.2 minh họa cho mô hình MDP trong ví dụ robot thu gom lon soda.

2.2.2 Chính sách và hàm giá trị

Có thể nói chính sách như "bộ não" của hệ thống: chính sách quyết định cách thức mà hệ thống hành động trong những trạng thái cụ thể. Do đó, hệ thống

có được một chính sách tốt đồng nghĩa với việc khả năng ra quyết định của hệ thống trở nên tốt và thông minh hơn.

Hàm giá trị cho biết những trạng thái hoặc những cặp hành động và trạng thái "tốt" đến mức thế nào. Khái niệm "tốt" ở đây nghĩa là giá trị kỳ vọng tổng điểm thưởng mà hệ thống có thể nhận được cao đến mức nào. Hầu hết các thuật toán trong học tăng cường đều tập trung vào việc đánh giá những hàm giá trị và qua đó cải thiện chính sách trở nên tốt hơn [16]. Tổng điểm thưởng mà hệ thống có thể nhận được trong tương lai phụ thuộc vào cách thức chọn hành động. Do đó, hàm giá trị chịu ảnh hưởng rất nhiều vào chính sách. Giá trị của trạng thái s dưới một chính sách π, ký hiệu $v_\pi(s)$, là kỳ vọng tổng điểm thưởng nhận được khi bắt đầu từ trạng thái s và thực hiện theo chính sách π sau đó. Với mô hình MDP, $v_\pi(s)$ được định nghĩa như sau:

$$v_\pi(s) = \mathbb{E}_\pi \left[G_t \mid S_t = s \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s \right] \tag{2.5}$$

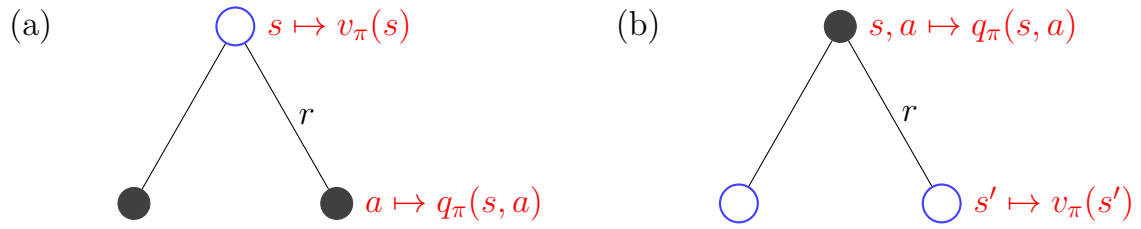
v_π được gọi là hàm giá trị trạng thái dưới chính sách π. Chính sách π được ghi dưới ký hiệu kỳ vọng thể hiện việc hàm giá trị này phụ thuộc vào π.

Tương tự, chúng ta định nghĩa giá trị hành động a tại trạng thái s, được ký hiệu $q_\pi(s, a)$, là giá trị kỳ vọng của tổng điểm thưởng khi thực hiện hành động a trong trạng thái s rồi sau đó chọn hành động theo chính sách π:

$$q_\pi(s, a) = \mathbb{E}_\pi \left[G_t \mid S_t = s, A_t = a \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s, A_t = a \right] \tag{2.6}$$

q_π được gọi là hàm giá trị hành động dưới chính sách π.

Từ các định nghĩa trên, ta có thể tìm mối liên hệ giữa giá trị trạng thái và giá trị hành động. Phương trình (2.7) xác định giá trị của một trạng thái bằng tổng có trọng số của các giá trị hành động tại trạng thái đó (với trọng số là xác suất thực hiện hành động). Hình 2.3a minh họa quan hệ giữa giá trị của một trạng thái s và giá trị của các hành động có thể thực hiện tại trạng thái đó. Hình 2.3b cho thấy từ việc thực hiện hành động a tại trạng thái s, trạng thái tiếp theo s' có thể khác nhau. Do đó giá trị của hành động a ở trạng thái



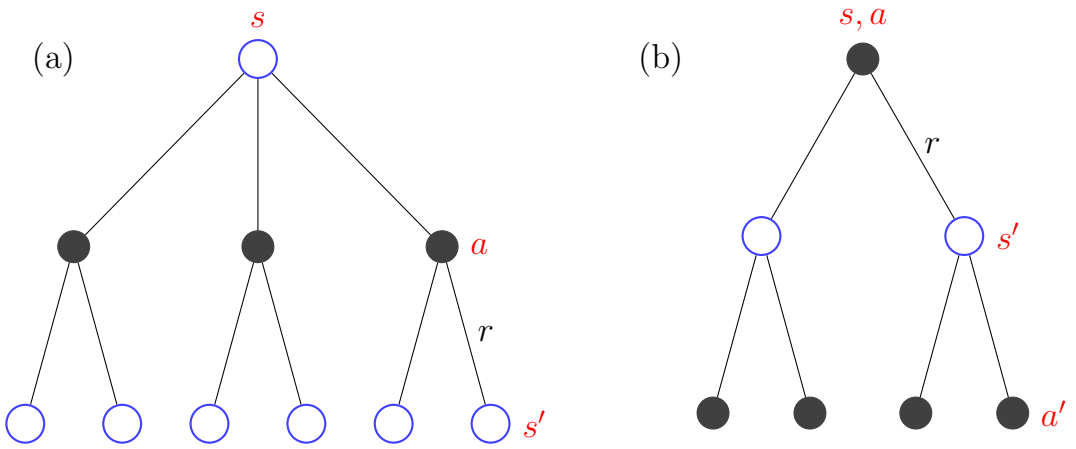
Hình 2.3: Đồ thị minh họa quan hệ giữa hàm giá trị trạng thái và hàm giá trị hành động. Node tròn rỗng ứng với trạng thái và node tròn tô đậm ứng với hành động tại trạng thái đó. Với hình (a), giá trị trạng thái s là $v_\pi(s)$. Tại trạng thái này, hệ thống có thể thực hiện **một trong số** các hành động ở ngay bên dưới. Xác suất chọn hành động này được thể hiện qua chính sách π. Vì vậy, giá trị trạng thái s chính là tổng có trọng số của giá trị từng hành động (với trọng số là xác suất thực hiện hành động đó). Với hình (b), giá trị hành động a tại trạng thái s là $q_\pi(s, a)$. Khi thực hiện hành động a, hệ thống nhận được điểm thưởng r và di chuyển sang trạng thái mới. Trạng thái mới là ngẫu nhiên nên để tính giá trị hành động, ta phải tính tổng có trọng số giá trị của mọi trạng thái tiếp theo có thể có (với trọng số là xác suất đi đến trạng thái đó).

s chính là kỳ vọng điểm thưởng nhận được ngay sau đó cộng với tổng giá trị kỳ vọng của các trạng thái tiếp theo đó đã được nhân với hệ số γ. Phương trình (2.8) thể hiện mối quan hệ này giữa giá trị hành động và giá trị trạng thái.

$$v_\pi = \sum_{a \in \mathcal{A}(s)} \pi(a \mid s) q_\pi(s, a) \tag{2.7}$$

$$q_\pi(s, a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s') \tag{2.8}$$

Hàm giá trị có một tính chất cơ bản thường được áp dụng trong học tăng cường đó là mối quan hệ đệ quy. Cho bất kỳ chính sách π với bất kỳ trạng thái



Hình 2.4: Đồ thị minh họa phương trình Bellman cho (a) v_π và (b) q_π.

s, giá trị của trạng thái đó được xác định bằng:

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi \left[G_t \mid S_t = s \right] \\ &= \mathbb{E}_\pi \left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... \mid S_t = s \right] \\ &= \mathbb{E}_\pi \left[R_{t+1} + \gamma (R_{t+2} + \gamma R_{t+3} + ...) \mid S_t = s \right] \\ &= \mathbb{E}_\pi \left[R_{t+1} + \gamma G_{t+1} \mid S_t = s \right] \\ &= \mathbb{E}_\pi \left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s \right] \end{aligned} \tag{2.9}$$

Phương trình (2.9) được gọi là phương trình Bellman cho v_π. Từ phương trình này ta thấy được mối liên quan giữa giá trị của một trạng thái s bất kỳ và giá trị của những trạng thái tiếp. Ý tưởng chính của phương trình Bellman đó là "nhìn trước một bước": đánh giá trạng thái hiện tại bằng cách nhìn trước tất cả những trạng thái tiếp theo có thể đạt được từ trạng thái đó. Ý tưởng này được được minh họa trong hình 2.4a. Từ một trạng thái, môi trường có thể trả ra trạng thái tiếp theo s' khác nhau. Phương trình (2.9) sẽ trung bình tất cả các trường hợp có thể xảy ra lại theo xác suất mà chúng xuất hiện. Phương trình này cũng cho thấy giá trị của một trạng thái phải bằng kỳ vọng **điểm thưởng tức thì** cộng với **giá trị của những trạng thái tiếp theo**.

$$q_\pi(s, a) = \mathbb{E}_\pi \left[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a \right] \tag{2.10}$$

Phân tích phương trình (2.6) tương tự như đã làm đối với hàm giá trị trạng

thái, ta có được phương trình (2.10). Hình 2.4b minh họa ý tưởng nhìn trước một bước để tính giá trị của một hành động ở trạng thái hiện tại. Thực hiện một hành động a ở trạng thái s, trạng thái tiếp theo s' có thể khác nhau và điểm thưởng r cũng có thể khác nhau. Trong mỗi trạng thái s' lại có nhiều hành động a' khác nhau có thể thực hiện. Phương trình (2.10) sẽ trung bình tất cả các trường hợp có thể đó lại theo xác suất mà chúng được thực hiện. Hay nói cách khác, phương trình (2.10) cho thấy giá trị của một hành động a tại trạng thái s, $q_\pi(a, s)$ được xác định bằng kỳ vọng điểm thưởng tức thì cộng với giá trị hành động tại trạng thái kế tiếp.

2.2.3 Chính sách tối ưu và hàm giá trị tối ưu

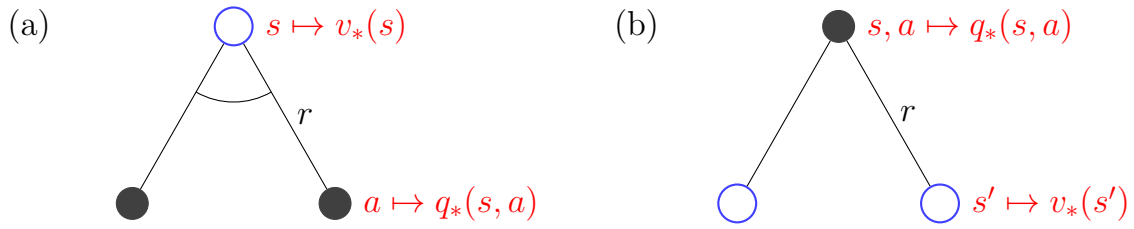
Để giải bài toán học tăng cường, chúng ta cần tìm một chính sách sao cho hệ thống có thể đạt được nhiều điểm thưởng nhất có thể. Một chính sách π được xác định là tốt hơn hoặc bằng chính sách π' khi với một trạng thái s bất kỳ, giá trị của trạng thái này theo chính sách π luôn lớn hơn hoặc bằng giá trị của trạng thái đó theo chính sách π':

$$\pi \geq \pi' \iff v_\pi(s) \geq v_{\pi'}(s), \forall s \in \mathcal{S} \tag{2.11}$$

Luôn có ít nhất một chính sách tốt hơn hoặc bằng tất cả các chính sách còn lại [16]. Các chính sách này được gọi chung là *chính sách tối ưu* và được ký hiệu π_*. Những chính sách tối ưu đều cùng có chung một hàm giá trị trạng thái và hàm giá trị hành động. Do vậy, ta ký hiệu $v_*(s)$ và $q_*(s, a)$ là hàm giá trị tối ưu thay vì $v_{\pi_*}(s)$ và $q_{\pi_*}(s, a)$. Hai loại hàm giá trị này có thể được gọi chung là *hàm giá trị tối ưu*. Chúng ta cũng có thể gọi tách biệt *hàm giá trị trạng thái tối ưu* và *hàm giá trị hành động tối ưu*. Phương trình (2.12) và (2.13) định nghĩa hình thức cho hai loại hàm này

$$v_*(s) = \max_\pi v_\pi(s), \forall s \in \mathcal{S} \tag{2.12}$$

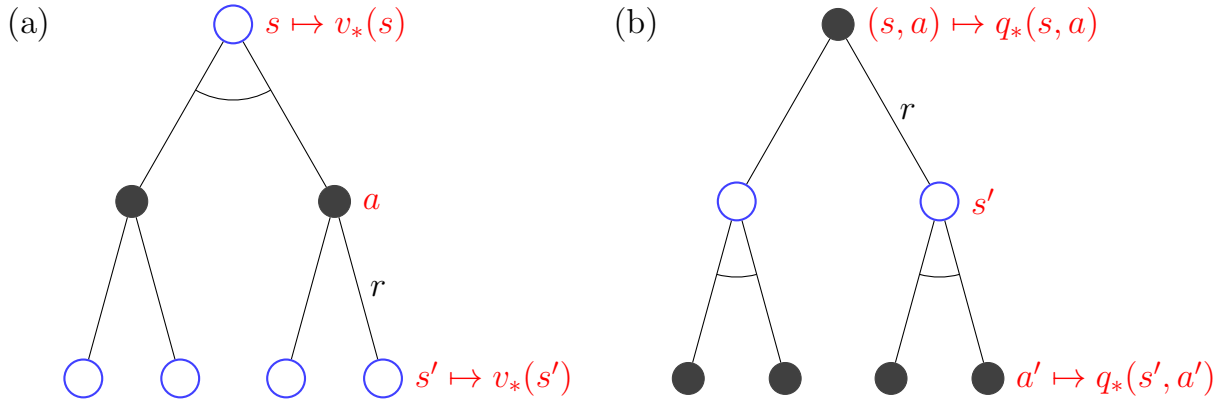
$$q_*(s, a) = \max_\pi q_\pi(s, a), \forall s \in \mathcal{S} \text{ và } \forall a \in \mathcal{A}(s) \tag{2.13}$$



Hình 2.5: Đồ thị minh hoạ quan hệ giữa hàm giá trị trạng thái tối ưu và hàm giá trị hành động tối ưu. Đồ thị bên trái thể hiện cách tính giá trị trạng thái tối ưu dựa vào giá trị hành động tối ưu. Đồ thị này cũng tương tự như giữa hàm giá trị trạng thái với hàm giá trị hành động. Khác biệt lúc này là có cung nối giữa các hành động thể hiện việc lấy max giá trị của các hành động. Đồ thị bên phải thể hiện cách tính giá trị hành động tối ưu dựa vào giá trị trạng thái tối ưu.

Từ hai phương trình (2.12) và (2.13), ta thấy rằng để xác định hàm giá trị tối ưu của mỗi trạng thái s hoặc cặp trạng thái và hành động (s, a), ta cần thử đánh giá giá trị của chúng theo tất cả các chính sách có thể có và chọn giá trị cao nhất.

Hình 2.5 minh hoạ quan hệ giữa hàm giá trị trạng thái tối ưu và hàm giá trị hành động tối ưu; khi có được hàm này ta dễ dàng có được hàm còn lại. Trong hình 2.5a, ta có thể xác định giá trị tối ưu cho trạng thái s dựa trên giá trị hành động tối ưu. Phương trình (2.14) xác định giá trị tối ưu cho trạng thái s bằng cách chọn giá trị hành động tối ưu lớn nhất trong các hành động có thể thực hiện. Tương tự trong hình 2.5b, ta có thể xác định giá trị tối ưu cho hành động a ở trạng thái s dựa trên hàm giá trị trạng thái tối ưu của các trạng thái kế tiếp. Phương trình (2.15) xác định giá trị tối ưu của hành động a tại trạng thái s gồm hai phần: kỳ vọng điểm thưởng nhận được tức thì và giá trị trạng thái tối ưu của trạng thái kế tiếp.



Hình 2.6: Đồ thị minh họa phương trình Bellman trong (a) v_* và (b) q_*

$$v_*(s) = \max_{a \in \mathcal{A}(s)} q_*(s,a) \tag{2.14}$$

$$q_*(s,a) = R_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s') \tag{2.15}$$

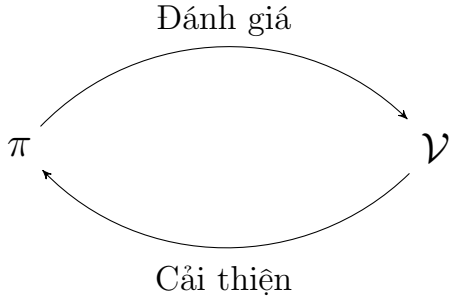
$$v_*(s) = \max_{a \in \mathcal{A}(s)} R_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s') \tag{2.16}$$

$$q_*(s,a) = R_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \max_{a' \in \mathcal{A}(s')} q_*(s',a') \tag{2.17}$$

Phương trình (2.16) và (2.17) dễ dàng có được bằng cách thay thế hai phương trình (2.14) và (2.15) qua lại lẫn nhau. Từ hai phương trình này, ta thấy được dạng phương trình Bellman trong hàm giá trị trạng thái tối ưu và hàm giá trị hành động tối ưu. Hình 2.6 minh họa ý tưởng nhìn trước một bước của phương trình Bellman trong hàm giá trị tối ưu. Trong đó hình 2.6a minh họa cách thức xác định giá trị tối ưu cho một trạng thái ứng với phương trình (2.16). Hình 2.6b minh họa cách thức xác định giá trị tối ưu của một hành động ở một trạng thái ứng với phương trình (2.17).

2.2.4 Phương pháp lặp chính sách

Trong các bài toán học tăng cường, mục tiêu chính của ta là tìm được chính sách tối ưu π_* nhằm giúp cho hệ thông giải quyết bài toán tốt nhất có thể. Do



Hình 2.7: Quy trình tìm chính chính sách tối ưu bằng phương pháp lặp chính sách

đó ta cần có một quy trình để tìm ra chính sách tối ưu. Quy trình lặp chính sách là một trong hai quy trình thường được áp dụng để tìm kiếm chính sách này. Hình 2.7 minh họa quy trình này được chia thành hai giai đoạn:

- **Đánh giá chính sách**: Việc đánh giá một chính sách π được thực hiện bằng cách xác định hàm giá trị trạng thái hoặc hàm giá trị hành động của dưới chính sách đó.
- **Cải thiện chính sách**: Sau khi có được hàm giá trị của một chính sách π, chính sách cải thiện mới π' được tạo ra bằng cách thực hiện tham lam trên hàm giá trị của chính sách π, tức là chỉ chọn thực hiện hành động có giá trị cao nhất dựa trên hàm giá trị hành động có được.

Một chính sách π_1 được cải thiện từ chính sách π_0 dựa trên hàm giá trị trạng thái v_{π_0}. Khi có được chính sách π_1 ta có thể tính được hàm giá trị v_{π_1} qua đó tiếp tục cải thiện để có được chính sách π_2. Quá trình này diễn ra cho đến khi đạt được chính sách tối ưu.

$$\pi_0 \xrightarrow{\text{Đánh giá}} v_{\pi_0} \xrightarrow{\text{Cải thiện}} \pi_1 \xrightarrow{\text{Đánh giá}} v_{\pi_1} \xrightarrow{\text{Cải thiện}} \pi_2 \cdots \xrightarrow{\text{Cải thiện}} \pi_* \xrightarrow{\text{Đánh giá}} v_{\pi_*}$$

2.2.4.1 Đánh giá chính sách bằng quy hoạch động (Dynamic Programming)

Quy hoạch động thường được dùng để giải quyết các bài toán tối ưu mà dữ liệu có tính thứ tự, ví dụ như dữ liệu chuỗi hay dữ liệu thời gian. Một bài toán tối ưu có thể được giải quyết bằng quy hoạch động cần có hai đặc điểm:

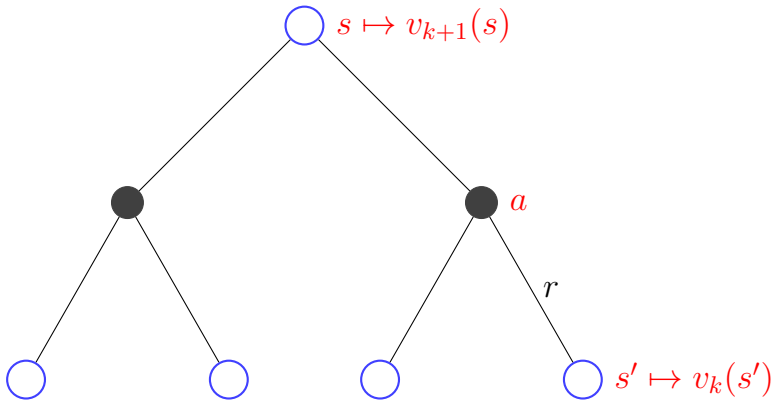
- Quy tặc tối ưu (Principle of Optimality): các bài toán có thể phân rã thành các bài toán con, và kết quả của bài toán con này đóng góp vào lời giải của bài toán gốc.

- Các bài toán con chồng lấn lên nhau và lặp lại nhiều lần: nhằm tận dụng lại kết quả của những bài toán con đã tính toán trước đó.

Trong nhiều bài toán học tăng cường, kỹ thuật quy hoạch động được dùng để tìm chính sách tối ưu hoặc tối ưu hàm giá trị. Để có thể áp dụng kỹ thuật quy hoạch động, những bài toán này cũng cần phải thỏa yêu cầu là hệ thống có kiến thức đầy đủ về môi trường hay cách khác môi trường có mô hình MDP. Quy hoạch động xác định hàm giá trị của một chính sách bằng cách cập nhật hàm giá trị được khởi tạo bất kỳ ban đầu qua nhiều vòng lặp, dựa vào phương trình Bellman. Ý tưởng của cách xác định này như sau: Ban đầu khởi tạo hàm giá trị v_0 bất kỳ cho tất cả các trạng thái, trừ trạng thái kết thúc được luôn có giá trị là 0. Tiến hành cập nhật hàm giá trị mới v_1 cho chính sách dựa trên hàm giá trị v_0 theo phương trình 2.18. Tương tự cập nhật hàm giá trị mới v_2 dựa trên v_1. Quá trình lặp cho đến khi độ khác biệt giữa hàm giá trị sau và giá trị trước đó nhỏ hơn một lượng cho trước. Quy trình cập nhật được minh họa trong hình 2.8, trong đó giá trị mới v_{k+1} của trạng thái s được xác định dựa trên giá trị kỳ vọng điểm thưởng nhận được theo chính sách π, và giá trị hiện tại v_k của các trạng thái s' kế tiếp trạng thái s. Tổng thể của việc đánh giá chính sách bằng quy hoạch động được trình bày ở thuật toán 2.1

$$v_{k+1}(s) \leftarrow \sum_{a \in \mathcal{A}} \pi(a \mid s)(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s')) \tag{2.18}$$

Mặc dù đã quy hoạch động đã được chứng minh là xấp xỉ tốt hay thậm chí là tìm được hàm giá trị trạng thái của chính sách π [7], nhưng trong các bài toán học thực tế của học tăng cường đặc biệt là những bài toán lớn thì quy hoạch động trở nên không khả thi do chi phí tính toán cao, trong trường hợp xấu nhất chi phí tính toán thuộc $O(k^n)$ với k là số hành động và n là số trạng thái.



Hình 2.8: Đồ thị minh họa cập nhật hàm giá trị bằng quy hoạch động

Thuật toán 2.1 Xác định hàm giá trị bằng quy hoạch động

Đầu vào: Chính sách π cần đánh giá
Đầu ra: Hàm giá trị V xấp xỉ hàm giá trị v_π của chính sách π
Thao tác:

1: Khởi tạo ngẫu nhiên $V(s)$ cho tất cả trạng thái s không phải trạng thái kết thúc. Nếu s là trạng thái kết thúc, $V(s) = 0$
2: **repeat**
3: $\Delta \leftarrow 0$ %% Tính độ khác biệt giữa hàm giá trị cũ và giá trị mới. Độ lớn của Δ được xác định là độ khác biệt lớn nhất giữa giá trị cũ và giá trị mới của một trạng thái trong tất cả các trạng thái.
4: **for** $s \in \mathcal{S}$ **do** %% Với mỗi trạng thái
5: $v \leftarrow V(s)$ %% Lưu giá trị hiện tại của trạng thái s
6: $V(s) \leftarrow \sum_{a \in \mathcal{A}} \pi(a \mid s)(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a V(s'))$ %% Tính giá trị mới cho trạng thái s dựa trên giá trị hiện tại của các trạng thái s' kế tiếp của trạng thái s, và giá trị kỳ vọng của các hành động tại trạng thái đó theo chính sách π.
7: $\Delta \leftarrow \max(\Delta, |v - V(s)|)$ %% Cập nhật giá trị mới cho Δ
8: **end for**
9: **until** $\Delta < \theta$ (Một lượng đủ nhỏ)

2.2.4.2 Cải thiện chính sách bằng phương pháp tham lam (greedy)

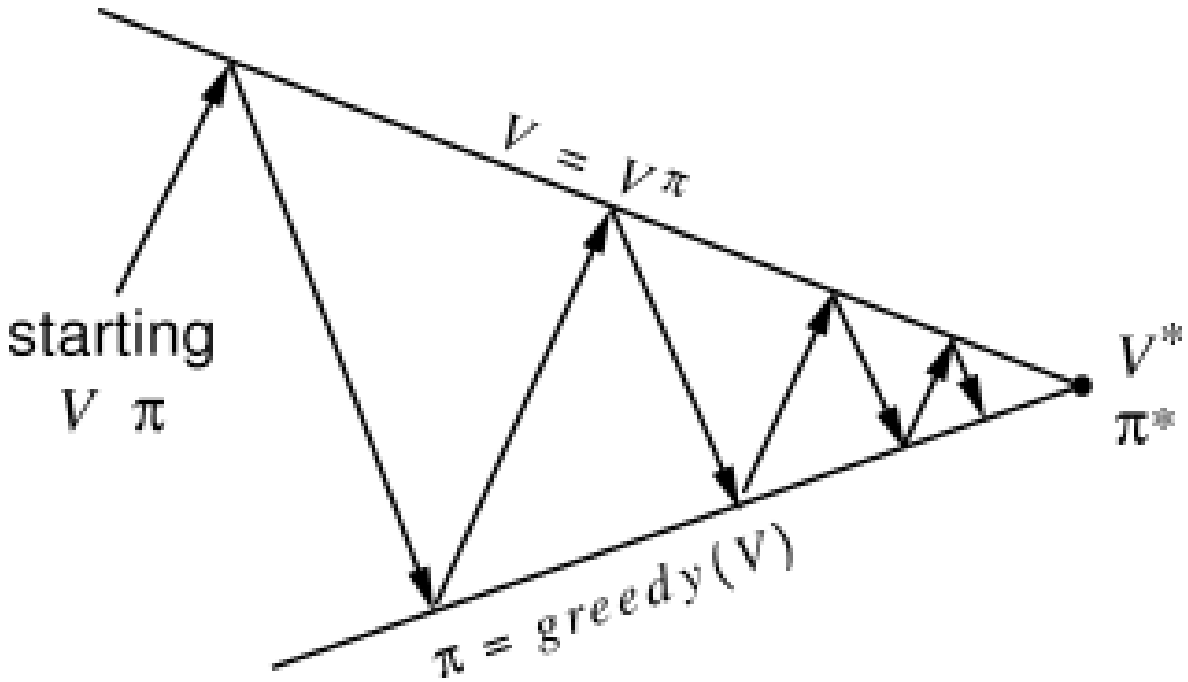
Mục tiêu chúng ta xác định hàm giá trị cho một chính sách là để tìm một chính sách tốt hơn chính sách hiện tại. Giả sử chúng ta có một chính sách π cố định tức là với mỗi trạng thái s, chính sách này luôn chọn thực hiện một hành động cố định, $\pi(s) = a$. Và cũng đã xác định được một hàm giá trị v_π cho một chính sách đó. Với một trạng thái s, câu hỏi đặt ra là chúng ta nên thay đổi một chính sách cố định khác chọn hành động $a' \neq \pi(s)$ không? Chúng ta biết giá trị của trạng thái s theo chính sách hiện tại π, $v_\pi(s)$, tốt như thế nào nhưng liệu chính sách mới π' có tốt hơn hay trở nên tệ đi? Theo [16], ta có thể cải thiện một chính sách bằng cách chọn hành động có giá trị cao nhất tại mỗi trạng thái s. Chính xách mới π' được xác định trong 2.19.

$$\pi'(a|s) \begin{cases} 1 & \text{nếu } a = \underset{a \in A}{\text{argmax}} q(s,a) \\ 0 & \text{ngược lại} \end{cases} \tag{2.19}$$

Ngoài ra, việc cải thiện chính sách bằng phương pháp tham lam này, ta đồng thời cũng cải thiện được hàm giá trị [16]. Việc đánh giá và cải thiện chính sách qua nhiều vòng lặp sẽ hội tụ về chính sách tối ưu cũng như hàm giá trị tối ưu. Hình 2.9 minh họa quá trình hội tụ của lặp chính sách.

2.2.5 Phương pháp lặp giá trị

Một nhược điểm của phương pháp lặp chính sách nằm ở giai đoạn đánh giá chính sách của phương pháp này. Bản thân giai đoạn này cũng là một quá trình lặp để tìm ra hàm giá trị của chính sách π đang theo. Ngoài ra, giai đoạn này cũng đã được chứng minh sẽ tìm được chính xác hàm giá trị v_π khi lặp vô hạn lần. Câu hỏi đặt ra là: Để tìm chính sách tối ưu, chúng ta có cần phải xác định chính xác hàm giá trị v_π của chính sách π hiện tại hay không? [15] đã đề xuất một phương pháp có thể tìm đươc chính sách tối ưu mà không cần xác định hàm giá trị v_π, được gọi là phương pháp lặp giá trị. Trong phương pháp lặp chính sách, tại mỗi vòng lặp một ước lượng V cho hàm giá trị v_π được cập nhật nhiều lần để đạt chính xác hàm v_π trước khi thực hiện cải thiện chính sách. Ngược



Hình 2.9: Ta có một chính π, đầu tiên ta thực hiện đánh giá chính sách π để có được hàm giá trị theo chính sách này. Khi có được hàm giá trị, ta thực hiện cải thiện chính sách bằng cách lựu chọn tham lam hành động có giá trị lớn nhất dựa trên hàm giá trị đang có. Sau khi có được chính sách mới, ta tiếp tục đánh giá chính sách để có được hàm giá trị theo chính sách đó. Và tiếp tục cải thiện khi đã có được hàm giá trị. Quá trình này được lặp nhiều lần cho đến khi đạt được chính sách tối ưu cũng như hàm giá trị tối ưu.

lại, phương pháp lặp giá trị thực hiện duy nhất một lần cập nhật ước lượng kết hợp với tối ưu bằng toán tử max ngay trong mỗi vòng lặp của nó. Chính vì vậy mà phương pháp lặp giá trị không tách biệt hai giai đoạn rõ ràng như phương pháp lặp chính sách và mục tiêu của phương pháp này là tìm được hàm giá trị tối ưu v_*. Phương trình 2.20 mô tả cách thức cập nhật hàm giá trị cũng như tối ưu trong một lần lặp. Ý nghĩa của phương trình này là dùng giá trị hành động lớn nhất cập nhật giá trị cho một trạng thái s.

$$\begin{aligned} v_{k+1}(s) &= \max_a \mathbb{E}\left[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s, A_t = a\right] \\ &= \max_a \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} \left[R_{t+1} + \gamma v_k(s')\right] \end{aligned} \tag{2.20}$$

với $s \in \mathcal{S}$ và v_0 được khởi tạo ngẫu nhiên. Qua nhiều cập nhật chuỗi $\{v_k\}$ sẽ hội tụ về hàm giá trị tối ưu v_*.

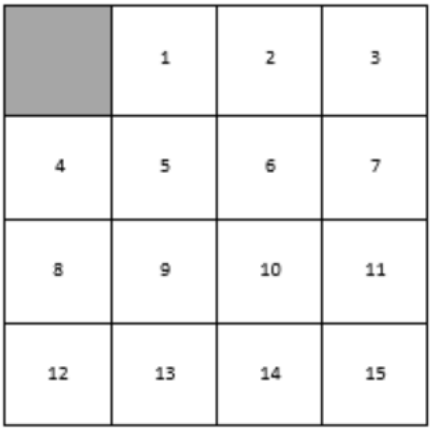
Cách thức cập nhật của phương pháp lặp giá trị dựa trên phương trình Bellman của hàm giá trị trạng thái tối ưu 2.16. Mô hình cập nhật của phương pháp lặp giá trị tương tự mô hình cập nhật của phương pháp lặp chính sách 2.8 ngoại trừ nó yêu cầu giá trị cực đại (maximum) được chọn qua các hành động. Thêm vào đó quá trình cập nhật của lặp chính sách để tính hàm giá trị trạng thái v_π của chính sách π hiện tại, trong khi đó quá trình cập nhật của lặp giá trị để tính hàm giá trị trạng thái tối ưu v_*. Các bước thực hiện trong phương pháp lặp giá trị được mô tả trong thuật toán 2.2.

Ví dụ: Bài toán di chuyển trên lưới 4×4 Hệ thống có thể thực hiện 4 hành động: lên, xuống, qua trái, qua phải. Vị trí đích là góc trái trên của lưới. Hệ thống sẽ dừng nếu di chuyển ra ngoài lưới. Mỗi hành động của hệ thống đều nhận được kiểm thưởng là -1. Và nhận được điểm thưởng +7 khi hệ thống về đích. Mục tiêu của hệ thống là tìm được đường về vị trí đích. Giả sử có nhiễu trong quá trình hệ thống di chuyển: 70% hệ thống sẽ di chuyển theo hướng trùng với hành động đã thực hiện, 30% hệ thống sẽ di chuyển khác hướng với hành động thực hiện. Ví dụ nếu hệ thống thực hiện hành động đi lên thì 70% hệ thống sẽ di chuyển đi lên, 10% hệ thống di chuyển sang trái, 10% hệ thống di chuyển sang phải, và 10% hệ thống di chuyển xuống dưới. Hệ số $\gamma = 1$. Hình 2.10 minh họa bài toán này.

Thuật toán 2.2 Phương pháp lặp giá trị trên hàm giá trị trạng thái

Đầu vào:
Đầu ra: Hàm giá trị V xấp xỉ hàm giá trị tối ưu v_*
Thao tác:

1: Khởi tạo ngẫu nhiên $V(s)$ cho tất cả trạng thái s không phải trạng thái kết thúc. Nếu s là trạng thái kết thúc, $V(s) = 0$
2: **repeat**
3: $\Delta \leftarrow 0$ %% Tính độ khác biệt giữa hàm giá trị cũ và giá trị mới. Độ lớn của Δ được xác định là độ khác biệt lớn nhất giữa giá trị cũ và giá trị mới của một trạng thái trong tất cả các trạng thái.
4: **for** $s \in \mathcal{S}$ **do** %% Với mỗi trạng thái
5: $v \leftarrow V(s)$ %% Lưu giá trị hiện tại của trạng thái s
6: $V(s) \leftarrow \max_a \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_s + \gamma V(s')\right]$ %% Duyệt và chọn ra giá trị hành động lớn nhất làm giá trị mới của trạng thái s
7: $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
8: **end for**
9: **until** $\Delta < \theta$ (Một lượng đủ nhỏ)



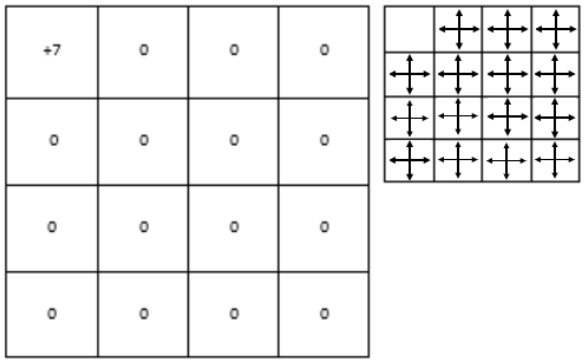
Hình 2.10: Bài toán di chuyển trên lưới

Qua ví dụ 2.11 minh họa quá trình tìm hàm giá trị trạng thái tối ưu bằng quá trình lặp giá trị, ta có thể thấy hàm giá trị trạng thái dần hội tụ về hàm giá trị q_* thông qua những chính sách tương ứng của mỗi lần cập nhật.

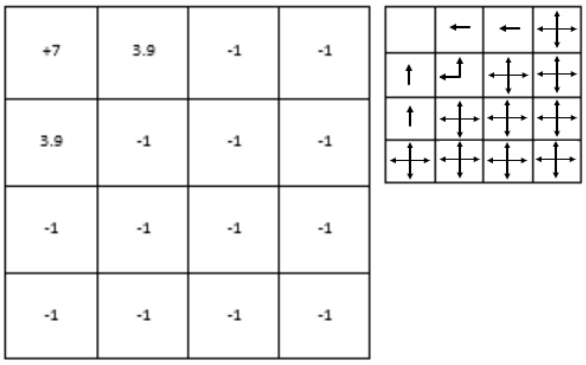
Trong nhiều trường hợp việc sử dụng hàm giá trị hành động trở nên khả thi hơn hàm giá trị trạng thái. Thực vậy, với việc có được mô hình MDP của môi trường , hàm giá trị trạng thái là đủ để xác định chính sách tối ưu nhưng việc xác định này phải qua một bước trung gian thông qua ma trận chuyển trạng thái. Nói cách khác để xác định chính sách tối ưu trên hàm giá trị trạng thái, hệ thống phải xác định những trạng thái tiếp theo có thể đến thông qua ma trận chuyển trạng thái, sau đó chọn hành động mà giúp cho hệ thống có thể đến được trạng thái tiếp theo có giá trị lớn nhất. Tuy nhiên trong một số bài toán hệ thống không biết mô hình MDP của môi trường thì hàm giá trị trạng thái là không đủ để xác định chính sách tối ưu. Ngược lại, có được hàm giá trị hành động là đủ để xác định chính sách tối ưu trong cả hai trường hợp biết và không biết MDP. Mặc dù khi biết được mô hình MDP của môi trường, việc xác định chính sách tối ưu bằng hàm giá trị hành động được cho là tốn kém trong việc lưu trữ nhưng những phương pháp tìm chính sách tối ưu vẫn thường sử dụng loại hàm giá trị này do nó có thể sử dụng trong cả hai trường hợp. Trong phương pháp lặp giá trị, ta cũng có thể xác định hàm giá trị hành động tối ưu qua nhiều lần lặp. Phương trình 2.21 minh họa cách thức cập nhật ước lượng của hàm giá trị q_*

$$\begin{aligned} q_{k+1}(s,a) &= \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q_k(S_{t+1}, a') \mid S_t = s, A_t = a\right] \\ &= \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} \left[R_{t+1} + \gamma \max_{a'} q_k(s', a')\right] \end{aligned} \tag{2.21}$$

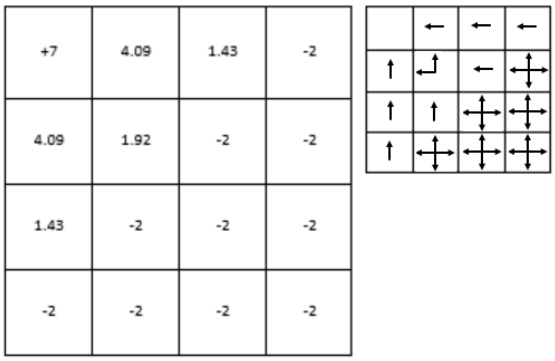
với $s \in \mathcal{S}, a \in \mathcal{A}(s)$. Thuật toán 2.3 minh họa các bước thực hiện lặp giá trị cho hàm hành động.



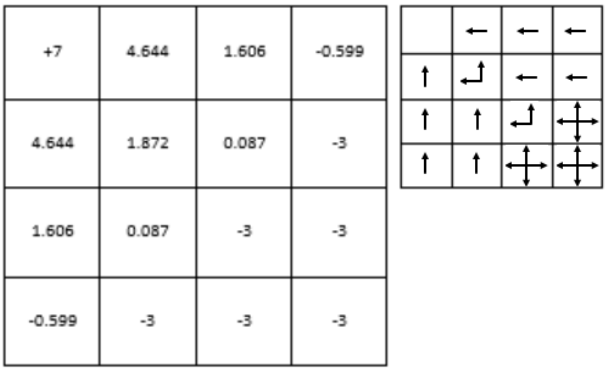
(a) Khởi tạo hàm giá trị V



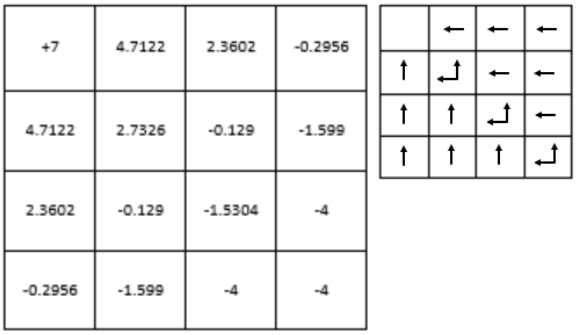
(b) Cập nhật hàm giá trị lần 1, và chính sách tham lam trên hàm giá trị hiện tại



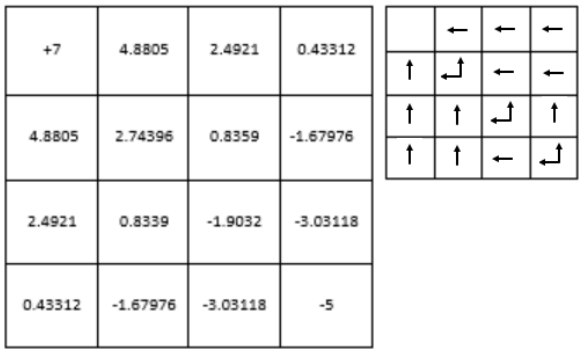
(c) Cập nhật hàm giá trị lần 2, và chính sách tham lam trên hàm giá trị hiện tại



(d) Cập nhật hàm giá trị lần 3, và chính sách tham lam trên hàm giá trị hiện tại



(e) Cập nhật hàm giá trị lần 4, và chính sách tham lam trên hàm giá trị hiện tại



(f) Cập nhật hàm giá trị lần 5, và chính sách tham lam trên hàm giá trị hiện tại

Hình 2.11: Quy trình cập nhật hàm giá trị trạng thái bằng phương pháp lặp giá trị

Thuật toán 2.3 Phương pháp lặp giá trị trên hàm giá trị hành động

Đầu vào:
Đầu ra: Hàm giá trị Q xấp xỉ hàm giá trị tối ưu q_*
Thao tác:

Khởi tạo ngẫu nhiên $Q(s, a)$ cho tất cả các cặp trạng thái s và hành động a với s không phải trạng thái kết thúc. Nếu s là trạng thái kết thúc, $Q(s, a) = 0$
repeat
    $\Delta \leftarrow 0$ %% Tính độ khác biệt giữa hàm giá trị cũ và giá trị mới. Độ lớn của Δ được xác định là độ khác biệt lớn nhất giữa giá trị cũ và giá trị mới của một trạng thái trong tất cả các trạng thái.
    for $s \in \mathcal{S}$ **do** %% Với mỗi trạng thái
        $q \leftarrow Q(s, a)$ %% Lưu giá trị hiện tại của cặp trạng thái, hành động (s, a)
        $Q(s, a) \leftarrow \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_s + \gamma \max_{a'} Q(s', a')\right]$ %% Thực hiện cập nhật
        $\Delta \leftarrow \max(\Delta, |q - Q(s, a)|)$
    end for
until $\Delta < \theta$ (Một lượng đủ nhỏ)

2.3 Áp dụng phương pháp lặp giá trị khi không có đầy đủ thông tin của môi trường

2.3.1 Bài toán học tăng cường khi không có đầy đủ thông tin của môi trường

Trong thực tế, việc có đầy đủ thông tin về cách hoạt động của môi trường dưới dạng các ma trận xác suất chuyển trạng thái $\mathcal{P}$ hay ma trận điểm thưởng $\mathcal{R}$ là không thể. Ví dụ như để xây dựng một robot tìm đường đi tự động, có rất nhiều yếu tố ảnh hưởng đến sự di chuyển của robot như gió, va chạm, ... Ta không thể tổng hợp hết tất cả các yếu tố này để xây dựng các ma trận xác suất như trên được. Tuy nhiên, điều mà ta có thể làm đó là tự học những thông tin cần thiết thông qua việc tương tác với môi trường. Ví dụ trong bài toán chơi cờ vua ta mong muốn xây dựng hệ thống có xác suất thắng cao nhất (điểm thưởng là 1 sau khi thắng và -1 sau khi thua mỗi trận). Khi đó, hệ thống có thể không biết hành động "khai cuộc bằng quân mã" sẽ cho bao nhiêu điểm thưởng (tức không biết xác suất thắng) nhưng khi chơi nhiều lần và thử "khai cuộc bằng quân mã", hệ thống có thể ước lượng được giá trị điểm thưởng này. Đây cũng chính là ý tưởng chính của các thuật toán học tăng cường trong những bài toán thực tế: tương tác với môi trường để **lấy mẫu (sampling)** và ước lượng hàm giá trị dựa vào các mẫu này. Theo đó, các mẫu dữ liệu lúc này sẽ là các bộ ba (S_t, A_t, R_{t+1}) tương ứng với trạng thái thời điểm t, hành động thời điểm t và điểm thưởng nhận được ngay sau đó.

Cách lấy mẫu dữ liệu trên dẫn đến một vấn đề: ta nên phân bố việc "thử và sai" này như thế nào? Như trong bài toán cờ vua, ta có rất nhiều cách khai cuộc nên nếu ta chỉ thử "khai cuộc bằng quân mã" thì ta sẽ không thử được những chiến thuật chơi khác. Ngoài ra, nếu như ta thử "khai cuộc bằng quân mã" và thấy đây là một hành động tốt (cho xác suất thắng cao), liệu ta có nên thử những cách khai cuộc khác không hay là tập trung tối ưu ở những nước cờ sau? Đây chính là một ví dụ của vấn đề quan trọng trong học tăng cường: **vấn đề khám phá và khai thác (exploration vs exploitation)**. Do ta chỉ có thể

tương tác với môi trường một số lần hữu hạn, ta cần phải cân bằng giữa việc khám phá không gian trạng thái với việc khai thác những kiến thức đã biết để xây dựng chính sách tốt hơn. Nếu như ta khám phá quá nhiều mà không khai thác, ta chỉ khám phá được những trạng thái "không tốt" do chính sách của ta không tốt nên khó gặp những trạng thái "tốt". Điều này dẫn đến việc khám phá sẽ không hiệu quả. Trong khi đó, nếu ta khai thác quá nhiều mà không khám phá, ta chỉ tận dụng những kiến thức đã biết (tức hành động theo chính sách hiện tại) nên ít khi nào gặp được những trạng thái mới (có thể là trạng thái "tốt hơn").

Một trong những cách đơn giản nhất để giải quyết vấn đề này đó là ta thực hiện khai thác phần lớn thời gian và thỉnh thoảng chèn vào đó một số hành động ngẫu nhiên để khám phá. Kỹ thuật này có tên gọi là "ϵ-greedy". Theo đó, mỗi khi cần lựa chọn hành động, với xác suất rất nhỏ (nhưng vẫn lớn hơn không) ϵ, ta thực hiện việc khám phá bằng cách chọn hành động ngẫu nhiên. Phần còn lại (tức là với xác suất $1 - \epsilon$), ta thực hiện việc khai thác kiến thức đã biết bằng cách chọn hành động mà có giá trị hành động cao nhất (tức chọn $a^{'} = \underset{a \in A}{\text{argmax}} Q(s, a)$). Thông thường, ϵ được chọn là một giá trị nhỏ như 0.1 hay 0.01.

2.3.2 Thuật toán "Q-learning"

Thuật toán "Q-learning" là một trong những thuật toán quan trọng nhất và được ứng dụng rộng rãi nhất trong học tăng cường [16]. Thuật toán này nói một cách đơn giản đó là áp dụng phương pháp lặp giá trị cho hàm giá trị hành động khi không có đầy đủ thông tin của môi trường. Chữ *"Q"* trong tên thuật toán cũng nói lên rằng thuật toán này sẽ học ra hàm giá trị hành động. Cụ thể hơn, do phương pháp lặp giá trị sẽ tìm được hàm giá trị **tối ưu** nên thuật toán "Q-learning" sẽ tìm thẳng ra hàm giá trị hành động tối ưu q_*.

Ý tưởng của thuật toán này rất đơn giản. Ta có công thức cập nhật của phương pháp lặp giá trị cho hàm giá trị hành động là:

$$q_{k+1}(s, a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \max_{a' \in \mathcal{A}} q_k(s', a') \tag{2.22}$$

Với công thức này, ta dễ dàng tương tác với môi trường để lấy mẫu cho biểu thức bên phải dấu = của (2.22). Phân tích cụ thể hơn, biểu thức này bao gồm:

- $\mathcal{R}_s^a$: đây là kỳ vọng điểm thưởng nhận được ngay lập tức khi thực hiện hành động a tại trạng thái s. Do bây giờ ta không biết thông tin này, ta cần phải tương tác với môi trường để lấy mẫu. Giả sử việc tương tác môi trường ở thời điểm t gặp trạng thái S_t và ta thực hiện hành động A_t. Khi đó, **điểm thưởng** R_{t+1} nhận được ngay tiếp theo chính là một ước lượng không chệch của $\mathcal{R}_s^a$. Lý do là theo định nghĩa, kỳ vọng của R_{t+1} chính là $\mathcal{R}_s^a$.

- $\sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \max_{a' \in \mathcal{A}} q_k(s', a')$: biểu thức này mang ý nghĩa là *trung bình (theo trạng thái tiếp theo s') của giá trị hành động tốt nhất tại trạng thái tiếp theo*. Khi tương tác với môi trường, ta không thể lấy trung bình theo trạng thái tiếp theo được vì khi thực hiện một hành động A_t thì ta sẽ sang trạng thái S_{t+1} (ta không thể quay lại trạng thái S_t để thử lại). Tuy nhiên, khi thực hiện hành động đó, trạng thái S_{t+1} mà ta đến trong một lần thực hiện hành động A_t ở trạng thái S_t chính là một mẫu của các trạng thái có thể có tiếp theo. Như vậy, ta hoàn toàn có thể sử dụng $\max_{a' \in \mathcal{A}} q_k(S_{t+1}, a')$ làm một ước lượng của biểu thức trên.

Ghép hai thành phần này lại, ta có công thức cập nhật hàm giá trị hành động của thuật toán "Q-learning" khi tương tác với môi trường:

$$q_{k+1}(S_t, A_t) = R_{t+1} + \gamma \max_{a' \in \mathcal{A}} q_k(S_{t+1}, a') \tag{2.23}$$

Hình 2.12 mô tả cách hoạt động của thuật toán "Q-learning".

Tuy nhiên, công thức trên gặp một vấn đề: nếu như ta gặp bộ trạng thái - hành động (S_t, A_t) nhiều lần thì ta sẽ cập nhật đè lên giá trị $q(S_t, A_t)$ cũ. Điều này làm cho những lần cập nhật trước là vô nghĩa. Ngoài ra, cách cập nhật này sẽ không đúng với "tinh thần" của việc ước lượng bằng cách lấy mẫu (đó là lấy trung bình của nhiều mẫu dữ liệu lại). Vì vậy, ta chỉ việc thay đổi nhỏ thuật toán "Q-learning" bằng cách thay vì cập nhật $q(S_t, A_t)$ bằng giá trị của lần gặp mới nhất, ta sẽ lấy trung bình của tất cả các lần gặp. Cách cập nhật này tuy

đúng về mặt toán học nhưng về mặt thực tiễn vẫn gặp một vấn đề nhỏ: việc lấy trung bình đồng nghĩa với việc coi tất cả các lần gặp đều có ý nghĩa ngang nhau. Một cách trực quan, ta có thể thấy những lần gặp càng mới thì càng có ý nghĩa. Ví dụ như trong bài toán mà môi trường có thể thay đổi theo thời gian, thông tin của những lần cập nhật cũ nhiều khả năng không còn đúng; những lần cập nhật càng mới thì thông tin càng có nhiều khả năng là chính xác. Để giải quyết điều này, ta chỉ việc sử dụng một phần thông tin của những lần gặp trước cùng với một phần thông tin của lần gặp mới.

$$\begin{aligned} q_{k+1}(S_t, A_t) &= \alpha \left[R_{t+1} + \gamma \max_{a' \in \mathcal{A}} q_k(S_{t+1}, a') \right] + (1-\alpha) q_k(S_t, A_t) \\ &= q_k(S_t, A_t) - \alpha \left[R_{t+1} + \gamma \max_{a' \in \mathcal{A}} q_k(S_{t+1}, a') - q_k(S_t, A_t) \right] \end{aligned} \tag{2.24}$$

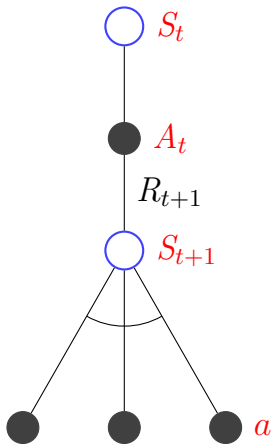
Hệ số α trong này điều khiển mức độ thông tin của lần gặp mới nhất mà ta mong muốn sử dụng. α có giá trị trong khoảng $(0, 1)$. α càng lớn thì ta càng sử dụng nhiều thông tin của lần gặp mới nhất (tức giá trị $R_{t+1} + \gamma \max_{a' \in \mathcal{A}} q_k(S_{t+1}, a')$) và giữ lại ít thông tin của những lần gặp cũ (tức giá trị $q_k(S_t, A_t)$). Khi α bằng 1 thì ta sẽ "quên" những lần cập nhật cũ và sử dụng hoàn toàn thông tin của lần gặp mới nhất. Trường hợp này tương đương với cách cập nhật của công thức (2.23).

Thuật toán "Q-learning" có một điểm khác biệt so với những thuật toán học tăng cường khác đó là việc hành động A_{t+1} tại trạng thái S_{t+1} không được dùng để cập nhật $q(S_t, A_t)$. Điều này có nghĩa là các hành động được thực hiện để tương tác với môi trường chỉ giúp cho "Q-learning" có được những bộ (trạng thái, hành động) cùng với giá trị điểm thưởng tương ứng để cập nhật hàm giá trị. Hành động A_{t+1} không xuất hiện ở biểu thức cập nhật cho thấy chính sách thực hiện hành động tương tác với môi trường không ảnh hưởng đến việc cập nhật hàm giá trị hành động (ngoại trừ việc cung cấp các bộ trạng thái - hành động). Như vậy, hàm giá trị mà thuật toán "Q-learning" học được ứng với chính sách nào? Hàm giá trị mà thuật toán "Q-learning" học được chính là hàm giá trị tối ưu và chính sách tương ứng là chính sách tối ưu. Một cách để hiểu tại sao "Q-learning" hội tụ đến hàm giá trị tối ưu, ta có thể so sánh với phương pháp

lặp giá trị. Phương pháp lặp giá trị hội tụ đến hàm giá trị tụ đến hàm giá trị tối ưu. Mà thuật toán "Q-learning" cũng chính là cách áp dụng phương pháp lặp giá trị cho hàm giá trị hành động khi không có đầy đủ thông tin của môi trường. Thuật toán "Q-learning" chỉ đơn giản là lấy mẫu thực tế thay cho các giá trị xác suất, kỳ vọng điểm thưởng của môi trường.

Thuật toán "Q-learning" học ra hàm giá trị của một chính sách khác với chính sách tương tác với môi trường. Các thuật toán có cùng tính chất này được gọi là thuật toán "off-policy". Ngược lại, nhưng thuật toán học ra hàm giá trị của chính sách đang tương tác với môi trường được gọi là "on-policy". Nhờ việc học "off-policy", "Q-learning" cố thể được dùng để học lại những mẫu dữ liệu cũ. Rõ ràng là ta hoàn toàn có thể dùng công thức cập nhật (2.24) của "Q-learning" cho những mẫu dữ liệu cũ $S_{t-1}, A_{t-1}, S_{t-2}, A_{t-2}, ...$ Các thuật toán "on-policy" không thể thực hiện việc cập nhật như vậy vì những mẫu dữ liệu cũ này được sinh ra bởi một chính sách khác (tức chính sách cũ nếu như ta cập nhật chính sách sau mỗi lần cập nhật hàm giá trị).

Vậy ta nên chọn chính sách nào để tương tác với môi trường trong thuật toán "Q-learning"? Rõ ràng là nếu chính sách này không gặp những trạng thái hoặc không thực hiện một số hành động thì "Q-learning" không thể cập nhật giá trị cho các trạng thái và hành động này được. Về mặt lý thuyết, thuật toán "Q-learning" luôn hội tụ đến hàm giá trị tối ưu nếu như mọi cặp *trạng thái - hành động* luôn được thăm [16]. Trong thực tế, một trong những chính sách thường được sử dụng đó là "ϵ-greedy". Lúc này, ta chỉ việc chọn hành động tham lam theo hàm giá trị hành động đang được cập nhật của "Q-learning" hoặc chọn hành động ngẫu nhiên với một xác suất nhỏ.



Hình 2.12: Đồ thị minh họa cách cập nhật hàm giá trị hành động bằng thuật toán "Q-learning". "Q-learning" xác định mục tiêu cập nhật cho giá trị của cặp trạng thái, hành động (S_t, A_t) bằng tổng của điểm thưởng R_{t+1} nhận được ngay lập tức và giá trị hành động lớn nhất tại trạng thái kế tiếp S_{t+1}.

Chương 3

Kết hợp học tăng cường với học sâu

Những thành công gần đây của học sâu (Deep learning) trong các bài toán như xử lý ngôn ngữ tự nhiên, nhận diện đối tượng trong ảnh... đặt ra vấn đề: liệu các kỹ thuật trong học sâu có thể áp dụng vào học tăng cường? Để trả lời câu hỏi đó, chương này trình bày về hướng tiếp cận kết hợp học tăng cường với học sâu để áp dụng vào bài toán tự động chơi game. Hướng tiếp cận này giúp cho các thuật toán học tăng cường được trình bày trong chương 2 có thể áp dụng vào những bài toán thực tế với số lượng trạng thái rất lớn. Chương này trình bày hai phần:

- *Các kiến thức cơ bản của học sâu và cách kết hợp học tăng cường với học sâu.*
- *Áp dụng hướng tiếp cận này vào bài toán tự động chơi game.*

3.1 Kết hợp học tăng cường với học sâu

3.1.1 Học sâu

Lý do cần áp dụng học sâu

Những thuật toán học tăng cường được trình bày trong chương trước đều tìm chính sách tối ưu dựa vào hàm giá trị. Việc tính **đúng** và **nhanh** hàm giá trị ảnh hưởng rất nhiều đến kết quả của bài toán. Các thuật toán học tăng cường cổ điển như "Monte Carlo" (MC) hay "Temporal-Difference" (TD) đều đã được chứng minh là luôn hội tụ trong những điều kiện nhất định [16]. Ngoài ra, khi áp dụng vào các bài toán kinh điển của học tăng cường thì các thuật toán này đều hội tụ khá nhanh.

Tuy nhiên, với những bài toán thực tế với số trạng thái rất lớn thì việc lưu véc-tơ hàm giá trị trạng thái v_π (hoặc ma trận hàm giá trị hành động q_π) là việc không thể. Ví dụ như frame hình của bài toán tự động chơi game có kích thước $210 \times 160 \times 3 = 100800$ điểm ảnh; mỗi điểm ảnh có giá trị trong khoảng $[0, 127]$ nên số trạng thái có thể có lên đến 128^{100800}. Vì vậy, việc lưu trữ hàm giá trị dưới dạng bảng là không khả thi về mặt bộ nhớ. Còn về mặt tốc độ tính toán thì các thuật toán học tăng cường trên đều tính hàm giá trị **rời rạc** cho từng trạng thái. Với số trạng thái quá lớn như trên thì ta không thể duyệt lần lượt từng trạng thái để tính được.

Những lý do trên dẫn đến việc sử dụng một phương pháp xấp xỉ hàm là bắt buộc cho các bài toán học tăng cường với số trạng thái lớn. Một trong những tiếp cận rất tự nhiên đó là sử dụng các mô hình học có giám sát như là một phương pháp xấp xỉ hàm giá trị. Đặc biệt, với những đột phá gần đây của học sâu trong lĩnh vực xử lý ảnh, video... [10] thì việc áp dụng các mô hình phổ biến của học sâu vào bài toán tự động chơi game là đầy hứa hẹn.

Giới thiệu học sâu

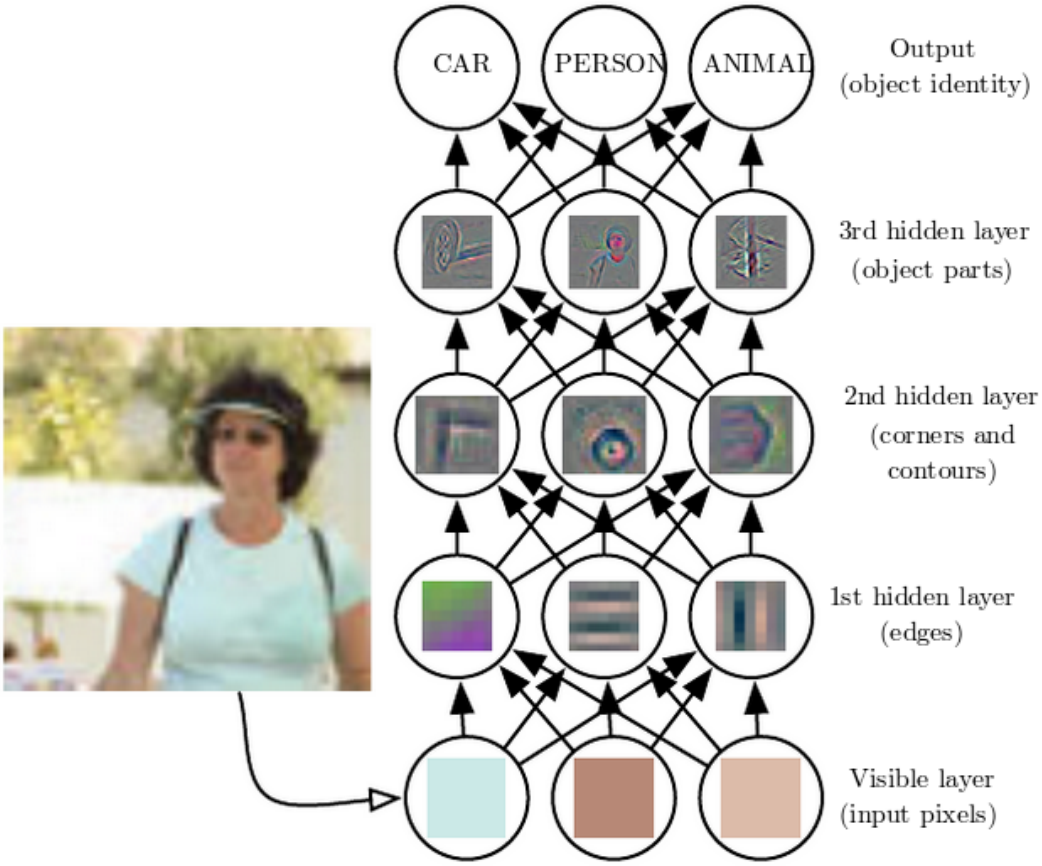
Các mô hình truyền thống trong lĩnh vực máy học như "Linear regression", "Bayesian learning"... thông thường đều hoạt động trên các biểu diễn dữ liệu

(thường được gọi là đặc trưng) được **rút trích một cách thủ công** (hand-designed features). Với dữ liệu thô thu thập được từ thực tế, các nhà khoa học xây dựng các phương pháp rút trích ra những "thông tin hữu ích" để cung cấp cho các mô hình máy học. Kết quả nhận được từ các mô hình này phụ thuộc rất lớn vào cách biểu diễn dữ liệu. Ví dụ như trong bài toán nhận diện người nói từ một đoạn âm thanh, các đặc trưng có thể bao gồm: độ lớn âm thanh, tần số trung bình của đoạn âm,... Nếu các đặc trưng này không đủ "tốt" (như có hai người nói đoạn âm thanh nhưng lại có chung độ lớn, tần số...) thì các mô hình học sẽ không thể phân biệt.

Để giải quyết vấn đề thiết kế đặc trưng, các mô hình máy học thuộc loại "Học biểu diễn" *(Representation learning)* ra đời. Các mô hình này có khả năng tự động học luôn các đặc trưng cần thiết cho quá trình phân tích dữ liệu. Nhờ vậy, các mô hình này có thể áp dụng được dễ dàng hơn vào các bài toán thực tế mà không cần con người phải can thiệp. Tuy nhiên, các đặc trưng cần thiết lại có thể rất phức tạp. Việc học ra các đặc trưng này có thể khó ngang với việc giải bài toán gốc. Ví dụ như trong bài toán nhận diện người nói trên, ta có thể sử dụng thông tin về giọng địa phương (accent) của người nói. Đặc trưng này rất trừu tượng, không dễ phân tích bằng máy tính mặc dù con người có thể nhận biết một cách khá dễ dàng. Vì vậy, nếu các đặc trưng cần thiết quá khó để học thì các mô hình máy học thuộc loại "Học biểu diễn" cũng không thể cho kết quả tốt.

Học sâu (Deep learning) được ra đời nhằm giải quyết các vấn đề trên. Học sâu là một nhánh của "Học biểu diễn" nên vẫn có thể tự động học ra các đặc trưng hữu ích. Học sâu được thiết kế để học ra các đặc trưng có quan hệ với nhau theo nhiều tầng (layer). Các đặc trưng ở tầng phía sau được xây dựng dựa vào các đặc trưng ở tầng phía trước. Các tầng đầu tiên bao gồm những đặc trưng đơn giản và các tầng tiếp theo ngày càng trừu tượng, ngày càng phức tạp hơn. Mỗi tầng chỉ cần học cách xây dựng đặc trưng từ những đặc trưng ở tầng phía trước (đã có sẵn độ trừu tượng nhất định) thay vì học từ dữ liệu thô ban đầu; điều này giúp cho học sâu có khả năng học được những đặc trưng có tính trừu tượng cao.

Một trong những mô hình học sâu phổ biến nhất và cũng cho kết quả rất



Hình 3.1: Hình mô phỏng cách hoạt động của mô hình học sâu cho bài toán nhận diện đối tượng trong ảnh. Dữ liệu đầu vào là hình ảnh RGB chứa đối tượng cần xác định. Tầng đầu tiên của mô hình là tầng input tiếp nhận thông tin này dưới dạng ma trấn số. Các tầng tiếp theo ngoại trừ tầng cuối cùng được gọi là tầng ẩn "hidden layer" vì đặc trưng học được tại đây con người không quan sát được. Các tầng ẩn học các đặc trưng ngày càng trừu tượng dựa vào đặc trưng ở tầng phía trước. Tầng ẩn đầu tiên học được các đặc trưng về cạnh bằng cách so sánh độ sáng giữa các điểm ảnh gần nhau. Tầng ẩn thứ hai học được các đặc trưng về đường cong bằng cách tổng hợp đặc trưng về cạnh ở tầng trước đó. Tầng ẩn thứ ba học được các đặc trưng về bộ phận của đồ vật như khuôn mặt, bánh xe... dựa vào các đặc trưng về đường cong ở tầng trước. Tầng cuối cùng được gọi là tầng output có nhiệm vụ tìm kiếm các bộ phận và trả về lớp đối tượng tương ứng. Bằng cách học đặc trưng ngày càng trừu tượng hơn, các mô hình học sâu có khả năng tự động học được các đặc trưng từ đơn giản đến phức tạp; tất cả đều nhằm hỗ trợ cho quá trình phân lớp dữ liệu (hình được chỉnh sửa từ [4])

tốt với dữ liệu ảnh [10] đó là mạng nơ-ron tích chập (Convolutional Neural Networks). Với bài toán tự động chơi game, dữ liệu hệ thống nhận được từ môi trường là ảnh RGB. Chính vì vậy, mạng nơ-ron tích chập là một mô hình rất phù hợp để kết hợp với các thuật toán học tăng cường.

3.1.2 Mạng nơ-ron tích chập

Mạng nơ-ron tích chập (Convolutional Neural Networks - CNN) là một mô hình học sâu được áp dụng rộng rãi trong các bài toán liên quan đến ảnh, video hoặc âm thanh... [10]. CNN được thiết kế để tận dụng thông tin về không gian (spatial structures) của các loại dữ liệu nêu trên. Ví dụ như trong bài toán nhận diện mặt người trong ảnh, thông tin về vị trí của mắt, mũi, miệng,... là rất quan trọng. Nếu ta coi các điểm ảnh đều có ý nghĩa tương tự nhau thì ta đã bỏ quên thông tin về vị trí của những đặc trưng này. Trong khi đó, thông tin về vị trí tương đối của mắt, mũi, miệng... rõ ràng là rất quan trọng trong việc nhận diện mặt người. Trong ví dụ trên, nếu ta áp dụng các mô hình không quan tâm đến loại dữ liệu (coi từng thuộc tính dữ liệu đầu vào là độc lập) thì ta sẽ không tận dụng được thông tin về không gian trong đó.

Để tận dụng thông tin về không gian trong dữ liệu ảnh, CNN được thiết kế để học các đặc trưng trên một vùng nhỏ của ảnh. Các đặc trưng này được lưu trữ dưới dạng những **bộ lọc** (filter) thường có **kích thước nh**ỏ hơn nhiều so với kích thước ảnh gốc. Các đặc trưng sau khi được học được áp dụng trên toàn bộ ảnh gốc để kiểm tra xem vị trí nào của ảnh xuất hiện đặc trưng này. CNN thực hiện phép kiểm tra này bằng cách "trượt" các bộ lọc này trên toàn bộ ảnh gốc. Phép "trượt" được thực hiện lần lượt từ trái qua phải và từ trên xuống dưới. Một cách tổng quát, phép "trượt" này có thể di chuyển không đồng đều theo hai chiều: ta có thể chỉ di chuyển qua phải một điểm ảnh nhưng lại di chuyển xuống dưới hai điểm ảnh. Tại mỗi vị trí, bộ lọc có nhiệm vụ kiểm tra thử đặc trưng được học có xuất hiện (hoặc mức độ rõ ràng của đặc trưng - đặc trưng xuất hiện nhiều hay ít) tại vị trí này không. Kết quả của phép kiểm tra này được lưu trữ lại dưới dạng một "bức ảnh" kết quả; giá trị mỗi "điểm ảnh" lúc này chính là kết quả của phép kiểm tra đặc trưng của bộ lọc. Do phép "trượt" được thực hiện theo thứ tự được nêu ở trên, các "điểm ảnh" kết quả vẫn mang thông tin

tương đối về vị trí: "điểm ảnh" bên trái ứng với kết quả của vùng nằm bên trái trong ảnh gốc và tương tự với điểm ảnh bên phải.

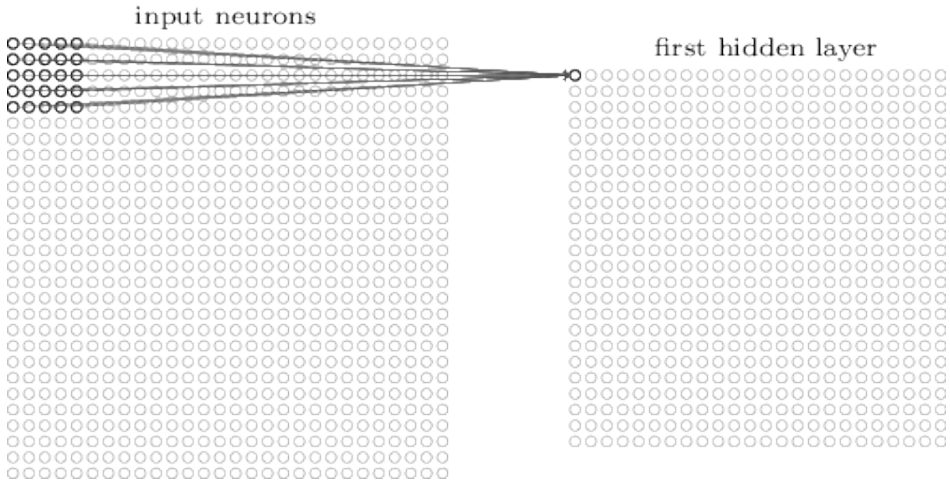
Phép "kiểm tra" đặc trưng của bộ lọc được thực hiện thông qua phép toán **tích chập** (convolution). Kết quả của phép tích chập được cộng với giá trị "bias" và đưa qua một hàm kích hoạt. Toàn bộ quá trình này có thể được biểu diễn thành công thức:

$$a_{i,j} = \sigma\left(b + \sum_{u=0}^{height}\sum_{v=0}^{width} w_{u,v}x_{i+u,j+v}\right) \tag{3.1}$$

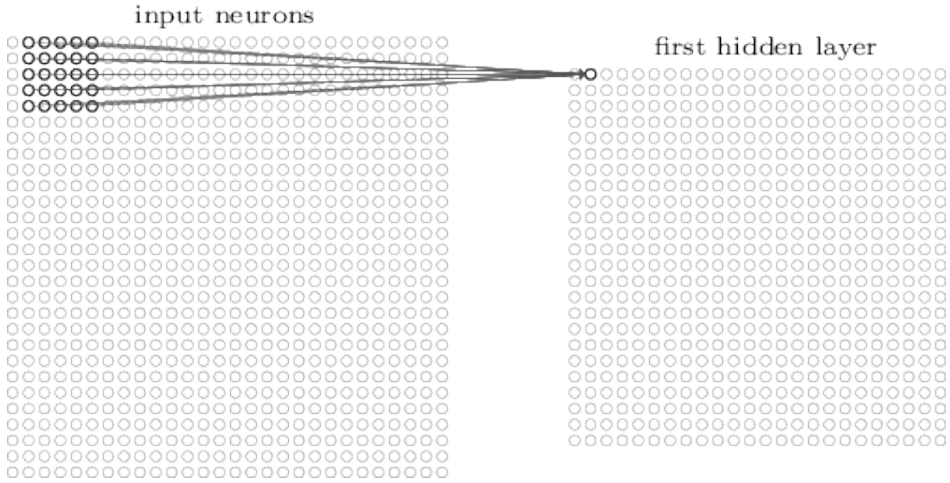
Trong đó:

- $a_{i,j}$ là kết quả của phép kiểm tra tại vùng có góc trái trên là i, j trong ảnh gốc.
- b là số thực gọi là "bias" của bộ lọc.
- $width, height$ lần lượt là chiều rộng và chiều cao của bộ lọc (5×5 trong hình 3.2).
- w là ma trận trọng số có kích thước $height \times width$ của bộ lọc. $w_{u,v}$ là thành phần dòng u cột v của ma trận.
- $x_{i+u,j+v}$ là giá trị điểm ảnh đầu vào tại vị trí dòng $i+u$ và cột $j+v$.
- σ là một hàm phi tuyến được gọi là hàm kích hoạt (activation function).

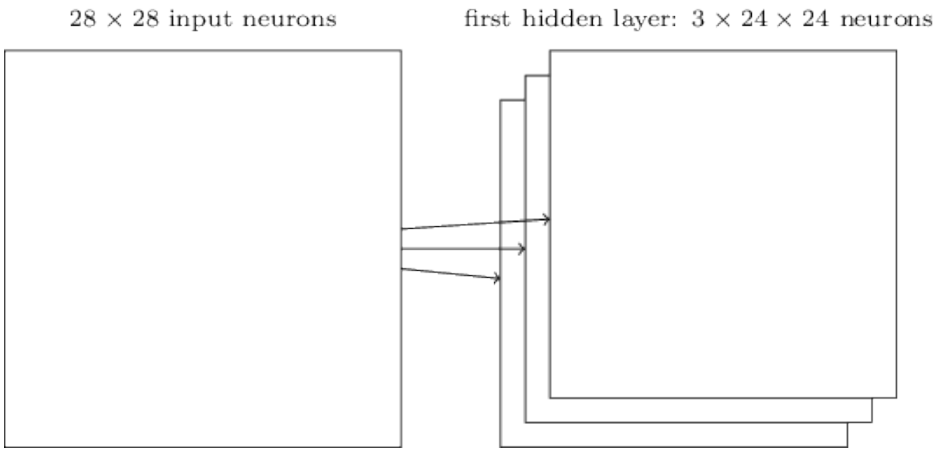
Phép toán tích chập này chỉ đơn giản là một tổng gồm các tích của trọng số và giá trị điểm ảnh. Hàm kích hoạt sau đó nhận kết quả của phép tích chập để thực hiện phép biến đổi phi tuyến tính; nhờ có hàm kích hoạt, bộ lọc có thể học được những đặc trưng phi tuyến tính. Một trong những hàm phi tuyến hay được áp dụng đó là hàm "Rectified linear": $\sigma(x) = \max(0, x)$. Công thức tích chập trên chứa hai thành phần cần được "học" đó là "bias" b và ma trận trọng số w. Hai thành phần này lưu trữ thông tin về đặc trưng mà bộ lọc học được. Một trong những đặc điểm quan trọng nhất là việc ma trận trọng số w và "bias" b được sử dụng chung cho việc "kiểm tra" đặc trưng tại mọi vị trí của ảnh gốc (hình



Hình 3.2: Hình mô tả phép "kiểm tra" đặc trưng của bộ lọc có kích thước 5×5 tại vị trí đầu tiên (góc trái trên) của ảnh đầu vào. Kết quả của phép "kiểm tra" này được lưu lại thành một "điểm ảnh" trong "bức ảnh kết quả". Lưu ý ảnh đầu vào trong ví dụ ở đây có kích thước 28×28. Do bộ lọc chỉ kiểm tra những vùng nằm hoàn toàn trong ảnh nên kích thước của "bức ảnh" đầu ra là 24×24.



Hình 3.3: Bộ lọc được trượt sang một điểm ảnh về phía bên phải để kiểm tra vùng cục bộ 5×5 bên cạnh. Kết quả của phép "kiểm tra" này được lưu lại thành một "điểm ảnh" bên phải của kết quả của vùng cục bộ trước đó. Thực hiện lần lượt quá trình "trượt" này đến hết ảnh ta sẽ có "bức ảnh kết quả" cuối cùng. "Bức ảnh kết quả" này mang thông tin về một đặc trưng cụ thể tại những vùng cục bộ liên tiếp nhau trên ảnh gốc.



Hình 3.4: Hình mô tả tầng tích chập với ba bộ lọc học đặc trưng từ ảnh đầu vào. Mỗi bộ lọc lúc này học một bộ trọng số w và giá trị "bias" b khác nhau. Do các bộ lọc có cùng kích thước (5×5) nên "bức hình" kết quả cũng có cùng kích thước. Ta coi mỗi "bức hình" kết quả như một kênh (channel) khác nhau của bức hình và ghép chúng lại thành một bức hình lớn hơn gồm ba kênh. Các kênh này cũng giống như ba kênh màu khác nhau của ảnh RGB.

3.3). Nhờ việc sử dụng chung ma trận trọng số và "bias" này, số lượng trọng số phải học của CNN được giảm đi rất nhiều. Cụ thể hơn, với ảnh gốc kích thước $28 \times 28 = 784$ như trong hình 3.2, nếu áp dụng mạng nơ-ron truyền thẳng thông thường để học đặc trưng từ toàn bộ ảnh, số lượng trọng số sẽ lên đến 784 cho một đặc trưng. Trong khi đó, CNN chỉ gồm khoảng $5 \times 5 = 25$ trọng số cần học.

Một bộ lọc của CNN chỉ học được một đặc trưng cụ thể. Tuy nhiên trong bài toán thực tế, ta cần nhiều đặc trưng khác nhau trên ảnh. Ví dụ như để nhận diện khuôn mặt trên ảnh, ta cần đặc trưng về mắt, miệng... Để học được nhiều đặc trưng khác nhau với CNN, ta chỉ việc thiết kế thêm nhiều bộ lọc học đặc trưng song song với nhau từ ảnh gốc. Với mỗi bộ lọc, ta cần học một ma trận trọng số và giá trị "bias" khác nhau. Ta gọi tập hợp các bộ lọc này là một tầng tích chập (convolution layer). Hình 3.4 mô tả tầng tích chập với ba bộ lọc.

CNN là một mô hình học sâu. Vì vậy để học được những đặc trưng có tính trừu tượng cao, CNN áp dụng phương pháp tổng hợp đặc trưng trừu tượng từ những đặc trưng đơn giản hơn. Để tổng hợp đặc trưng, ta chỉ việc coi "bức hình kết quả" như là bức ảnh đầu vào và thêm tầng tích chập mới học đặc trưng từ bức hình này. Do kết quả của tầng tích chập trước mang thông tin về đặc trưng được học bởi các bộ lọc, tầng tích chập tiếp theo thực hiện việc học từ các đặc

trưng đã học ở tầng trước. Nhờ vậy, việc thêm vào các tầng tích chập tiếp theo sẽ giúp mô hình học được những đặc trưng ngày càng trừu tượng hơn.

Phần tiếp theo sẽ trình bày cách sử dụng CNN như là một công cụ xấp xỉ hàm hỗ trợ cho các thuật toán học tăng cường. Do có khả năng học đặc trưng tự động, ta có thể thiết kế một công cụ xấp xỉ hàm "đa năng": vừa học đặc trưng từ hình ảnh, vừa xấp xỉ hàm mong muốn từ những hình ảnh đó.

3.1.3 Sử dụng mạng nơ-ron để xấp xỉ hàm

Việc sử dụng mạng nơ-ron để xấp xỉ hàm giá trị mang lại ba lợi ích quan trọng:

- Mạng nơ-ron có khả năng học được những đặc trưng phức tạp từ dữ liệu thô.
- Mạng nơ-ron có thể xấp xỉ hàm giá trị phức tạp.
- Mạng nơ-ron có thể tổng quát hoá từ trạng thái đã gặp sang trạng thái chưa gặp.

Mạng nơ-ron là một mô hình học sâu nên có thể học được những đặc trưng trừu tượng từ dữ liệu thô như đã nói ở phần trên. Cụ thể hơn trong bài toán tự động chơi game, ta có thể đưa dữ liệu đầu vào là các frame hình RGB thẳng vào input của mạng nơ-ron để tính ra giá trị tương ứng. Nhờ vậy, ta không cần phải thiết kế các đặc trưng bằng tay để biểu diễn lại từng trạng thái của game. Ngoài ra, do quá trình học đặc trưng là hoàn toàn tự động, thuật toán học tăng cường lúc này **có thể áp dụng cho bất kỳ game** nào mà không cần phải thay đổi cách rút trích đặc trưng. Với một mô hình cố định lúc này, ta có thể học chơi được nhiều game. Đây cũng chính là mục đích của bài toán tự động chơi game: xây dựng mô hình có khả năng tự động học chơi *tốt* nhiều game chứ không chỉ chơi *"hoàn hảo"* một game.

Với bài toán tự động chơi game, giá trị của một trạng thái bất kỳ rất khó xác định do một màn game thường rất dài. Vì thế, **hàm giá trị của bài toán này là một hàm phi tuyến phức tạp** và không liên tục. Công cụ xấp xỉ hàm giá trị phải có khả năng xấp xỉ những hàm phức tạp như vậy thì thuật toán học tăng cường mới đạt được hiệu quả. Với cách nhìn nhận mạng nơ-ron như là một

công cụ xấp xỉ hàm, ta có thể thấy mạng nơ-ron rất linh hoạt với khả năng xấp xỉ hàm đích bất kỳ.

Một tính chất quan trọng khác của mạng nơ-ron chính là khả năng tổng quát hoá (generalization) từ trạng thái đã gặp sang trạng thái chưa gặp [13]. Nhờ đặc điểm này mà quá trình học được tăng tốc đáng kể. Thay vì phải duyệt qua từng trạng thái (thậm chí phải duyệt nhiều lần) để tính hàm giá trị tại đó, mạng nơ-ron có khả năng **"dự đoán" giá trị** của một trạng thái chưa từng thấy dựa vào những trạng thái đã thấy. Như trong bài toán tự động chơi game thì các frame hình liên tiếp thường rất giống nhau và các trạng thái này thường cũng có giá trị tương đương nhau. Vì vậy, khi học xong cách chơi một game nào đó, hệ thống vẫn hoạt động tốt trong quá trình kiểm thử khi gặp những tình huống game chưa từng thấy trong lúc huấn luyện.

Các thuật toán học tăng cường ở chương 2 đều lưu hàm giá trị dưới dạng bảng (lookup table). Để sử dụng mạng nơ-ron như một công cụ xấp xỉ hàm, lúc này ta coi hàm giá trị là một hàm có tham số (parameterized function) và đi tìm các tham số này:

$$\hat{v}(s;\theta) \approx v_\pi(s) \tag{3.2}$$

$$\hat{q}(s,a;\theta) \approx q_\pi(s,a) \tag{3.3}$$

θ là bộ trọng số của mạng nơ-ron mà ta cần học. Công thức (3.2) được dùng khi ta muốn xấp xỉ hàm giá trị trạng thái và công thức (3.3) là để xấp xỉ hàm giá trị hành động. Để học được bộ trọng số xấp xỉ tốt hàm đích ($v_\pi(s)$ hoặc $q_\pi(s,a)$), ta cung cấp các mẫu dữ liệu (data sample). Mỗi mẫu bao gồm dữ liệu đầu vào của mạng nơ-ron (tức trạng thái s hoặc bộ trạng thái, hành động s,a) cùng với giá trị đích mong muốn (tức $v_\pi(s)$ hoặc $q_\pi(s,a)$). Khi ta cung cấp đủ nhiều mẫu dữ liệu cho mạng nơ-ron, các bộ tham số sẽ được thay đổi để mạng xấp xỉ được hàm đích mong muốn. Số lượng mẫu dữ liệu càng lớn thì mạng nơ-ron càng "thấy" được nhiều giá trị tại nhiều vị trí khác nhau của hàm đích hơn, khi đó mạng nơ-ron càng có khả năng xấp xỉ hàm đích tốt hơn. Để xét xem mạng nơ-ron có xấp xỉ tốt hàm đích hay chưa, ta có thể tính độ "khác biệt" của giá

trị đích với giá trị xấp xỉ trên cả không gian đầu vào:

$$J(\theta) = \mathbb{E}_{s\sim\pi}[(v_\pi(s) - \hat{v}(s;\theta))^2] \tag{3.4}$$

$$J(\theta) = \mathbb{E}_{s,a\sim\pi}[(q_\pi(s,a) - \hat{q}(s,a;\theta))^2] \tag{3.5}$$

Kỳ vọng $\mathbb{E}_{s\sim\pi}$ ý chỉ kỳ vọng với biến ngẫu nhiên là trạng thái s được lấy từ phân bố do chính sách π tạo nên. Ví dụ như ta cần xấp xỉ hàm giá trị của một chính sách chỉ luôn "đi qua trái" thì s sẽ là những trạng thái mà khi đi theo chính sách này, ta có thể gặp được s. Trong khi đó nếu chính sách đang xét là "đứng yên" (không thay đổi trạng thái) thì s chỉ có thể có một trạng thái duy nhất đó là trạng thái bắt đầu. Giá trị nằm trong kỳ vọng là độ lỗi bình phương giữa giá trị xấp xỉ và giá trị đích. Với hàm lỗi bình phương, có thể thấy khi $J(\theta)$ càng nhỏ thì bộ trọng số θ giúp cho mạng nơ-ron xấp xỉ hàm đích càng tốt. Lý do chính ta chọn độ lỗi bình phương (ví dụ thay vì độ lỗi theo trị tuyệt đối) là vì việc tính toán (như đạo hàm) trên hàm bình phương dễ dàng hơn.

Lưu ý là ở đây, hàm lỗi $J(\theta)$ là hàm theo θ; tức là lúc này, ta cố định bộ trọng số θ để tìm ra "sai số" mà bộ trọng số này gây ra. Nếu ta có hai bộ trọng số θ_1 và θ_2, ta có thể so sánh khả năng xấp xỉ hàm đích của chúng bằng cách so sánh giá trị $J(\theta_1)$ và $J(\theta_2)$ tương ứng.

Tuy nhiên ta không thể tính độ lỗi trên mọi điểm dữ liệu đầu vào theo công thức (3.4) và (3.5) được vì số lượng trạng thái là rất lớn. Vậy ta có thể xấp xỉ giá trị $J(\theta)$ bằng cách chỉ xét độ lỗi trên một tập huấn luyện nhỏ (hay còn gọi là "batch") các trạng thái mà ta biết được giá trị đích. Khi đó, kỳ vọng $\mathbb{E}_{s\sim\pi}$ được thay thế bằng giá trị trung bình độ lỗi trên từng mẫu dữ liệu của "batch":

$$J(\theta) = \frac{1}{N}\sum_{i=1}^{N}(v_\pi(S_i) - \hat{v}(S_i;\theta))^2 \tag{3.6}$$

$$J(\theta) = \frac{1}{N}\sum_{i=1}^{N}(q_\pi(S_i,A_i) - \hat{q}(S_i,A_i;\theta))^2 \tag{3.7}$$

Trong đó:

- N là số mẫu dữ liệu trong "batch".

- S_i, A_i tương ứng là trạng thái và hành động của mẫu dữ liệu thứ i của "batch".

Với một tập huấn luyện, ta mong muốn tìm được bộ trọng số giúp cho mạng nơ-ron xấp xỉ tốt hàm đích trên các mẫu dữ liệu thuộc tập huấn luyện này. Một thuật toán đơn giản và hay được sử dụng để tìm bộ trọng số này đó là "Batch Gradient Descent" (BGD). Để cực tiểu hoá hàm $J(\theta)$, thuật toán BGD thực hiện lặp lại nhiều "bước đi" nhỏ để thay đổi bộ trọng số θ dần dần; mỗi bước đi sẽ giúp cho hàm $J(\theta)$ giảm đi một ít. Để chọn "hướng đi" (tức cách cập nhật θ) thì BGD sẽ "nhìn" xung quanh vị trí hiện tại và đi theo hướng nào giúp giảm $J(\theta)$ nhiều nhất có thể. Hướng đi này chính là ngược hướng véc-tơ đạo hàm riêng (tức "gradient") của hàm $J(\theta)$ tại θ. Như vậy, thuật toán BGD thực hiện lặp lại nhiều lần việc cập nhật bộ trọng số θ theo công thức:

$$\theta_{t+1} = \theta_t - \alpha \nabla_\theta J(\theta) \tag{3.8}$$

Trong đó:

- θ_t là bộ trọng số tại bước thứ t.
- α là hệ số học (learning rate); giá trị này dùng để điều khiển độ lớn của "bước đi".
- $\nabla_\theta J(\theta)$ là véc-tơ đạo hàm riêng của hàm $J(\theta)$ tại vị trí θ.

Do ta chỉ "nhìn" cục bộ tại vị trí hiện tại nên nếu ta đi một bước quá dài (hệ số học α lớn) thì giá trị hàm lỗi J tại điểm đến sẽ không chắc là luôn giảm; còn nếu bước đi quá ngắn thì mỗi bước chỉ giảm $J(\theta)$ một ít, khi đó ta sẽ tốn rất nhiều thời gian để $J(\theta)$ đạt giá trị cực tiểu.

Một thuật toán cải tiến của BGD hay được sử dụng đó là "Stochastic Gradient Descent" (SGD). Điểm yếu của thuật toán BGD là ta cần phải tính véc-tơ đạo hàm riêng cho **tất cả** các mẫu trong "batch" để cập nhật được một lần cho bộ trọng số. Thuật toán SGD khắc phục điểm yếu này bằng cách chọn ngẫu nhiên một số mẫu dữ liệu trong "batch" (gọi là "mini-batch"), tính véc-tơ đạo hàm riêng trung bình trên "mini-batch" này và thực hiện cập nhật bộ trọng số. Lúc

này, véc-tơ đạo hàm riêng trung bình trên "mini-batch" có thể coi là một xấp xỉ của véc-tơ đạo hàm riêng trung bình trên toàn bộ tập huấn luyện. Do véc-tơ này chỉ tính trên "mini-batch" nên khi cập nhật, giá trị $J(\theta)$ có thể tăng. Tuy nhiên, khi cập nhật nhiều lần thì xu hướng chung là hàm lỗi $J(\theta)$ sẽ giảm. Khi tập huấn luyện càng lớn thì lợi thế của thuật toán SGD càng rõ. Với tập huấn luyện gồm 1000 mẫu thì thuật toán BGD chỉ cập nhật trọng số được **một** lần sau khi duyệt qua hết dữ liệu; trong khi đó, thuật toán SGD với kích thước "mini-batch" là 10 có thể cập nhật được 100 lần. Kích thước của "mini-batch" lúc này ảnh hưởng đến độ chính xác của véc-tơ đạo hàm riêng của $J(\theta)$. Nếu kích thước quá nhỏ thì véc-tơ đạo hàm riêng trung bình trên "mini-batch" sẽ không còn là một xấp xỉ tốt. Nếu kích thước quá lớn thì lợi thế về tốc độ của SGD so với BGD sẽ không còn cao.

Như đã đề cập ở chương 2, để cải tiến chính sách khi không có đầy đủ thông tin về môi trường (tức không có các ma trận của MDP) ta cần xấp xỉ q_π thay vì v_π. Áp dụng thuật toán SGD để cực tiểu hoá hàm lỗi (3.7), công thức cập nhật bộ trọng số tại thời điểm t có dạng:

$$\begin{aligned}
\theta_{t+1} &= \theta_t - \Delta\theta_t \\
&= \theta_t - \alpha\nabla_{\theta_t} J(\theta_t) \\
&= \theta_t - \alpha\nabla_{\theta_t} \left(\frac{1}{B} \sum_{i=1}^{B} (q_\pi(S_i, A_i) - \hat{q}(S_i, A_i; \theta_t))^2 \right) \\
&= \theta_t - \alpha\frac{1}{B} \sum_{i=1}^{B} (q_\pi(S_i, A_i) - \hat{q}(S_i, A_i; \theta_t))\nabla_{\theta_t}\hat{q}(S_i, A_i; \theta_t)
\end{aligned} \tag{3.9}$$

Ở đây:

- $\Delta\theta_t$ là giá trị cập nhật bộ trọng số θ_t với một "mini-batch".
- B là kích thước của "mini-batch".
- $q_\pi(S_i, A_i)$ là giá trị **thật sự** của hành động A_i tại trạng thái S_i.
- $\hat{q}(S_i, A_i; \theta_t)$ là giá trị **xấp xỉ** (với bộ trọng số θ_t) của hành động A_i tại trạng thái S_i.

- $\nabla_{\theta_t}\hat{q}(S_i, A_i; \theta_t)$ là véc-tơ đạo hàm riêng theo θ_t của hàm $\hat{q}$ tại trạng thái S_i và hành động A_i.

Công thức trên bao gồm một tổng các số hạng có thể được tính riêng lẻ cho từng mẫu dữ liệu. Vậy để đơn giản hoá công thức, ta viết lại công thức trên cho duy nhất một mẫu dữ liệu; khi cần thiết lập công thức tổng quát cho một "mini-batch", ta chỉ việc tính giá trị trung bình của nhiều mẫu. Lúc này, công thức mới sẽ ứng với trường hợp kích thước "mini-batch" đúng bằng một nên ta có thể bỏ chỉ số i của từng mẫu dữ liệu:

$$\theta_{t+1} = \theta_t - \alpha(q_\pi(S, A) - \hat{q}(S, A; \theta_t))\nabla_{\theta_t}\hat{q}(S, A; \theta_t) \tag{3.10}$$

3.1.4 Học tăng cường kết hợp với xấp xỉ hàm

Áp dụng thuật toán SGD để tối ưu hàm lỗi bình phương theo công thức (3.9) thì ta sẽ cực tiểu hoá được độ khác biệt giữa hai hàm q_π và $\hat{q}$. Tuy nhiên, giá trị đích thực sự mà ta mong muốn là $q_\pi(s, a)$ ta không biết được mà chỉ có một số **mẫu** S_t, A_t khi tương tác với môi trường. Các thuật toán học tăng cường chính là kỹ thuật giúp ta có được một ước lượng đơn giản của giá trị đích này. Như ở chương 2, ta có hai thuật toán để ước lượng hàm giá trị: thuật toán "Monte-Carlo" (MC) và thuật toán "Temporal Difference" (TD).

Với thuật toán MC, ta chỉ việc thay thế $q_\pi(s, a)$ bằng một ước lượng không chệch của giá trị này. Theo định nghĩa:

$$q_\pi(s, a) = \mathbb{E}_\pi \left[R_{t+1} + \gamma R_{t+2} + ... + \gamma^{T-t-1} R_T \middle| \ S_t = s, A_t = a\right] \tag{3.11}$$

Vậy ta có thể lấy $R_{t+1} + \gamma R_{t+2} + ... + \gamma^{T-t-1} R_T$ làm một ước lượng cho $q_\pi(s, a)$. Giá trị này có thể dễ dàng có được bằng cách cho hệ thống tương tác với môi trường đến khi kết thúc một màn (tức một "episode"). Sau đó với mỗi trạng thái S_t của "episode", ta chỉ cần tính tổng điểm thưởng đến cuối "episode" để có giá trị ước lượng mong muốn. Đây là một ước lượng không chệch do kỳ vọng của biểu thức này bằng đúng $q_\pi(s, a)$. Vậy công thức cập nhật bộ trọng số trong

(3.10) được thay bằng:

$$\theta_{t+1} = \theta_t - \alpha(G_t - \hat{q}(S, A; \theta_t))\nabla_{\theta_t}\hat{q}(S, A; \theta_t) \tag{3.12}$$

G_t ở đây là tổng điểm thưởng (đã giảm điểm) nhận được khi thực hiện hành động A tại trạng thái S. Kết hợp thuật toán MC với thuật toán SGD để cực tiểu hoá hàm lỗi của mạng nơ-ron, ta có được một thuật toán học tăng cường kết hợp học sâu hoàn chỉnh để giải các bài toán có số trạng thái lớn.

Với ý tưởng tương tự, để sử dụng mạng nơ-ron để xấp xỉ hàm giá trị cho thuật toán "Q-learning", ta sử dụng tổng điểm thưởng được "bootstrap" để ước lượng $q_\pi(s, a)$. Cụ thể hơn, ta sẽ thay thế giá trị $q_\pi(s, a)$ trong công thức (3.11) bằng $R_{t+1} + \gamma \max_a \hat{q}(S_{t+1}, a; \theta)$. Có thể thấy rằng, giá trị "bootstrap" này là một ước lượng chệch của $q_\pi(s, a)$. Tuy vậy, ước lượng này chỉ gồm tổng của hai số hạng R_{t+1} và $\gamma \max_a \hat{q}(S_{t+1}, a; \theta)$ nên có phương sai thấp hơn nhiều so với ước lượng của MC (gồm tổng của nhiều số hạng R_{t+1}, γR_{t+2}, ...). Đây cũng chính là một sự "thoả hiệp" (trade-off) giữa hai giá trị "bias" và "variance" của hai thuật toán MC và "Q-learning". Thuật toán MC sử dụng một ước lượng không chệch nên có "bias" bằng không nhưng "variance" (tức phương sai) cao; khi đó tổng điểm thưởng của cùng một bộ (S, A) sẽ thay đổi rất nhiều. Khi các giá trị này thay đổi quá nhanh thì mạng nơ-ron sẽ khó học được hơn. Trong khi đó "Q-learning" sử dụng một ước lượng chệch nên có "bias" lớn nhưng phương sai lại nhỏ; khi đó giá trị "bootstrap" của cùng một bộ (S, A) sẽ ít thay đổi trong những lần duyệt đến khác nhau. Tương tự như thuật toán MC, công thức cập nhật bộ trọng số trong (3.10) được thay bằng:

$$\theta_{t+1} = \theta_t - \alpha \left[R_{t+1} + \gamma \max_a \hat{q}(S, a; \theta_t) - \hat{q}(S, A; \theta_t) \right] \nabla_{\theta_t}\hat{q}(S, A; \theta_t) \tag{3.13}$$

Mã giả của thuật toán "Q-learning" với xấp xỉ hàm được trình bày ở bảng 3.1.

Thuật toán 3.1 "Q-learning" kết hợp với xấp xỉ hàm

Đầu vào: Số "episode" cần thực hiện để cập nhật bộ trọng số mạng nơ-ron
Đầu ra: Bộ trọng số θ của mạng nơ-ron
Thao tác:

1: Khởi tạo ngẫu nhiên bộ trọng số θ của mạng nơ-ron
2: **repeat**
3: Tương tác với môi trường dựa vào chính sách có hàm giá trị được xấp xỉ bởi $\hat{q}(s, a; \theta)$ đến khi kết thúc "episode" để có được tập các mẫu dữ liệu $S_1, A_1, R_2, S_2, A_2, R_3, ..., S_{T-1}, A_{T-1}, R_T$
4: Chia tập dữ liệu trên thành các "mini-batch" gồm B mẫu dữ liệu có dạng S_i, A_i, R_{i+1}
5: **for** mỗi "mini-batch" của tập dữ liệu **do**
6: Cập nhật θ theo công thức (3.13) cho toàn bộ các phần tử trong "mini-batch"
7: **end for**
8: **until** Thực hiện đủ số "episode"

3.2 Kết hợp học tăng cường với học sâu vào bài toán tự động chơi game

Phần này trình bày cách kết hợp thuật toán học tăng cường với học sâu vào bài toán tự động chơi game. Đầu tiên, chúng em sẽ trình bày về cấu trúc mạng "Deep Q-Network" [13] - cấu trúc mạng nơ-ron tích chập kết hợp với mạng nơ-ron truyền thẳng được thiết kế riêng biệt cho bài toán tự động chơi game. Phần tiếp theo trình bày hai kỹ thuật quan trọng giúp tăng tính ổn định của quá trình học. Phần cuối cùng đề cập đến vấn đề "đánh giá quá cao" (overestimation) ảnh hưởng thế nào lên kết quả của hệ thống cũng như cách thức giải quyết vấn đề này.

3.2.1 "Deep Q-Network"

Để có được cấu trúc mạng "Deep Q-Network" [13] hoàn chỉnh và hoạt động tốt cho bài toán tự động chơi game, ta kết hợp thuật toán "Q-learning" với mạng nơ-ron tích chập có cấu trúc phù hợp với bài toán. Trong bài toán tự động chơi game, dữ liệu đầu vào mà hệ thống nhận được từ môi trường tại mỗi thời điểm

là một ảnh RGB có kích thước 210×160. Ta có thể đưa cả hình ảnh này làm dữ liệu đầu vào cho mạng nơ-ron tích chập; tuy nhiên với kích thước khá lớn như vậy, việc huấn luyện hệ thống sẽ tốn nhiều thời gian. Để tăng tốc độ huấn luyện lên, ta có thể thực hiện việc thu nhỏ (scale) ảnh về kích thước nhỏ hơn. Việc tiền xử lý ảnh bằng cách thu nhỏ có thể làm mất mát thông tin, tuy nhiên thực nghiệm cho thấy hệ thống vẫn có độ chính xác cao.

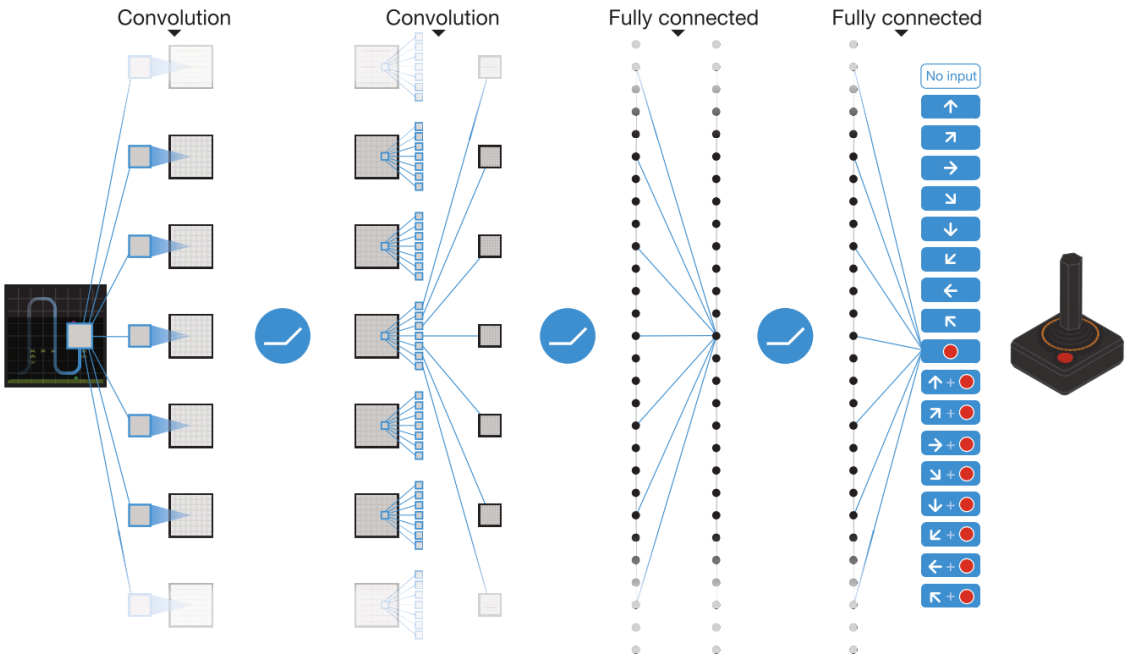
Một điểm quan trọng trong bài toán tự động chơi game là hình ảnh tại mội thời điểm không mô tả hết thông tin cần thiết. Ví dụ như khi có hình của một quả bóng, ta không biết hướng và vận tốc hiện tại của quả bóng. Để giải quyết điều này, ta có thể ghép hình ảnh của nhiều thời điểm liên tiếp lại theo thứ tự thời gian. Khi đó một trạng thái S_t sẽ gồm nhiều hình ảnh của các thời điểm liên tiếp nhau và chứa luôn cả thông tin về thời gian.

Với cách thiết kế mạng nơ-ron thông thường, ta cần cung cấp trạng thái S và hành động A để tính được giá trị xấp xỉ $\hat{q}(S, A; \theta)$. Cách thiết kế này không phù hợp cho thuật toán "Q-learning": ta cần phải lan truyền tiến nhiều lần để tính giá trị $\max_a \hat{q}(S, a; \theta)$ trong công thức (3.13). Để khắc phục nhược điểm này, ta chỉnh lại cấu trúc mạng để input là trạng thái S và output là giá trị của mọi hành động tại trạng thái này: $\hat{q}(S, a; \theta), \forall a$. Với cấu trúc này, ta chỉ việc lan truyền tiến một lần và lấy max giá trị output của mạng nơ-ron. Như vậy, số nơ-ron của tầng output là số lượng hành động có thể có.

Để tổng hợp các đặc trưng được học từ các tầng tích chập, các tầng ẩn cuối cùng của mạng nơ-ron sẽ là các tầng "fully-connected". Các tầng tích chập có chức năng tìm kiếm các đặc trưng cục bộ cần thiết còn tầng "fully-connected" có nhiệm vụ tổng hợp các đặc trưng đó trên **toàn ảnh**. Tầng output cũng là một tầng "fully-connected" với số nơ-ron là số hành động có thể có. Lưu ý là do tầng này trả về kết quả là giá trị của từng hành động nên ta không áp dụng hàm kích hoạt tại đây. Hình 3.5 mô tả cấu trúc mạng "Deep Q-Network".

3.2.2 Kỹ thuật làm tăng tính ổn định

Đặc điểm của thuật toán "Q-learning" khi kết hợp với xấp xỉ hàm đó là thuật toán không chắc sẽ hội tụ [16]. Hội tụ ở đây ý chỉ sau một số bước cập nhật hữu hạn, giá trị hành động của trạng thái sẽ hội tụ về giá trị nhất định. Do đặc



Hình 3.5: Hình mô tả cấu trúc mạng "Deep Q-Network" [13]. Các tầng ẩn đầu tiên là tầng tích chập với hàm kích hoạt "rectified linear". Tầng ẩn tiếp theo là tầng "fully-connected" có nhiệm vụ tổng hợp đặc trưng trên toàn ảnh của tầng trước. Tầng output cũng là tầng "fully-connected" trả về kết quả là giá trị của từng hành động ứng với trạng thái đầu vào. Tầng output không có hàm kích hoạt.

điểm này, nếu ta huấn luyện mạng nơ-ron bằng thuật toán SGD thông thường thì nhiều khả năng "Q-learning" sẽ không hội tụ dẫn đến chính sách tương ứng sẽ không được tốt. Để giải quyết vấn đề này, chúng em áp dụng hai kỹ thuật giúp tăng tính ổn định và khả năng hội tụ: kỹ thuật "Experience replay" [11] và kỹ thuật cố định "Q-target" [12].

"Experience replay"

Thuật toán "Q-learning" kết hợp với xấp xỉ hàm cho phép ta học "online": sau mỗi bước tương tác với môi trường và nhận được dữ liệu mới, ta có thể cập nhật ngay bộ trọng số mạng nơ-ron. Tuy nhiên, cách học này có thể gây ra vấn đề không hội tụ. Lý do là các mẫu dữ liệu liên tiếp nhau thường có tương quan (correlation) với nhau rất lớn. Ngoài ra, các mẫu dữ liệu phía sau bị ảnh hưởng bởi chính sách (tương ứng với bộ trọng số mạng nơ-ron) ngay trước đó. Ví dụ như chính sách hiện tại là "luôn qua trái" thì các mẫu dữ liệu liên tiếp sẽ lấy từ phân bố "qua trái". Khi chính sách thay đổi sang "luôn qua phải" thì tất cả các mẫu dữ liệu mới sẽ lấy từ phân bố "qua phải". Khi các mẫu dữ liệu có tương quan lớn và phân bố dữ liệu thay đổi nhanh thì "Q-learning" sẽ khó hội tụ [13].

Kỹ thuật "experience replay" [11] giải quyết vấn đề này bằng cách lưu trữ lại một tập các mẫu dữ liệu nhận được gần đây nhất. Sau đó, mỗi lần muốn cập nhật bộ trọng số, ta chỉ việc lấy ngẫu nhiên một "mini-batch" từ tập dữ liệu này để học. Ý tưởng của kỹ thuật này giống như việc ta "gợi nhớ" lại các kinh nghiệm đã có từ trước đó và học lại từ chúng. Cách làm này cũng mang tính "củng cố" lại những kinh nghiệm mà hệ thống học được trong quá khứ. Do việc lấy ngẫu nhiên nên các mẫu dữ liệu trong "mini-batch" sẽ không còn tương quan lớn với nhau. Ngoài ra, việc lưu trữ lại giúp các mẫu dữ liệu có thể được học lại nhiều lần. Điều này giúp hệ thống học được nhiều hơn mà không phải tương tác với môi trường quá nhiều.

Cố định "Q-target"

Mỗi khi mạng nơ-ron được cập nhật bộ trọng số thì chính sách hiện tại sẽ thay đổi. Khi chính sách thay đổi thì hàm q_π mục tiêu cũng thay đổi theo. Thực tế cho thấy, việc thay đổi nhỏ của bộ trọng số sẽ gây ra thay đổi lớn của chính

sách [12]. Để thấy được điều này, ta có thể thay đổi trọng số ở những tầng đầu tiên của mạng nơ-ron. Khi đó, trọng số này sẽ ảnh hưởng dây chuyền đến những tầng sau làm cho output của mạng nơ-ron bị thay đổi nhiều. Do vậy, mỗi khi cập nhật bộ trọng số, hàm mục tiêu q_π tại một bộ trạng thái s và hành động a nhiều khả năng sẽ thay đổi lớn. Do giá trị mục tiêu của việc xấp xỉ hàm thay đổi quá nhiều và liên tục, mạng nơ-ron có thể không xấp xỉ được hàm giá trị như ta mong muốn.

Một kỹ thuật được đề xuất bởi [13] có tên **cố định "Q-target"** (fix Q-target) nhằm giải quyết vấn đề này. Thay vì thay đổi giá trị mục tiêu một cách liên tục, ta có thể cố định lại hàm mục tiêu q_π trong một số bước cập nhật trọng số để mạng nơ-ron xấp xỉ tốt hàm này. Sau một khoảng thời gian, ta mới thay đổi hàm mục tiêu thành hàm ứng với chính sách hiện tại. Các lần cập nhật trọng số tiếp theo lại tiếp tục cố gắng xấp xỉ tốt hàm mục tiêu mới. Quá trình này được lặp lại nhiều lần để hàm mục tiêu cuối cùng mà ta có cũng chính là hàm giá trị tối ưu.

Để có cái nhìn trực quan hơn về kỹ thuật này, ta có thể lấy ví dụ về việc xây một căn nhà. Do quá trình thiết kế ban đầu chưa tính trước đến những trường hợp phát sinh (như thiếu vật liệu, thợ không đủ trình độ...) trong lúc xây dựng nên ta phải điều chỉnh thiết kế lại liên tục. Ta có thể vừa xây dựng căn nhà (ứng với việc thay đổi trọng số mạng nơ-ron) vừa điều chỉnh lại thiết kế căn nhà cho phù hợp (ứng với việc thay đổi chính sách mục tiêu). Tuy nhiên, khi thiết kế căn nhà thay đổi quá nhanh và liên tục thì việc xây dựng sẽ gặp nhiều khó khăn và có khả năng khi hoàn thành kết quả sẽ không tốt. Kỹ thuật cố định "Q-target" tương tự với việc ta cố gắng xây dựng hoàn tất một phần của căn nhà (tức xấp xỉ tốt hàm giá trị một chính sách cũ) trước khi thay đổi thiết kế. Cứ như vậy, mỗi khi hoàn tất một phần, ta thay đổi thiết kế (ứng với việc thay đổi hàm mục tiêu) rồi lại xây dựng tiếp. Cách làm này giống như tạo dựng một số các "điểm kiểm soát" (checkpoint) với những mục tiêu xác định. Do các "điểm kiểm soát" này cố định nên việc thay đổi kiến trúc căn nhà sẽ không quá nhiều, dẫn đến quá trình xây dựng sẽ dễ dàng đạt được mục tiêu hơn.

Tuy nhiên, kỹ thuật này dẫn đến một vấn đề là ta cần phải thay đổi hàm mục tiêu sau bao nhiêu lần cập nhật trọng số mạng nơ-ron (tức tần suất thay

đổi hàm mục tiêu)? Nếu tần suất quá cao (tức thay đổi hàm quá nhiều) thì kỹ thuật này không mang lại được nhiều hiệu quả. Nếu tần suất quá thấp (tức ít khi thay đổi hàm mục tiêu) thì việc học sẽ rất chậm do ta phải cập nhật trọng số rất nhiều để xấp xỉ một hàm giá trị cũ; khi cập nhật hàm mục tiêu thì chính sách mới chỉ tốt lên một chút so với chính sách cũ. Vì vậy, ta cần phải chọn một giá trị vừa phải, phù hợp với bài toán tự động chơi game.

3.2.3 Vấn đề đánh giá quá cao của thuật toán "Q-learning"

Thuật toán "Q-learning" thực hiện việc lấy max giá trị các hành động của trạng thái tiếp theo. Nhắc lại công thức cập nhật của "Q-learning" là:

$$\theta_{t+1} = \theta_t - \alpha \left[R_{t+1} + \gamma \max_a \hat{q}(S, a; \theta_t) - \hat{q}(S, A; \theta_t) \right] \nabla_{\theta_t} \hat{q}(S, A; \theta_t) \qquad (3.14)$$

Nhờ việc lấy max mà thuật toán "Q-learning" hội tụ nhanh hơn [16]: ta luôn ước lượng giá trị của trạng thái tiếp theo dựa vào hành động tốt nhất tại đó. Đây có thể hiểu là một cách nhìn "lạc quan" về tương lai: nếu có nhiều trường hợp tốt, xấu khác nhau thì ta luôn giả sử điều tốt nhất sẽ xảy ra. Tuy việc giả sử này có thể không đúng trong mọi tình huống, do các trạng thái tiếp theo sau đó cũng được cập nhật lại giá trị nên thuật toán "Q-learning" vẫn hội tụ chính xác về chính sách tối ưu.

Rõ ràng là cách nhìn "lạc quan" này cũng có điểm xấu: ta sẽ có cái nhìn quá "lạc quan" (optimistic) về tương lai và dễ dẫn đến khả năng đánh giá quá cao (overestimation) giá trị của trạng thái kế tiếp. Nhất là khi áp dụng xấp xỉ hàm, giá trị ước lượng hiện tại của mạng nơ-ron chưa được chính xác. Khi đó nếu mạng nơ-ron ước lượng cao hơn giá trị thật sự thì vấn đề đánh giá quá cao sẽ xảy ra nhiều hơn. Mặc dù vậy, ta vẫn cần đặt ra câu hỏi: liệu việc đánh giá quá cao có gây ảnh hưởng đến kết quả của chính sách cuối cùng hay không? Rõ ràng là nếu tất cả hành động đều bị đánh giá quá cao (ví dụ như tăng một lượng bằng nhau C) thì việc lựa chọn hành động không bị ảnh hưởng (do ta vẫn chọn hành động tốt nhất bằng cách tham lam). Tuy nhiên, thuật toán "Q-learning" với xấp xỉ hàm gặp hiện tượng đánh giá quá cao không đồng đều [19]. Điều này làm cho kết quả của một số bài toán cụ thể bị ảnh hưởng lớn. Nghiên cứu của

[8] cho thấy sai số do môi trường gây ra cũng gây ảnh hưởng đến việc xấp xỉ hàm và gây nên hiện tượng đánh giá quá cao. Các nghiên cứu trên cho thấy vấn đề này có ảnh hưởng lớn đến chính sách tìm được của thuật toán "Q-learning".

Để khắc phục vấn đề đánh giá quá cao, thuật toán "Double Q-learning" [8] ra đời và được áp dụng vào bài toán tự động chơi game [19]. Đây là một thuật toán cải tiến của "Q-learning" bằng cách thay đổi lại cách ước lượng giá trị của trạng thái tiếp theo. Cụ thể hơn, giá trị này theo "Q-learning" là:

$$R_{t+1} + \gamma \max_a \hat{q}(S_{t+1}, a; \theta) \tag{3.15}$$

Ta có thể tách việc lấy max giá trị ra thành hai bước: chọn hành động tối ưu và đánh giá hành động đó. Bước chọn hành động được thực hiện bằng cách chọn hành động cho giá trị cao nhất tại trạng thái đó: $a_* = \arg\max_a \hat{q}(S_{t+1}, a; \theta)$. Bước đánh giá hành động được thực hiện bằng cách lấy lại giá trị của hành động tốt nhất: $\hat{q}(S_{t+1}, a_*; \theta)$. Ghép hai bước này lại ta có cách ước lượng giá trị của thuật toán "Q-learning":

$$R_{t+1} + \gamma \hat{q}(S_{t+1}, \arg\max_a \hat{q}(S_{t+1}, a; \theta); \theta) \tag{3.16}$$

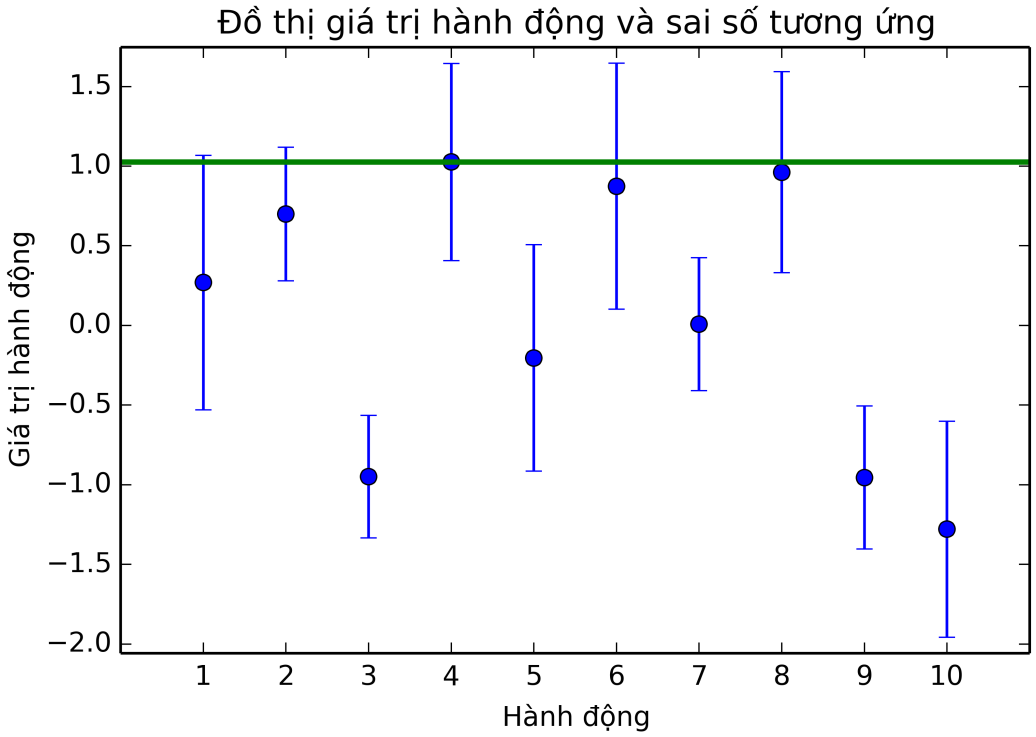
Lưu ý rằng công thức (3.16) chỉ là một cách nhìn khác của công thức (3.15) và hai công thức này hoàn toàn tương đương nhau.

Thuật toán "Double Q-learning" giải quyết vấn đề đánh giá quá cao bằng cách đánh giá hành động tốt nhất bằng hàm giá trị của chính sách khác với chính sách hiện tại. Cụ thể hơn, ta cần học hai mạng nơ-ron với hai bộ tham số θ và θ^-. Khi cần cập nhật bộ tham số θ của mạng đầu tiên, ta áp dụng công thức:

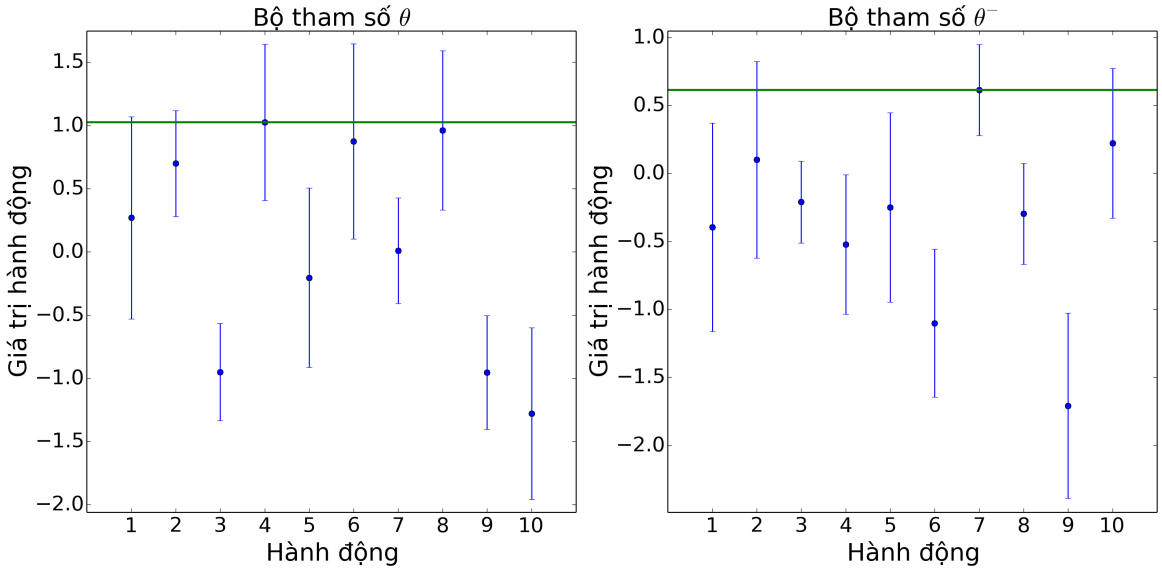
$$R_{t+1} + \gamma \hat{q}(S_{t+1}, \arg\max_a \hat{q}(S_{t+1}, a; \theta); \theta^-) \tag{3.17}$$

Công thức trên chỉ khác với công thức (3.16) ở chỗ ta đánh giá hành động được chọn bằng một mạng nơ-ron khác. Nhờ việc "phân tách" (decouple) hai bước trên, "Double Q-learning" làm giảm khả năng bị đánh giá quá cao mà "Q-learning" gặp phải ([8]). Vậy tại sao cách cập nhật này lại không gặp vấn đề nêu trên? Trong "Q-learning", ta cần tính max của giá trị của nhiều hành động. Do

mỗi giá trị hành động tính được đều là xấp xỉ nên sẽ có sai số nhất định (tức sai số giữa giá trị xấp xỉ và giá trị thực $\hat{q}(s, a; \theta) - q_\pi(s, a)$). Giả sử sai số này là "uniform" trong đoạn $[-\epsilon, \epsilon]$ với ϵ là một giá trị dương nhỏ. Khi lấy max giá trị nhiều hành động lại thì nhiều khả năng hành động được chọn sẽ có sai số dương (do các hành động có sai số âm thì giá trị xấp xỉ bị giảm so với giá trị đúng nên khả năng được chọn thấp hơn). Khi càng có nhiều hành động thì khả năng hành động đó có sai số dương lại càng cao hơn. Điều này làm cho "Q-learning" gặp vấn đề đánh giá quá cao. Trong khi với thuật toán "Double Q-learning", gọi hành động được chọn theo bộ tham số θ là a_*. Khi đó, xác suất để a_* cũng có sai số dương theo bộ tham số θ^- là thấp (do hai bộ tham số học từ dữ liệu khác nhau). Nhờ vậy, việc phân tách hai bước chọn và đánh giá hành động giúp làm giảm vấn đề đánh giá quá cao. Xem hình 3.6 và 3.7 mô tả một ví dụ đơn giản để hiểu hơn về vấn đề đánh giá quá cao trong hai thuật toán "Q-learning" và "Double Q-learning".



Hình 3.6: Hình mô tả vấn đề đánh giá quá cao của thuật toán "Q-learning". Trục hoành thể hiện mười hành động và trục tung thể hiện giá trị tương ứng. Chấm tròn thể hiện giá trị thực sự của từng hành động và các đoạn song song với trục tung thể hiện sai số do quá trình xấp xỉ. Đường thẳng nằm ngang thể hiện max của giá trị thực sự của tất cả các hành động. Đây là giá trị đích mong muốn của "Q-learning". Có nhiều hành động có giá trị xấp xỉ với sai số dương lớn hơn giá trị đích này (hành động 4, 6 và 8) nên khi lấy max theo "Q-learning" thì ta gặp tình trạng đánh giá quá cao.



Hình 3.7: Hình mô tả cách giải quyết vấn đề đánh giá quá cao của thuật toán "Double Q-learning". Đồ thị bên trái thể hiện giá trị hành động được xấp xỉ theo bộ tham số θ; đồ thị bên phải là giá trị hành động theo bộ tham số θ^-. Khi chọn hành động theo bộ tham số θ, nhiều khả năng ta sẽ chọn hành động 4, 6 hoặc 8. Tuy nhiên, theo bộ tham số θ^- thì các hành động này lại không phải là những hành động có giá trị xấp xỉ cao nhất. Nhờ việc lựa chọn và đánh giá hành động một cách độc lập như vậy mà thuật toán "Double Q-learning" không gặp vấn đề đánh giá quá cao.

Chương 4

Kết quả thực nghiệm

Trong chương này, chúng em trình bày kết quả thực nghiệm của bài toán tự động chơi game. Đầu tiên, môi trường giả lập của bài toán tự động chơi game cùng với các thiết lập thực nghiệm được giới thiệu. Phần tiếp theo, chúng em trình bày về kết quả thực nghiệm của thuật toán "Q-learning" để chứng minh tính khả thi của việc kết hợp học tăng cường với học sâu vào bài toán tự động chơi game. Kế tiếp, chúng em phân tích mức độ ảnh hưởng của vấn đề đánh giá quá cao và kết quả của thuật toán "Double Q-learning". Phần cuối cùng là một thực nghiệm nhằm phân tích ý nghĩa hàm giá trị học được.

4.1 Giới thiệu Arcade Learning Environment

Arcade Learning Environment (ALE) [3] là một framework được thiết kế giúp dễ dàng trong việc phát triển những hệ thống có thể tự động chơi game trên hệ máy Atari 2600. Do ALE giúp ta giả lập game Atari trên máy vi tính mà không bị giới hạn số khung hình/giây, việc huấn luyện được tăng tốc đáng kể. Hình ảnh của những game này là ảnh RGB với kích thước 210×160. Số lượng hành động tối đa của một game là 18.

ALE cung cấp một giao diện lập trình cho ngôn ngữ Python. Và quan trọng nhất đó là ALE giúp mô hình hoá các game trên Atari về thành cái bài toán học tăng cường. Cụ thể hơn, ta có thể lấy frame ảnh hiện tại, lấy điểm số của hành động vừa thực hiện... Cuối cùng, ALE hỗ trợ khá nhiều game (hơn 50 game) để

ta có thể thử nghiệm tính tổng quát của mô hình.

4.2 Các thiết lập thực nghiệm

Chúng em thực hiện huấn luyện hệ thống trên ba trò chơi: *Assault*, *Breakout* và *Seaquest*. Việc thực nghiệm trên ba trò chơi là để chứng minh hướng tiếp cận kết hợp học sâu với học tăng cường có thể áp dụng vào nhiều trò chơi khác nhau. *Assault* là trò chơi bắn máy bay truyền thống: hệ thống điều khiển máy bay bay qua trái, phải hoặc bắn tên lửa để hạ máy bay địch. *Breakout* là trò chơi "phá gạch": hệ thống điều khiển một mái chèo (paddle) qua trái, phải để giữ cho trái banh không rơi xuống dưới; phía trên là những viên gạch bị phá huỷ và cho điểm thưởng khi chạm phải trái banh. Cuối cùng là trò chơi *Seaquest* có nội dung tương tự như "Assault" nhưng là điểu khiển tàu ngầm theo cả tám hướng xung quanh. Tuy các trò chơi này có nội dung tương tự nhau nhưng luật chơi cụ thể lại rất khác biệt và đòi hỏi chiến thuật mang tính "dài hơi". Ví dụ như màn hình trò chơi *Seaquest* có một thanh không khí thể hiện mức độ không khí còn lại trong tàu ngầm. Khi thanh này cạn thì ta cần điều khiển tàu ngầm nổi lên mặt nước để lấy không khí. Vì vậy, hệ thống cần học được đặc trưng thể hiện mức không khí còn lại và dựa vào đó để điều khiển tàu ngầm nổi lên mặt nước. Ngoài ra, để chơi tốt và đạt nhiều điểm, hệ thống cần học được chiến thuật đợi không khí còn lại ít nhất rồi mới nổi lên mặt nước một lần để tiết kiệm thời gian. Ví dụ này cho thấy các trò chơi này đều đòi hỏi các đặc trưng học được phải rất trừu tượng và chiến thuật chơi cũng phải khá phức tạp. Hình () thể hiện màn hình một khung ảnh của ba trò chơi trên.

Để giảm bớt tính toán của mô hình, khung ảnh RGB đầu vào được chuyển về ảnh mức xám (grayscale) và được giảm kích thước về 84×84. Để lưu thông tin về sự di chuyển và tốc độ của các đối tượng trong ảnh, ta ghép bốn khung ảnh liên tiếp nhau lại thành một trạng thái. Vậy một trạng thái lúc này sẽ gồm bốn ảnh mức xám 84×84 và được đưa trực tiếp vào input của mạng nơ-ron. Ngoài ra, do các khung hình gần nhau thường không thay đổi nhiều, một kỹ thuật đơn giản để tăng tốc độ huấn luyện đó là ta lặp lại một hành động cho k khung hình liên tiếp. Nhờ kỹ thuật này, ta có thể huấn luyện nhiều hơn k lần so

với khi không áp dụng. Trong thực nghiệm này, chúng em chọn k bằng 4 theo đề xuất của ([12]).

Cấu trúc mạng nơ-ron tích chập được dùng để xấp xỉ hàm giá trị là "Deep Q-Network" ([13]). Mạng này bao gồm bốn tầng ẩn với ba tầng đầu là tầng tích chập và tầng cuối là tầng "Fully-connected". Tầng tích chập đầu tiên gồm 32 bộ lọc với kích thước 8×8 và bước dịch chuyển (stride) là 4×4. Tầng tích chập thứ hai gồm 64 bộ lọc với kích thước 4×4 và bước dịch chuyển là 2×2. Tầng tích chập cuối cùng gồm 64 bộ lọc với kích thước 3×3 và bước dịch chuyển là 1×1. Đầu ra của tầng tích chập được đưa vào một tầng "Fully-connected" gồm 512 nơ-ron ẩn để tổng hợp đặc trưng. Tầng output của mạng cũng là một tầng "Fully-connected" với số nơ-ron ẩn là số hành động của trò chơi (6 cho "Assault", 4 cho "Breakout" và 18 cho "Seaquest"). Tầng này sẽ trả về giá trị hành động tương ứng cho trạng thái đầu vào. Hàm kích hoạt của tất cả các tầng ẩn là hàm "Rectified": $\sigma(x) = \max(0, x)$.

Để huấn luyện mạng nơ-ron này, chúng em áp dụng thuật toán "RMSprop" ([18]) để tối tiểu hàm lỗi bình phương. Đây là một thuật toán cải tiến của SGD với ý tưởng chọn một hệ số học khác nhau cho từng trọng số của mạng nơ-ron. Hệ số học của mỗi trọng số sẽ phụ thuộc vào giá trị đạo hàm của hàm lỗi đối với trọng số này ở các bước cập nhật trước. Cụ thể hơn, nếu trọng số này được cập nhật ít ở những lần cập nhật trước thì hệ số học sẽ được tăng lên; trọng số được cập nhật nhiều thì sẽ có hệ số học giảm đi. Nhờ thuật toán này mà mức độ thay đổi của các trọng số là tương đương nhau.

Chúng em sử dụng Python (kết hợp với framework "Theano" ([17])) làm ngôn ngữ lập trình chính. Do việc huấn luyện đòi hỏi sức mạnh tính toán rất lớn nên chúng em thực hiện cài đặt tính toán song song trên GPU để tăng tốc. GPU được dùng là NVIDIA GTX 980.

Với các thiết lập thực nghiệm trên, chúng em tiến hành huấn luyện ba trò chơi với hai thuật toán "Q-learning" và "Double Q-learning". Mỗi trò chơi với mỗi thuật toán được huấn luyện 50 triệu hành động. Thời gian huấn luyện là khoảng 50 giờ cho mỗi trò chơi với mỗi thuật toán.

4.3 Thuật toán "Q-learning"

Để thực nghiệm khả năng học và tự động chơi game, chúng em thực hiện thử nghiệm thuật toán "Q-learning" kết hợp với học sâu. Chúng em sử dụng chính sách ϵ-greedy để làm chính sách khám phá không gian trạng thái và không gian hành động. ϵ-greedy được chọn ở đây là vì chính sách này đơn giản cho việc cài đặt mà vẫn đảm bảo khả năng khám phá không gian cũng như khai thác kiến thức đã học của mô hình. Do ban đầu hệ thống chưa có nhiều hiểu biết về môi trường, chúng ta nên khám phá nhiều hơn; sau khi có một số hiểu biết nhất định thì hệ thống lại cần khai thác những hiểu biết đó. Ý tưởng này có thể dễ dàng được cài đặt bằng cách giảm dần hệ số ϵ trong khi đang huấn luyện. Với thuật toán "Q-learning", chúng em thực hiện giảm dần ϵ từ 1.0 về 0.1 trong 1 triệu hành động đầu tiên.

Để tiện cho việc quan sát kết quả và thử nghiệm hệ thống, chúng em chia quá trình huấn luyện 50 triệu hành động ra thành 200 "epoch" (tức mỗi "epoch" bao gồm 250000 hành động). Sau mỗi "epoch", chúng em kiểm thử bộ trọng số học được trên 30 màn chơi và lưu lại điểm số trung bình. Kết quả điểm số cuối cùng được báo cáo của mỗi trò chơi là giá trị điểm số lớn nhất của 200 epoch. Các thiết lập trên tương đồng với ([13]) để việc so sánh là công bằng nhất.

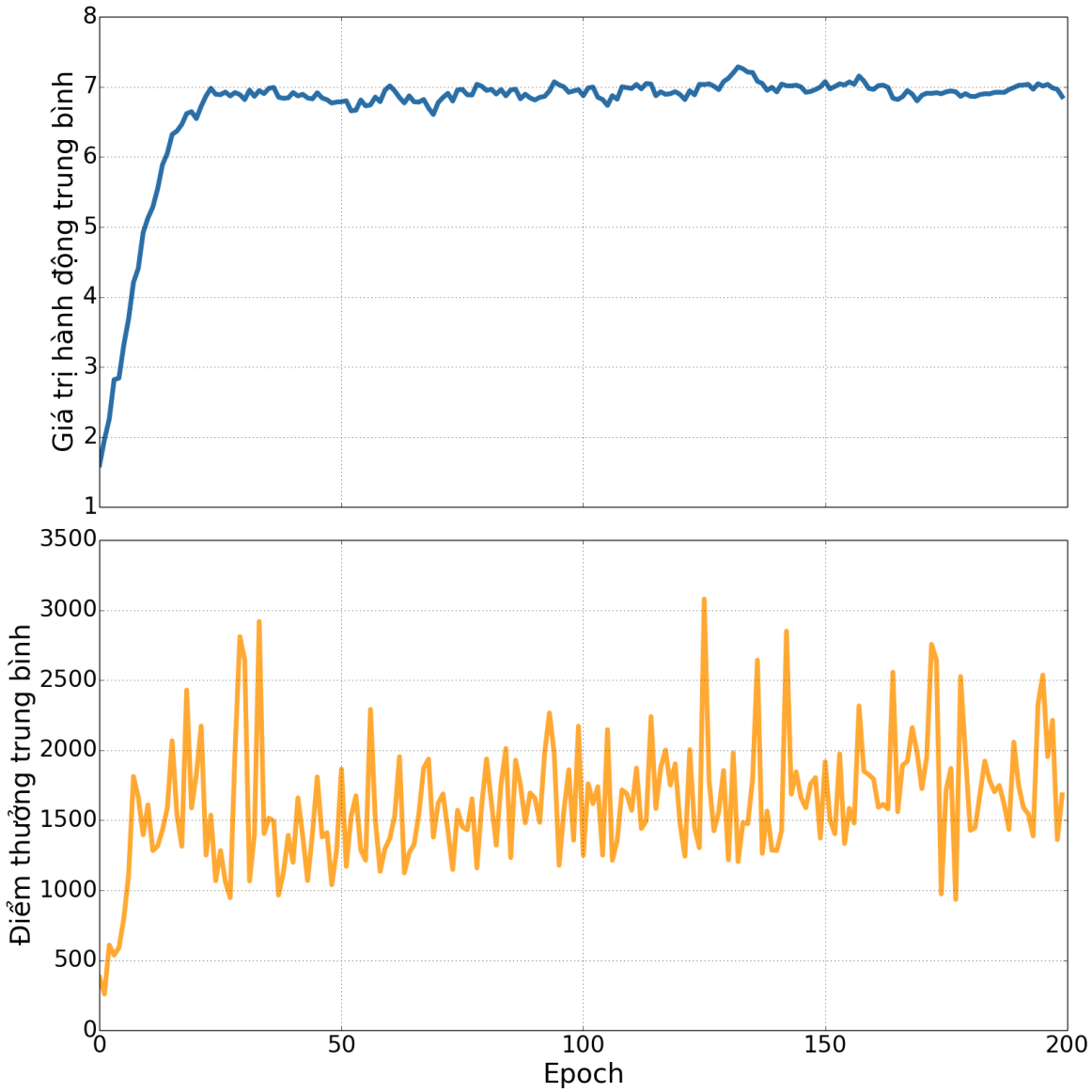
Một đặc điểm của bài toán tự động chơi game đó là kết quả điểm thưởng sau từng "epoch" thường rất nhiễu. Giá trị của các trọng số mạng nơ-ron có thể thay đổi ít sau mỗi "epoch" nhưng hàm giá trị tương ứng sẽ thay đổi nhiều. Điều này dẫn đến chính sách tương ứng cũng thay đổi nhiều làm cho điểm thưởng nhận được khi chơi game của các chính sách này sẽ rất khác nhau. Vì vậy nếu ta chỉ quan sát đồ thị điểm thưởng của quá trình kiểm thử sau mỗi "epoch", ta sẽ không biết chắc là hệ thống có đang chơi tốt lên hơn không. Một cách khác để quan sát quá trình học được đề xuất bởi ([12]) đó là ta quan sát đồ thị giá trị của một tập trạng thái sau từng "epoch". Đầu tiên, ta thực hiện chơi game một cách ngẫu nhiên và lưu lại một tập các trạng thái gặp được trong quá trình chơi này (gọi là tập "hold-out"). Trong quá trình huấn luyện, sau mỗi "epoch", ta tính trung bình giá trị hành động tốt nhất của các trạng thái trong tập "hold-out". Một chính sách chơi tốt thì thường sẽ đánh giá các hành động của các trạng thái

Bảng 4.1: Kết quả *Ngẫu nhiên* nhận được bằng cách chơi với chính sách ngẫu nhiên (chọn hành động theo phân bố đều). Kết quả *Con người* nhận được bằng cách cho một số người học và chơi các game này. *Kết quả gốc* là kết quả thực nghiệm của bài báo [13]. Kết quả *Thực nghiệm* là kết quả do nhóm thực hiện cài đặt lại thuật toán "Q-learning". Giá trị được in đậm là giá trị lớn nhất của mỗi dòng.

	Ngẫu nhiên[13]	*Con người*[13]	*Kết quả gốc*[13]	*Thực nghiệm*
Assault	222.4	742.0	**4,280.4**	3,078.8
Breakout	1.7	30.5	**385.5**	377.6
Seaquest	68.4	**42,054.7**	5,860.6	6,340.7

trong tập "hold-out" cao hơn so với các chính sách chơi không tốt. Vì vậy, quan sát đồ thị này là một cách đơn giản để theo dõi quá trình học của hệ thống.

Hình 4.1 thể hiện đồ thị điểm số và giá trị trạng thái trên tập "hold-out" của trò chơi *Assault*. Điểm thưởng kết quả của ba game được báo cáo trong bảng (4.1). Cả ba game được thực nghiệm cho kết quả vượt xa kết quả chơi ngẫu nhiên. Điều này cho thấy hệ thống thật sự học được cách chơi để đạt được nhiều điểm số dù ban đầu không hề biết quy luật chơi. Hai trong số ba game còn cho kết quả hơn cả kết quả do con người chơi. Tuy nhiên, bài báo [13] chỉ cho người chơi học thử khoảng 2 giờ cho mỗi game, ít hơn nhiều so với việc huấn luyện 50 triệu hành động (tức khoảng 38 ngày chơi) của hệ thống. Vì vậy, điểm thưởng nhận được của con người trong bảng (4.1) chỉ mang tính tham khảo. Dù chúng em thực hiện cài đặt lại thuật toán được đề xuất bởi bài báo [13] nhưng kết quả cũng không bằng nhau (cao hơn ở game *Seaquest* và thấp hơn ở hai game còn lại). Lý do chính của điều này là vì cách cài đặt khác nhau và chương trình giả lập game Atari khác nhau.



Hình 4.1: Đồ thị kết quả của trò chơi *Assault*. Đồ thị ở trên thể hiện trung bình giá trị hành động trên tập "hold-out" theo "epoch" huấn luyện; đồ thị ở dưới thể hiện điểm thưởng trung bình theo "epoch" huấn luyện. Điểm thưởng trung bình thay đổi rất nhiều sau từng "epoch". Nếu sử dụng đồ thị này để theo dõi quá trình huấn luyện thì ta sẽ không biết là hệ thống có đang học được cách chơi game hay không. Trong khi đó, đồ thị giá trị hành động lại ít nhiễu hơn và hội tụ dần đến một mức nhất định. Quan sát đồ thị này, ta thấy những "epoch" đầu hệ thống học rất nhanh và sau khoảng 50 "epoch" thì giá trị hành động không tăng cao hơn nữa. Lúc này, ta có thể dừng việc học sớm. Tuy nhiên, một số game có thể cần học lâu hơn. Vì vậy, chúng em cố định số "epoch" để huấn luyện được nhiều trò chơi mà không phải thay đổi siêu tham số.

4.4 Thuật toán "Double Q-learning"

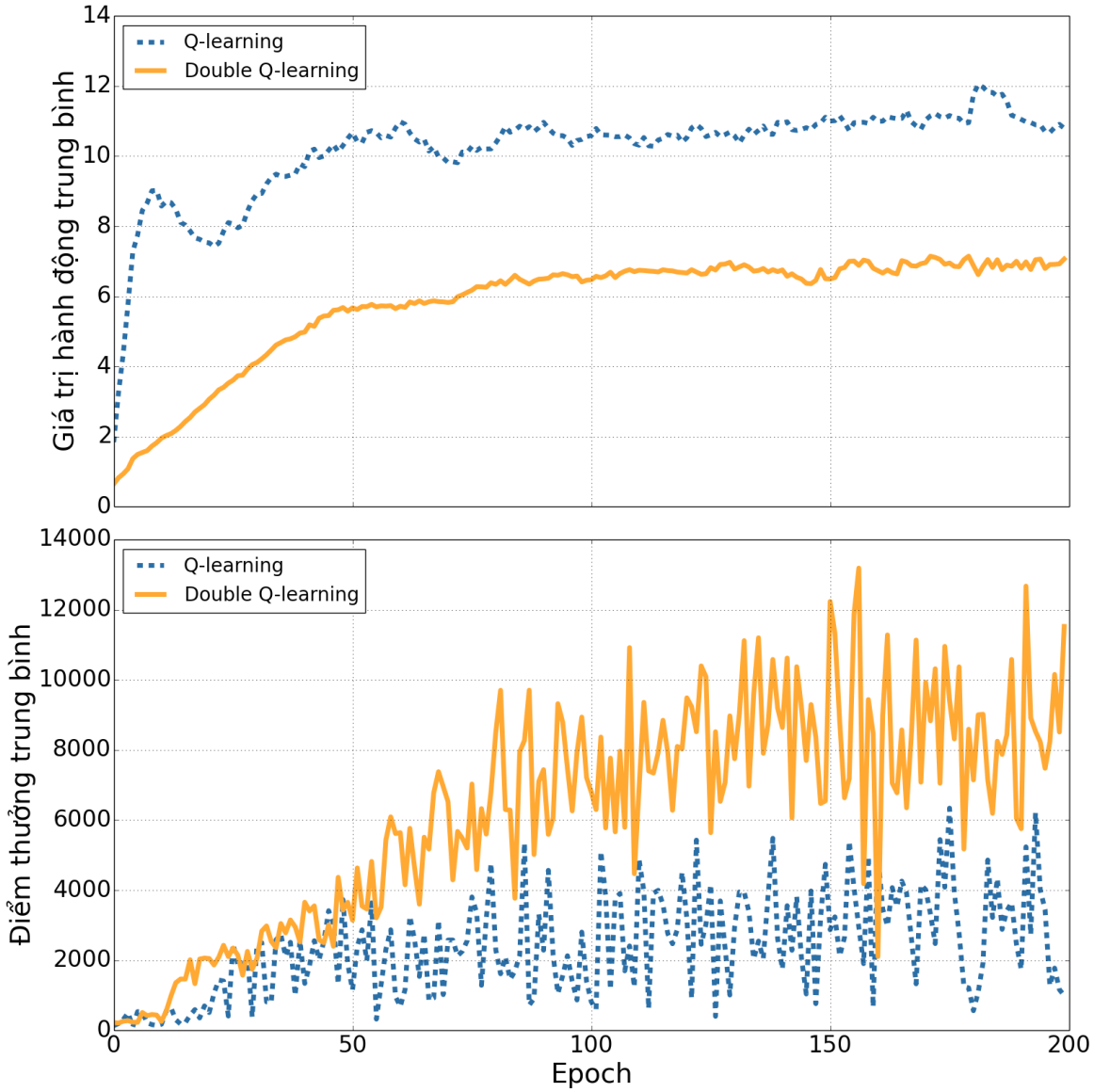
Chúng em thực hiện cài đặt và thử nghiệm lại thuật toán "Double Q-learning" [8], [19] để giải quyết vấn đề đánh giá quá cao. Các thiết lập thực nghiệm đều giống như khi thực nghiệm thuật toán "Q-learning". Tuy nhiên, để tương đồng với cài đặt của bài báo [19], chúng em giảm hệ số ϵ từ 1.0 xuống 0.01 (thay vì 0.1 như ở phần thực nghiệm "Q-learning") trong 1 triệu hành động đầu tiên của quá trình huấn luyện.

Bảng 4.2 cho biết kết quả của thuật toán "Double Q-learning" trên ba game. Thuật toán "Double Q-learning" cho kết quả tốt hơn hẳn điểm số của "Q-learning" trên cả ba game và cao hơn gấp đôi ở game *Seaquest*. Lý do "Double Q-learning" hoạt động tốt như vậy trên game *Seaquest* là vì game này có nhiều hành động (18 so với 4 của *Breakout* và 6 của *Assault*). Số lượng hành động càng nhiều thì khả năng bị đánh giá quá cao lại càng tăng. Vì vậy, thuật toán "Q-learning" bị ảnh hưởng rất lớn bởi vấn đề đánh giá quá cao và thuật toán "Double Q-learning" cho kết quả tốt hơn hẳn khi giải quyết được vấn đề này. Các kết quả này cho thấy thuật toán "Double Q-learning" là một cải tiến rất tốt cho bài toán tự động chơi game.

Để thấy rõ hơn việc hạn chế vấn đề đánh giá quá cao của "Double Q-learning", ta có thể coi đồ thị giá trị hành động trung bình trên tập "hold-out" và đồ thị điểm thưởng trung bình của cả hai thuật toán ở hình 4.2. Thuật toán "Double Q-learning" cho kết quả điểm thưởng tốt hơn hẳn "Q-learning" vì đánh giá hành động chính xác hơn (không bị vấn đề đánh giá quá cao).

Bảng 4.2: Kết quả *Ngẫu nhiên* và *Con người* được trích từ bài báo [19]. Do môi trường giả lập game khác nhau nên chúng em chỉ so sánh "Double Q-learning" với phiên bản cài đặt "Q-learning" của nhóm thay vì so sánh với phiên bản "Double Q-learning" của bài báo [19]. Giá trị được in đậm là giá trị lớn nhất của mỗi dòng.

	Ngẫu nhiên[13]	*Con người*[13]	*Q-learning*	*Double Q-learning*
Assault	222.4	742.0	3,078.8	**4864.6**
Breakout	1.7	30.5	377.6	**431.2**
Seaquest	68.4	**42,054.7**	6,340.7	13,186

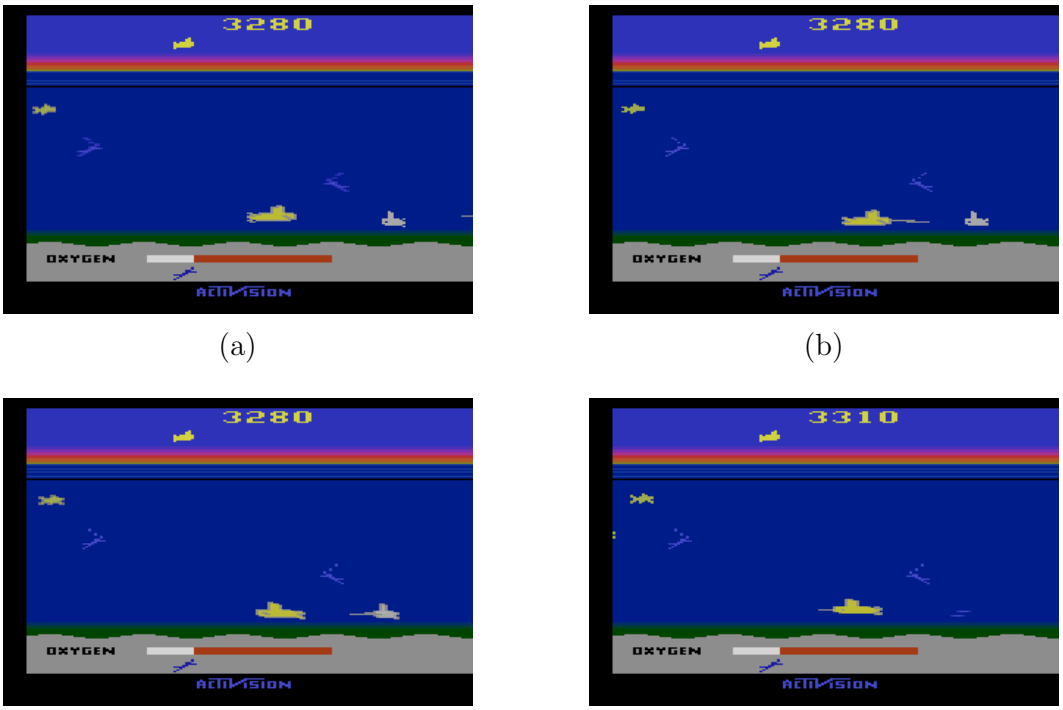


Hình 4.2: Đồ thị kết quả của trò chơi *Seaquest*. Hình phía trên là đồ thị giá trị hành động trung bình theo "epoch"; hình phía dưới là đồ thị điểm thưởng trung bình theo "epoch". Ở đồ thị phía trên, thuật toán "Q-learning" đánh giá rất cao các hành động của trạng thái thuộc tập "hold-out". Tuy nhiên, khi chơi thử game thì thuật toán "Q-learning" lại cho điểm thưởng rất thấp. Điều này cho thấy thuật toán "Q-learning" gặp vấn đề đánh giá quá cao và vấn đề này ảnh hưởng đến kết quả chơi game của hệ thống. Trong khi đó, thuật toán "Double Q-learning" tuy đánh giá các hành động của trạng thái thuộc tập "hold-out" thấp hơn nhưng khi chơi game thì lại được nhiều điểm thưởng hơn. Thuật toán "Double Q-learning" đánh giá chính xác hơn giá trị hành động và vì thế, chính sách chơi game cũng tốt hơn.

4.5 Phân tích hàm giá trị hành động

Để thấy rõ hơn khả năng xấp xỉ hàm giá trị của mạng nơ-ron, ta có thể xem xét giá trị của một số trạng thái trong quá trình chơi game. Cụ thể hơn, chúng em thực hiện lưu một số frame ảnh trong quá trình chơi cùng với giá trị của chúng được tính từ mạng nơ-ron (với thuật toán "Q-learning").

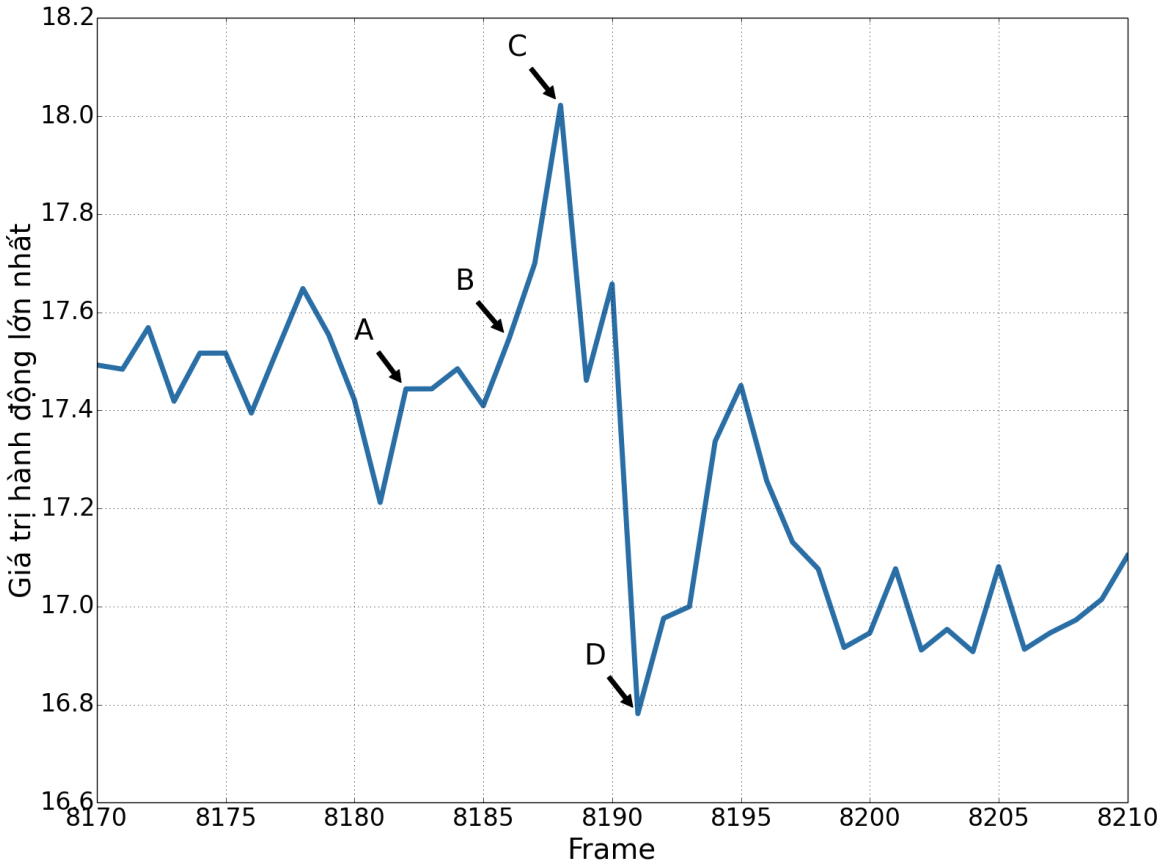
Hình 4.3 và đồ thị 4.4 thể hiện các frame ảnh và giá trị của các trạng thái tương ứng. Quan sát giá trị tương đối của các trạng thái liên tiếp nhau trong một lần chơi, ta có thể thấy mạng nơ-ron học được một hàm giá trị phức tạp và có ý nghĩa cao. Nhờ hàm giá trị được xấp xỉ tốt, chính sách chơi game mới lựa chọn hành động hợp lý để nhận được điểm thưởng cao nhất có thể.



(a) (b)

(c) (d)

Hình 4.3: Hình ảnh 4 frame của trò chơi *Seaquest*. Ở frame đầu tiên (hình 4.3a), tàu ngầm do hệ thống điều khiển ở giữa và tàu ngầm địch ở bên phải. Ở frame kế tiếp (hình 4.3b), hệ thống thực hiện hành động phóng ngư lôi. Tiếp đó (hình 4.3c), ngư lôi trúng tàu ngầm địch. Cuối cùng (hình 4.3d), tàu ngầm địch bị phá huỷ cho điểm thưởng. Các frame này ứng với các giá trị trong đồ thị ở hình 4.4.



Hình 4.4: Đồ thị thể hiện giá trị hành động lớn nhất tại một số trạng thái; các trạng thái này ứng với các frame trò chơi *Seaquest*. Tại frame được đánh dấu A, giá trị hành động lớn nhất đang ở mức trung bình ứng với việc tàu ngầm của ta chưa thực hiện hành động gì ảnh hưởng đến điểm số. Tại frame B, tàu ngầm của ta bắn ngư lôi về phía tàu ngầm địch. Mạng nơ-ron khi nhận được frame ảnh này "hiểu" được rằng nếu ngư lôi chạm vào địch thì ta sẽ nhận được điểm thưởng. Vì vậy, đồ thị giá trị bắt đầu tăng lên đến "frame" C. Tại frame C, ngư lôi trúng tàu ngầm địch nên việc nhận được điểm là chắc chắn; hàm giá trị biết được điều này nên đồ thị giá trị đạt đỉnh điểm tại đây. Sau khi nhận được điểm thưởng và tàu ngầm địch bị phá huỷ, mạng nơ-ron nhận thấy rằng xung quanh không có yếu tố nào có thể giúp tăng điểm số nên đánh giá frame D có giá trị thấp.

Chương 5

Kết luận và hướng phát triển

5.1 Kết luận

Khoá luận này thực hiện cài đặt lại hướng tiếp cận kết hợp học tăng cường với học sâu trong bài toán tự động chơi game được đề xuất bởi [13]. Hướng tiếp cận này sử dụng mạng nơ-ron tích chập của phương pháp học sâu cùng với thuật toán "Q-learning" của phương pháp học tăng cường để tạo thành một mô hình "end-to-end". Việc thực nghiệm lại cho thấy, hướng tiếp cận này giúp xây dựng một mô hình có khả năng tự động học và chơi nhiều game khác nhau của hệ máy Atari. Kết quả thực nghiệm không hoàn toàn tương đồng với kết quả được báo cáo trong [13] dù cấu trúc mô hình và siêu tham số đều giống nhau (có một game cho kết quả tốt hơn [13] và hai game cho kết quả thấp hơn [13]). Lý do chính của vấn đề này là do cách cài đặt khác nhau và hệ thống giả lập game cũng khác nhau. Ngoài ra, bản chất của bài toán tự động chơi game là giá trị điểm thưởng của nhiều lần chơi có phương sai rất lớn nên việc so sánh phải chấp nhận một mức độ sai số nhất định.

Tiếp đó, chúng em tìm hiểu vấn đề đánh giá quá cao của thuật toán "Q-learning". Vấn đề này được đề cập bởi [8, 19]. Theo các nghiên cứu này, vấn đề đánh giá quá cao ảnh hưởng lớn đến kết quả bài toán tự động chơi game. Do đó, nhóm đã cài đặt lại và thực nghiệm thuật toán "Double Q-learning" được đề xuất bởi [19]. Đây là một cải tiến đơn giản của "Q-learning" giúp hạn chế vấn đề đánh giá quá cao. Kết quả thực nghiệm cho thấy, thuật toán "Double Q-learning" giải quyết vấn đề này và giúp cải thiện rất nhiều cho hệ thống tự

động chơi game. Tương tự như với "Q-learning", kết quả thực nghiệm của nhóm cũng không hoàn toàn tương đồng với [19] vì cách cài đặt và hệ thống giả lập game khác nhau.

5.2 Hướng phát triển

Các kết quả thực nghiệm cho thấy hệ thống có khả năng chơi và đạt được điểm số cao nhưng việc thực nghiệm chỉ mới trên ba game khác nhau; số lượng game thực nghiệm còn nhỏ nên vẫn chưa thể rút ra kết luận chặt chẽ cho hướng tiếp cận này. Lý do nhóm không thực nghiệm trên nhiều game hơn là vì thời gian huấn luyện quá lâu (hơn hai ngày huấn luyện liên tục cho mỗi game và mỗi thuật toán). Đây cũng là một nhược điểm lớn của hướng tiếp cận kết hợp học tăng cường với học sâu. Những hướng phát triển tiếp theo của đề tài có thể tập trung vào việc tăng tốc độ huấn luyện của mô hình. Cụ thể hơn, nhóm xác định có hai hướng chính để cải tiến hướng tiếp cận này:

- **Tăng tốc việc huấn luyện mạng nơ-ron**: sử dụng các kỹ thuật tiên tiến của học sâu (như "Weight normalization", "Dropout", ...) để mạng nơ-ron xấp xỉ hàm đích nhanh hơn và chính xác hơn. Thực tế cho thấy, phần lớn thời gian huấn luyện là để cập nhật trọng số của mạng nơ-ron. Khi mạng nơ-ron xấp xỉ hàm càng nhanh thì tốc độ chung của toàn mô hình càng được cải thiện. Còn nếu mạng nơ-ron xấp xỉ hàm chính xác hơn thì ta có thể giảm số "frame" ảnh huấn luyện. Khi đó, ta có thể thử nhiều game hơn trong cùng một khoảng thời gian.

- **Cải thiện thuật toán học tăng cường**: thử nghiệm các thuật toán khác của học tăng cường (như thuật toán "Expected Sarsa", "Policy Gradient", ...) để tìm chính sách tốt hơn. Các thuật toán khác của học tăng cường có thể tìm được những chính sách tốt hơn thuật toán "Q-learning" trong bài toán tự động chơi game. Ngoài ra, việc thử nghiệm các thuật toán khác cũng giúp ta hiểu hơn về ảnh hưởng của mạng nơ-ron tích chập đến các thuật toán học tăng cường.

TÀI LIỆU THAM KHẢO

[1] M. Bellemare, J. Veness, and M. Bowling, "Bayesian learning of recursively factored environments," in *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, 2013, pp. 1211–1219. 4

[2] M. Bellemare, J. Veness, and E. Talvitie, "Skip context tree switching," in *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, T. Jebara and E. P. Xing, Eds. JMLR Workshop and Conference Proceedings, 2014, pp. 1458–1466. [Online]. Available: http://jmlr.org/proceedings/papers/v32/bellemare14.pdf 4

[3] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *Journal of Artificial Intelligence Research*, 2012. 60

[4] I. G. Y. Bengio and A. Courville, "Deep learning," 2016, book in preparation for MIT Press. [Online]. Available: http://www.deeplearningbook.org 38

[5] A. Defazio and T. Graepel, "A comparison of learning algorithms on the arcade learning environment," *arXiv preprint arXiv:1410.8620*, 2014. 3

[6] M. Genesereth, N. Love, and B. Pell, "General game playing: Overview of the aaai competition," *AI magazine*, vol. 26, no. 2, p. 62, 2005. 4

[7] G. J. Gordon, "Stable function approximation in dynamic programming," in *Proceedings of the twelfth international conference on machine learning*, 1995, pp. 261–268. 20

[8] H. V. Hasselt, "Double q-learning," in *Advances in Neural Information Processing Systems*, 2010, pp. 2613–2621. 56, 66, 71

[9] M. Hausknecht, J. Lehman, R. Miikkulainen, and P. Stone, "A neuroevolution approach to general atari game playing," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 6, no. 4, pp. 355–366, 2014. 4

[10] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015. 36, 39

[11] L.-J. Lin, "Reinforcement learning for robots using neural networks," DTIC Document, Tech. Rep., 1993. 53

[12] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013. 53, 54, 62, 63

[13] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, pp. 529–533, 2015. 5, 44, 50, 52, 53, 54, 62, 63, 64, 66, 71

[14] S. Risi and J. Togelius, "Neuroevolution in games: State of the art and open challenges," 2014. 5

[15] R. D. Smallwood and E. J. Sondik, "The optimal control of partially observable markov processes over a finite horizon," *Operations Research*, vol. 21, no. 5, pp. 1071–1088, 1973. 22

[16] R. S. Sutton and A. G. Barto, *Introduction to reinforcement learning*. MIT Press Cambridge, 1998, vol. 135. 1, 11, 13, 16, 22, 30, 33, 36, 51, 55

[17] Theano Development Team, "Theano: A Python framework for fast computation of mathematical expressions," *arXiv e-prints*, vol. abs/1605.02688, May 2016. [Online]. Available: http://arxiv.org/abs/1605.02688 62

[18] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural Networks for Machine Learning*, vol. 4, p. 2, 2012. 62

[19] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," *arXiv preprint arXiv:1509.06461*, 2015. 55, 56, 66, 71, 72