MỤC LỤC

DANH MỤC HÌNH ẢNH

DANH MỤC BẢNG

Chương 1

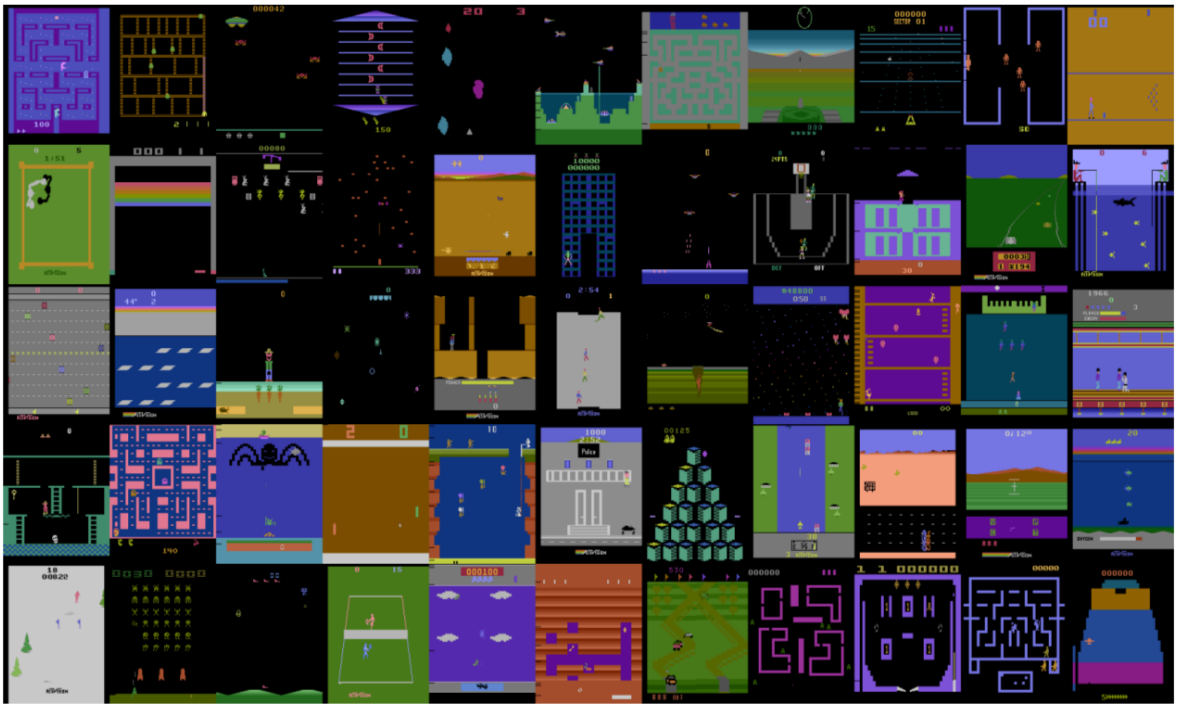
Giới thiệu

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 (Artificial Intelligence). Những đóng góp nổi bật của phương pháp này 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.

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. Một điểm quan trọng của học tăng cường là nó 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 vào của bài toán học tăng cường giúp cho phương pháp này 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 thường được gán nhán thủ công sẵn và việc chủ yếu của hệ thống là làm sao dự đoán chính xác các nhãn đó với 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 nghìn hành động. Ngoài ra, giá trị điểm thưởng thường là 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. Đây 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 luận văn 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 hiện tại. Từ hình ảnh thô này, hệ thống cần tìm cách chơi sao cho điểm số cuối màn chơi (Episode) là lớn nhất có thể. Hệ thống hoàn toàn không biết quy luật của game trước khi bắt đầu quá trình học mà phải tự tìm hiểu quy luật và chiến thuật chơi tối ưu. Lý do luận văn 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.



Hình 1.1: Hình ảnh các game trên hệ máy Atari

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

- Hệ thống không được cung cấp 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 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 có hình ảnh, nội dung rất khác nhau. 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.
- 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.

[TODO: Thêm hướng tiếp cận liên quan + các thực nghiệm + Reference]

Trong những năm gần đây, học sâu đạt đươc nhiều bước đột phá trong nhiều lĩnh vực như Thị giác máy tính (Computer Vision), Nhận diện giọng nói (Speech Recognition), ... Việc kết hợp giữa học sâu và học tăng cường đã dẫn đến một hướng tiếp cận mới cho bài toán tự động chơi game: học tăng cường sâu (Deep reinforcement learning) [3]. 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 luận văn 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, luận văn thử nghiệm kỹ thuật học chuyển tiếp (Transfer learning) nhằm giảm thời gian huấn luyện cho nhiều game.

Chương 2

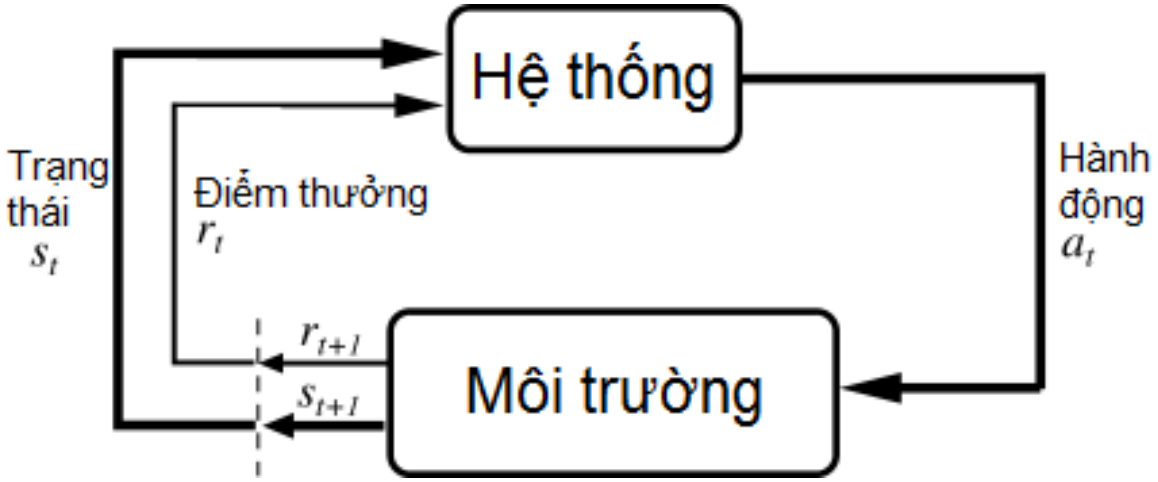
Kiến thức nền tảng

Trong chương này sẽ trình bày những kiến thức nền tảng của học tăng cường. Trong phần đầu tiên chúng em sẽ trình bày định nghĩa của các thành phần cơ bản trong học tăng cường. Tiếp đó sẽ đề cập đến mô hình Markov Decision Processes được áp dụng trong việc đánh giá lý thuyết một số thành phần của bài toán học tăng cường. Cùng với đó sẽ trình bày qui trình tổng quát để đánh giá và cải thiện chính sách trong bài toán. Cuối cùng chúng em sẽ trình bày một số phương pháp phổ biến thường đượcAgent áp dụng để đánh giá cũng như cải thiện giúp hệ thông có cách giải tốt hơn cho bài toán trên.

2.1 Các thành phần cơ bản của học tăng cường

2.1.1 Agent và môi trường

Trong học tăng cường, đối tượng học và đưa ra quyết định được gọi chung là *agent*. Nó tương tác trực tiếp tới một đối tượng được gọi là *môi trường*. Sự tương tác này được diễn ra liên tục. Agent lựa chọn hành động dựa trên những gì nó nhận được từ môi trường. Môi trường cung cấp giá trị điểm thưởng (reward) cho hành động vừa được thực hiện và những quan sát (observation) tiếp theo cho agent. Từ những quan sát này, agent có thể xây dựng ra các *trạng thái* (state) dựa vào đó để ra quyết định chọn hành động với mục tiêu cố gắng đạt được nhiều điểm thưởng nhất.



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

Cụ thể hơn, agent và môi trường tương tác theo một chuỗi tuần tự các time-steps, $t = 0, 1, 2, ...$ Tại mỗi time step t, agent nhận những mô tả trạng thái của môi trường, $S_t \in \mathcal{S}$, với $\mathcal{S}$ là tập các trạng thái có thể có. Dựa vào những mô tả trạng thái nhận được, agent chọn một hành động, $A_t \in (S_t)$, trong đó (S_t) là tập các hành động có thể thực hiện tại trạng thái S_t. Tại time step sau đó, agent nhận được giá trị điểm thưởng, $R_{t+1} \in \mathbb{R}$, cùng với trạng thái tiếp theo S_{t+1} Quá trình tương tác giữa agent và môi trường được mô tả trong hình 2.1

Các thành phần của agent gồm có:

- **Chính sách**. Chính sách, π, xác định khả năng chọn một hành động khi agent nhận được một trạng thái s. Chính xác tại time step t được xác định $\pi_t(a \mid s) = \mathbb{P}[A_t = a \mid S_t = s]$. Để đạt được mục tiêu được nhiều điểm thưởng nhất, agent cần có một*chính sách* chọn lựa hành động phù hợp mỗi khi gặp một trạng thái. Những phương pháp học tăng cường thường tập trung thay đổi các chính sách của agent sao cho đạt được kết quả tốt trong thực nghiệm.

- **Hàm giá trị**. Hầu hết các thuật toán học tăng cường đầu tập trung đánh giá những *hàm giá trị*, các hàm này đánh giá một trạng thái hoặc hành động là tốt như thế nào cho agent thông qua việc ước lượng điểm thưởng nhận được ở tương lai. Thông thường, giá trị của một trạng thái s, dưới một chính sách π được ký hiệu $v_\pi(s)$ là lượng điểm thưởng kỳ vọng nhận

được bắt đầu từ trạng thái s về sau.

- **Mô hình**. Agent xây dựng mô hình cho riêng mình để mô phỏng môi trường và dự doán các thông tin của môi trường trong tương lai.

2.1.2 Returns

Return G_t xác định lượng điểm thưởng mà agent nhận được kể từ thời điểm time step t đến tương lai. Return thường được xác định bằng nhiều hàm khác nhau, trong đó hàm đơn giản nhất xác định return bằng tổng các điểm thưởng có thể nhận được. Nó có dạng như sau:

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

ở đây T là time step cuối cùng.

Mặt khác, return cũng có thể được xác định bằng tổng điểm thưởng đã bị discount qua từng time step. Nó được định nghĩa như sau:

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

Trong đó γ là một hệ số với giá trị $0 \leqslant \gamma \leqslant 1$. γ cũng được gọi là tỉ lệ discount. Tỉ lệ này xác định độ tin tưởng của agent vào giá trị điểm thưởng ở tương lai. Khi $\gamma \to 1$, agent có su hướng quan tâm đến giá trị điểm thưởng tương lai càng nhiều. Đặc biệt với $\gamma = 0$, khi đó agent chỉ quan tâm giá trị điểm thưởng ở hiện tại mà bỏ qua những giá trị điểm thưởng ở tương lai.

Trong thực nghiệm, việc tương tác giữa agent và môi trường có thể được phân chia thành những chuỗi con. Chúng được gọi là những *episode*. [TODO]

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 có hình thức. Cụ thể, MDP là một bộ bao gồm 5 thành phần $< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma >$ trong đó:

- $\mathcal{S}$: tập trạng thái hữu hạn có thể có của môi trường.
- $\mathcal{A}$: tập những hành động hữu hạn mà hệ thống có thể thực hiện để tương tác với môi trường.
- γ: Hệ số 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 theo từng bộ (trạng thái, hành động). $\mathcal{R}^a_s$ là kỳ vọng giá trị điểm thưởng nhận được khi hệ thống thực hiện hành động a ở trạng thái s.

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

Ví dụ: Mô hình MDP trong robot thu gom Công việc của robot này là thu lượm những lon soda đã được uống hết trong văn phòng. Nó 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 từ đó điểu 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? Robot có thể có ba quyết định (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 trạng thái của 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 của nó không giảm. Mỗi khi dung lượng pin của robot ở mức thấp nó 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 nó 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 của nó 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 $\mathcal{S} = \{\text{cao}, \text{thấp}\}$. 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.

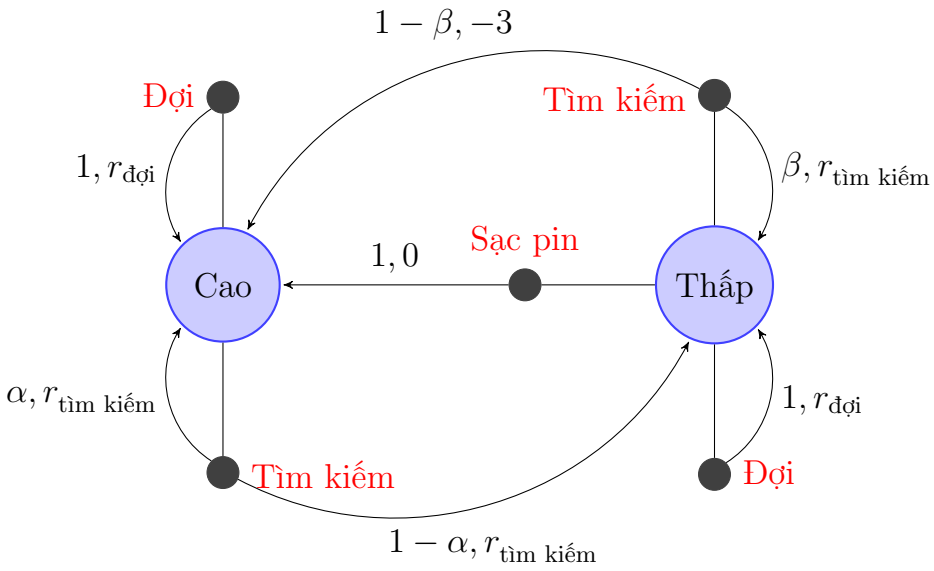
$$\mathcal{A}(\text{cao}) = \{\text{tìm kiếm}, \text{đợi}\}$$

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

Khi mức năng lượng pin ở mức cao, việc robot thực hiện tìm kiếm sẽ có xác suất α năng lượng pin vẫn ở mức cao, và $1 - \alpha$ năng lượng của pi 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, trường hợp này xảy ra khi dung lượng pin cạn kiệt và cần ai đó mang nó đến chỗ sạc cho đến khi đạt mức năng lượng cao. Ngoài ra, mỗi lần robot thu gom được một lon soda nó sẽ nhận được $+1$ điểm thưởng và sẽ bị -3 điểm thưởng mỗi khi nó 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 robot thu gom.

2.2.2 Chính sách và hàm giá trị

Một chính sách π xác định xác suất mà hệ thông thực hiện hành động a khi nó trong trạng thái s được ký hiệu $\pi(a \mid s)$. Có thể nói chính sách như "bộ não"



Hình 2.2: Đồ thị minh họa 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 có thể có mà hệ thống có thể nhận được, nó được ký hiệu một vòng tròn lớn với tên của trạng thái bên trong. Node hành động tường 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 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. Xác suất chuyển tới trạng thái s' khi thực hiện hành động a ở trạng thái s $p(s' \mid s, a)$, và giá trị điểm thưởng kỳ vọng nhận được trong trường hợp này $r(s, a, s')$ tương ứng với ký hiệu trên mũi tên. Ví dụ: khi mức năng lượng pin đăng ở trạng thái *thấp*, hệ thống quyết định thực hiện hành động *sạc pin* khi đó trạng thái tiếp theo mà hệ thống nhận được sẽ là mức năng lượng pin ở trạng thái *cao* và xác suất chuyển tới trạng thái *cao* $p(\text{cao} \mid \text{thấp}, \text{sạc pin})$ là 1 và giá trị kỳ vọng điểm thưởng tương ứng $r(\text{thấp}, \text{sạc pin}, \text{cao})$ là 0.

của hệ thống, nó quyết định cách thức mà hệ thống hành động trong những trạng thái cụ thể do đó một chính sách tốt cũng làm cho khả năng hệ thống ra quyết định trở nên tốt hơn.

Hàm giá trị cho biết những trạng thái hoặc những cặp hành động và trạng thái tốt như thế nào cho hệ thông khi nó trong những trạng thái hoặc thực hiện những cặp hành động và trạng thái đó. Khái niệm tốt ở đây nghĩa là giá trị điểm thưởng kỳ vọng mà hệ thông có thể nhận được ở tương lai. Hầu hết các thuật toán trong học tăng cường đều tập trung vào việc đánh giá những hàm giá trị qua đó cải thiện chính sách trở nên tốt hơn. Điểm thưởng mà hệ thống có thể nhận được trong tương lai phụ thuộc vào những hành động mà nó thực hiện. 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 giá trị kỳ vọng của return mà hệ thống nhận được bắt đầu từ trạng thái s theo chính sách π sau đó. Với mô hình MDP, $v_\pi(s)$ được định nghĩa như sau:

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

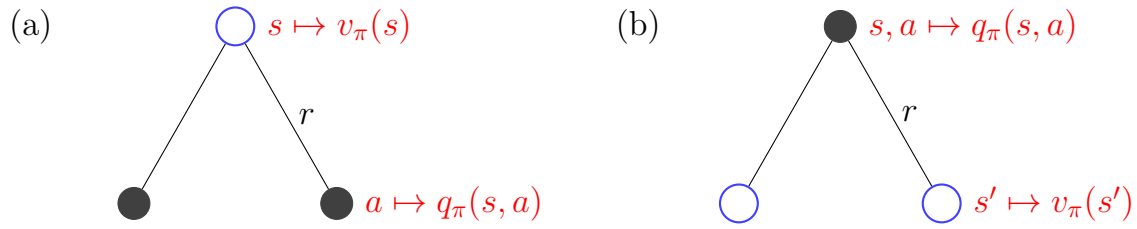
v_π được gọi là hàm giá trị trạng thái dưới chính sách π.

Tương tự, chúng ta định nghĩa giá trị của việc thực hiện hành động a trong trạng thái s dưới chính sách π, được ký hiệu $q_\pi(s, a)$, là giá trị kỳ vọng của return mà hệ thống nhận được bắt đầu từ việc thực hiện hành động a trong trạng thái s theo chính sách π

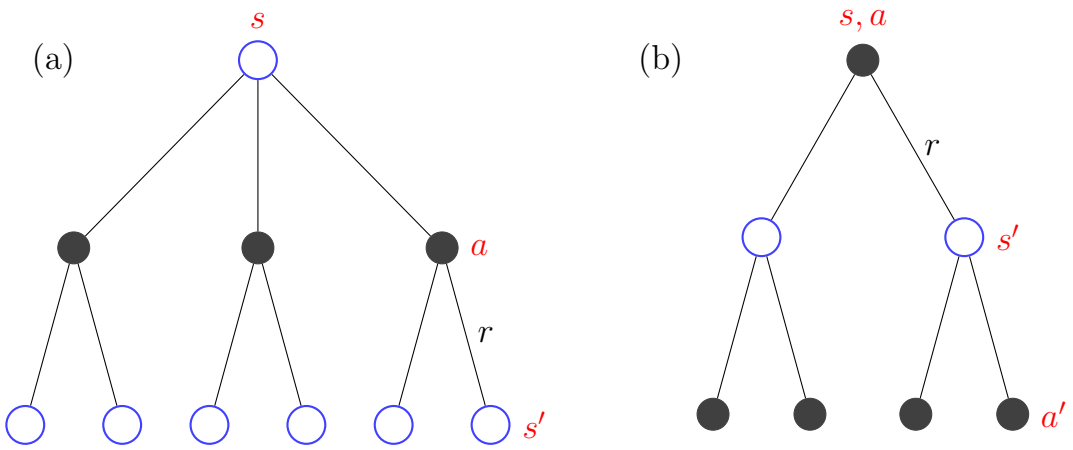
$$q_\pi(s, a) = \mathbb{E}_\pi \left[G_t \mid S_t = s, A_t = a \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s, A_t = a \right] \tag{2.6}$$

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

Hình 2.3 minh họa quan hệ giữa hàm giá trị trạng thái và hàm giá trị hành động, khi có được hàm giá trị này ta có thể có được hàm giá trị còn lại. Phương trình 2.7 xác hàm giá trị của một trạng thái bằng giá trị kỳ vọng giá trị của các hành động thực hiện tại trạng thái đó. Hình 2.3a minh họa quan hệ giữa giá trị của một trạng thái s và giá trị của các hành động thực hiện tại trạng



Hình 2.3: Đồ thị minh họa quan hệ giữa hàm giá trị trạng thái và hàm giá trị hành động



Hình 2.4: Đồ thị minh họa cho (a) v_π và (b) q_π

thái đó. Hình 2.3b cho thấy từ việc thực hiện hành động a tại trạng thái s, môi trường có thể trả ra nhiều trạng thái tiếp theo s' khác nhau. Do đó giá trị của hành động a ở trạng thái s có thể được xác định bằng tổng giá trị kỳ vọng điểm thưởng nhận được và giá trị kỳ vọng của các trạng thái tiếp theo đó đã được nhân với hệ số γ. Cách xác định này được biểu diễn trong phương trình 2.8.

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

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

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

s, hàm giá trị cho một trạng thái được xác định:

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

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 s bất kỳ và giá trị của những trạng thái tiếp theo đạt được từ trạng thái đó. Ý tưởng nhìn trước một bước, hay nói cách khác đánh giá trạng thái hiện tại bằng cách nhìn trước tất cả những trạng tái tiếp theo có thể đạt được từ trạng thái đó, được minh họa trong hình 2.4a. Từ một trạng thái, môi trường có thể trả ra nhiều điểm thưởng r và trạng thái tiếp theo s' khác nhau. Phương trình 2.9 sẽ trung bình tất cả các trường hợp có thể đó lại theo xác suất mà chúng xuất hiện. Phương trình này cũng cho thấy giá trị của một trạng thái phải bằng tổng giá trị kỳ vọng của những trạng thái tiếp sau đó và giá trị kỳ vọng điểm thưởng nhận được.

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

Phân tích phương trình 2.6 tương tự như đã làm đối với hàm giá trị hành động, ta có được phương trình 2.10. Hình 2.4b minh họa ý tưởng nhìn trước một đước để đánh giá giá trị của một hành động ở trạng thái hiện tại. Từ một hành động a ở trạng thái s, môi trường có thể trả ra nhiều điểm thưởng r và trạng thái s' 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, $q_\pi(a, s)$ cũng được xác định tổng bằng giá trị kỳ vọng điểm thưởng hệ thống nhận được nhận được ngay sau khi thực hiện thực hiện hành động đó và giá trị kỳ vọng của các hành động trong những trạng thái kế tiếp.

2.2.3 Hàm giá trị tối ưu

Để giải quyết những vấn đề trong 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 giá trị kỳ vọng của return theo chính sách π lớn hơn hoặc bằng giá trị đó theo chính sách π'. Hay có thể định nghĩa theo cách khác:

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

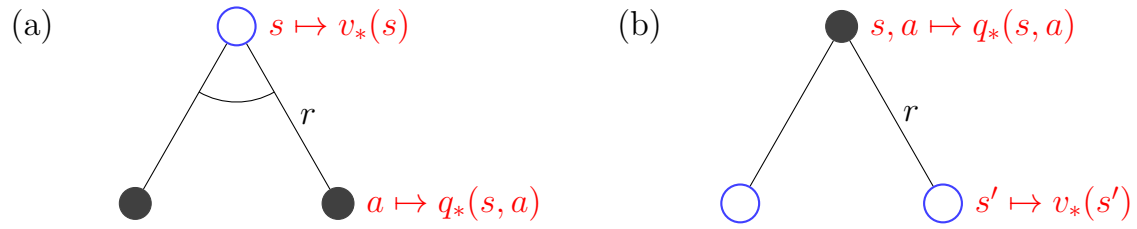
Luôn có ít nhất một chính sách tốt hơn hoặc bằng tất cả các chính sách còn lại [4]. Chúng được gọi chung là *chính sách tối ưu* và được ký hiệu π_*. Những chính sách tối ưu đều cùng có chung một hàm giá trị trạng thái và hàm giá trị hành động. Hai loại hàm giá trị này có thể được gọi chung là *hàm giá trị tối ưu*. Chúng ta cũng có thể gọi tách biệt *hàm giá trị trạng thái tối ưu* đối với hàm giá trị trạng thái và *hàm giá trị hành động tối ưu* đối với hàm giá trị hành động. Phương trình 2.12 và 2.13 định nghĩa hình thức cho hai loại hàm này

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

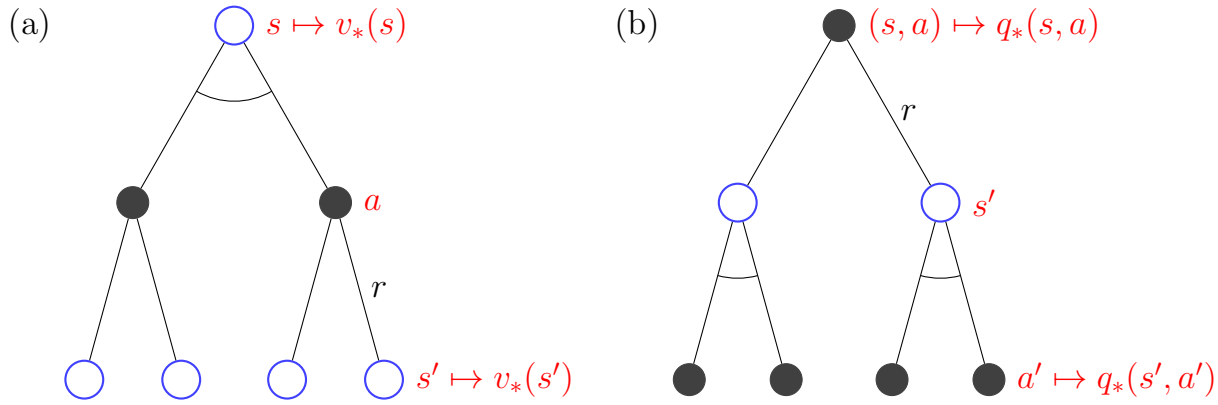
$$q_*(s,a) = \max_\pi q_\pi(s,a), \forall s \in \mathcal{S} \text{ và } \forall a \in \mathcal{A}(s) \tag{2.13}$$

Từ hai phương trình 2.12 và 2.13 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 là giá trị tối ưu cho trạng thái s hoặc cặp trạng thái và hành động (s,a).

Hình 2.5 minh họa quan hệ giữa 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 hàm giá trị hành động tối ưu của các hành động có thể thực hiện tại trạng thái đó. 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 ở trọng thái đó. 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 đạt được từ



Hình 2.5: Đồ thị minh họa quan hệ giữa hàm giá trị trạng thái tối ưu và hàm giá trị hành động tối ưu



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

hành động đó. Phương trình 2.15 xác định giá trị tối ưu của hành động a tại trạng thái s bằng tổng giá trị kỳ vọng điểm thưởng nhận được từ môi trường và giá trị tối ưu kỳ vọng của những trạng thái kế tiếp đã nhân với hệ số γ.

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

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

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

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

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

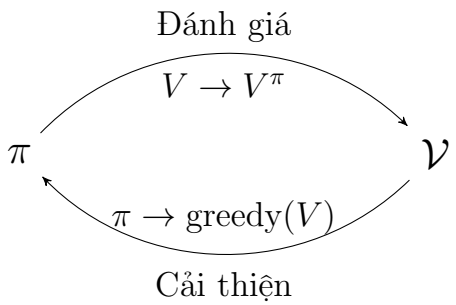
phương trình Bellman trong hàm giá trị trạng thái tối ưu và hàm giá trị hành động tối ưu. Hình 2.6 minh họa ý tưởng nhìn trước một bước của phương trình Bellman trong hàm giá trị tối ưu. Trong đó hình 2.6a minh họa cách thức xác định giá trị tối ưu cho một trạng thái ứng với phương trình 2.16. Hình 2.6b minh họa cách thức xác định giá trị tối ưu của một hành động ở một trạng thái ứng với phương trình 2.17.

2.3 Quy trình 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 đó ta cần có quy trình để thay đổi chính sách hiện tại trở nên tối ưu. Quy trình này được gọi là *quy trình lặp chính sách*. Hình 2.7 minh họa quy trình chung của lặp chính sách. Trong 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 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 trạng thái; việc này có thể thực hiện được do mối quan hệ giữa hai loại hàm.

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



Hình 2.7: Quy trình chung trong lặp chính sách

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.

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

2.3.1 Phương pháp đánh giá chính sách

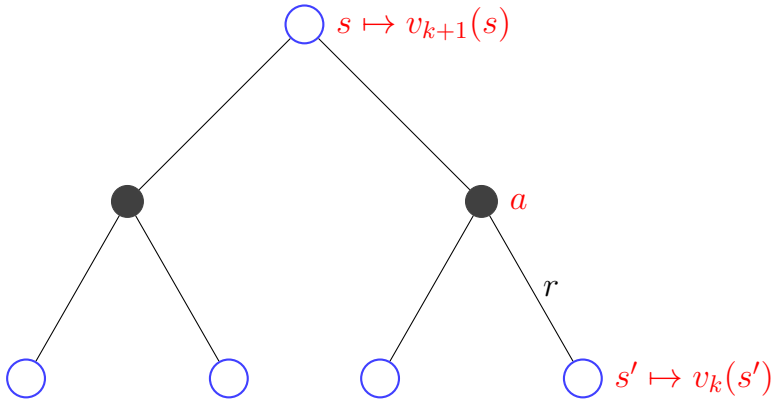
Trong phần này, chúng em sẽ trình bày một số phương pháp được áp dụng để đánh giá chính sách.

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



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

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 xác theo quy hoạch động được trình bày ở thuật toán 2.1

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

[TODO] Ví dụ minh họa TD

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 [2], 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. Ngoài ra, trong nhiều bài toán thực tế thông thường chúng ta không có kiến thức đầy đủ về môi trường như ma trận chuyển trạng thái $\mathcal{P}$, ma trận điểm thưởng $\mathcal{R}$, tập các trạng thái $\mathcal{A}$. Do đó hệ thống phải có khả năng học từ những thông tin mà nó tiếp nhận được qua việc tương tác với môi trường. Các thông tin này thường ở dạng chuỗi (trạng thái, hành động, điểm thưởng) $S_1, A_1, R_2, S_2, A_2, R_3, \ldots, S_T$. Với những đặc điểm đó, quy hoạch động không thể áp dụng để đánh giá chính sách trong các bài toán này.

Thuật toán 2.1 Đánh giá hàm giá trị theo 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)

Monte Carlo (MC)

Tương tự với quy hoạch động, Monte Carlo (MC) xác định hàm giá trị của một chính sách bằng cách cập nhật hàm giá trị khởi tạo qua nhiều vòng lặp. Điểm biệt khác với quy hoạch động của phương pháp MC là nó có thể áp dụng để đánh giá chính sách khi hệ thông không có kiến thức đầy đủ về môi trường. MC dựa trên những thông tin mà hệ thông có được qua việc tương tác với môi trường để xấp xỉ hàm giá trị. Thông thường những thông tin này được chia thành các *episode*. Mỗi episode là một chuỗi bắt đầu từ một trạng thái bất kỳ cho đến khi đạt được một trong những trạng thái kết thúc. Khi đó MC chỉ thực hiện cập nhật hàm giá trị khi kết thúc một episode.

Ý tưởng của MC để đánh giá giá trị của một trạng thái s từ các mẫu thực nghiệm là trung bình những return mà hệ thông nhận được sau khi hệ thống quan sát được trạng thái s. Khi quan sát càng nhiều mẫu thực nghiệm xuất hiện trạng thái s, giá trị trung bình sẽ càng xấp xỉ tốt giá trị thực của trạng thái này theo chính sách π.

Một episode đạt được bằng cách thực hiện theo chính sách π. Giá trị của trạng thái s $v(s)$ được tính dựa trên những episode có trạng thái s xuất hiện. Mỗi lần trạng thái s xuất hiện trong một episode được gọi là một lần quan sát trạng thái đó. Một trạng thái s có thể xuất hiện nhiều lần trong một episode.

Temporal Difference (TD)

Sarsa

2.3.2 Phương pháp cải thiện chính sách

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 sâu với học tăng cường để áp dụng vào bài toán "Tự động chơi game". Hướng tiếp cận mới mẻ này của lĩnh vực học tăng cường này mang tên "Học tăng cường sâu" (Deep reinforcement learning)
Chương này trình bày hai phần:

- *Nguyên nhân cần sử dụng học sâu và kiến thức cơ bản về học sâu*
- *Áp dụng học tăng cường sâu vào bài toán "Tự động chơi game"*

3.1 Học tăng cường kết hợp với học sâu

3.1.1 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 chính minh là luôn hội tụ trong những điều kiện nhất định [4]. 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... thì việc áp dụng các mô hình phổ biến của học sâu vào bài toán tự động chơi game là đầy hứa hẹn.

3.1.2 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 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" (thường được gọi là đặc trưng) 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 "mạnh" (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.

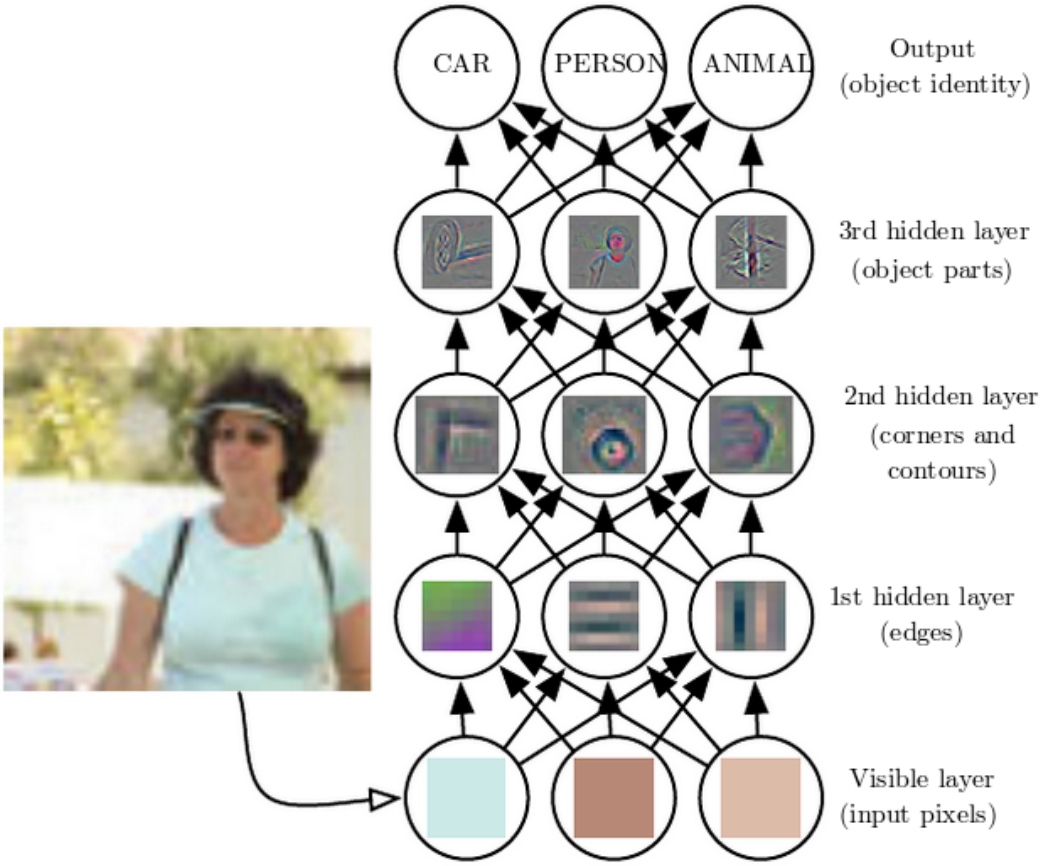
Để giải quyết vấn đề thiết kế đặc trưng, các mô hình máy học thuộc loại "Học biểu diễn" *(Representation learning)* ra đời. Các mô hình này có khả năng tự động học luôn các đặc trưng cần thiết cho quá trình phân tích dữ liệu. Nhờ vậy, các mô hình này có thể áp dụng được dễ dàng hơn vào các bài toán thực 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 của người nói (accent). 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 rất phức tạp.

Một trong những mô hình học sâu phổ biến nhất và cũng cho kết quả rất tốt với dữ liệu ảnh đó là mạng nơ-ron tích chập (Convolutional 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.

3.1.3 Mạng nơ-ron tích chập

Mạng nơ-ron tích chập (Convolutional 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... CNN được thiết kế để tận dụng thông tin về không gian (spatial



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 [1])

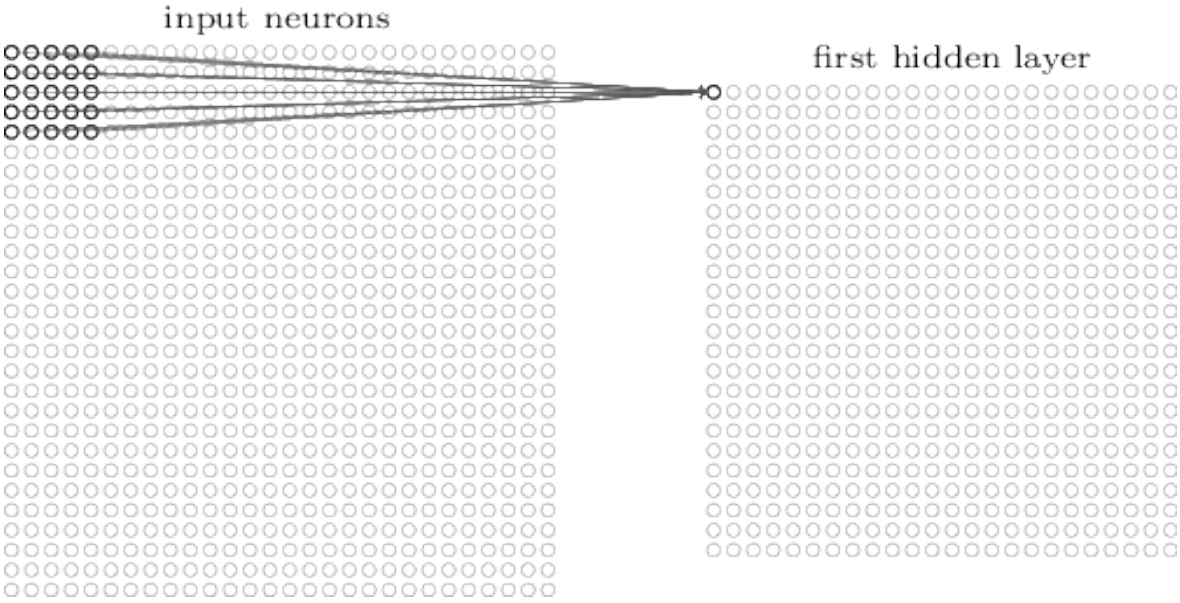
structures) của các loại dữ liệu nêu trên. Ví dụ như trong bài toán nhận diện mặt người trong ảnh, thông tin về vị trí của mắt, mũi, miệng... là rất quan trọng. Nếu ta coi các điểm ảnh đều có ý nghĩa tương tự nhau thì ta đã bỏ quên thông tin về vị trí của những đặc trưng này. Ví dụ như khi ta tìm được hai mắt và miệng trong bức ảnh, ta có thể xác định vị trí tương đối của mũi là nằm ở giữa hai đặc trưng này. 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 nhỏ 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" nhỏ hơn; 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.

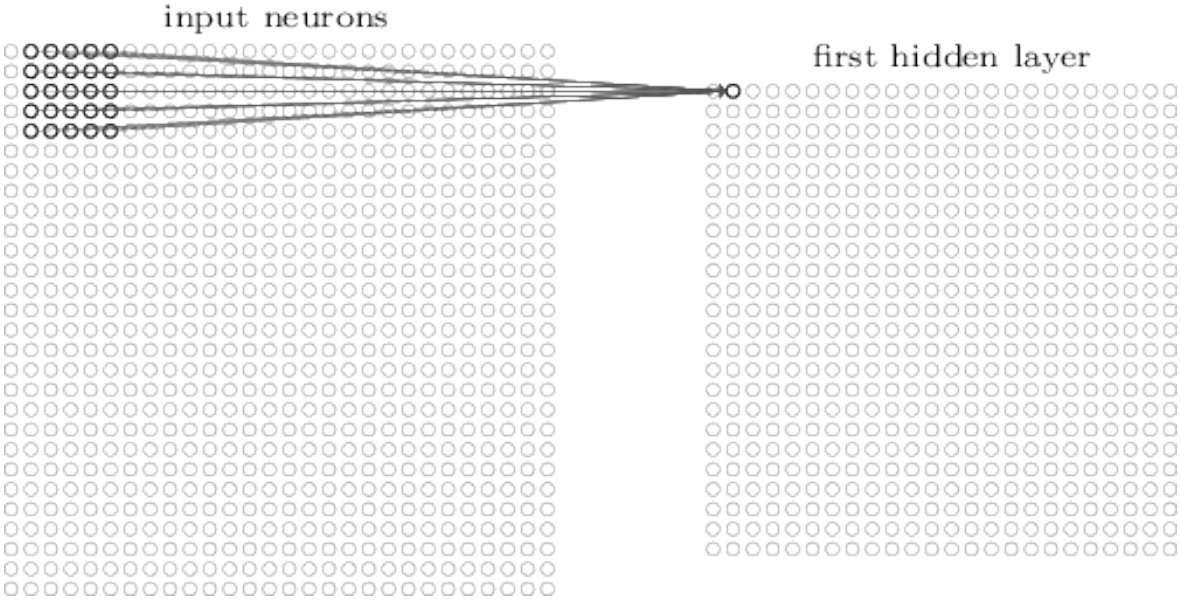
Phép "kiểm tra" đặc trưng của bộ lọc được thực hiện thông qua phép toán **tích chập** (convolution):

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

Trong đó:



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

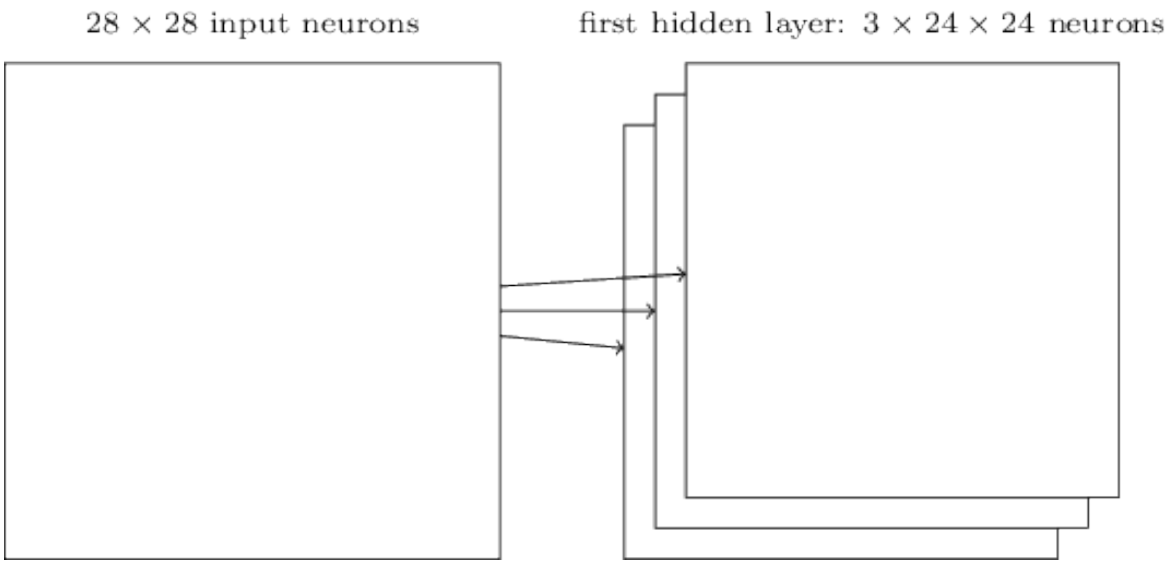


Hình 3.3: Bộ lọc được trượt sang một điểm ảnh về phía bên phải để kiểm tra vùng cục bộ 5×5 bên cạnh. Kết quả của phép "kiểm tra" này được lưu lại thành một "điểm ảnh" bên phải của kết quả của vùng cục bộ trước đó. Thực hiện lần lượt quá trình "trượt" này đến hết ảnh ta sẽ có "bức ảnh kết quả" cuối cùng.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

- $a_{i,j}$ 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 (??)).
- 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 giá trị tổng 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. 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 $28 \times 28 = 764$ như trong hình (??), 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 764 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



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.

trọng số và giá trị "bias" khác nhau. Ta gọi tập hợp các bộ lọc này là một tầng tích chập (convolution layer). Hình (??) 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 phức tạp 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 phức tạp 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.

3.1.4 Sử dụng mạng nơ-ron để xấp xỉ hàm

Việc sử dụng mạng nơ-ron tích chập (hoặc mạng nơ-ron nói chung) 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 tính tổng quát hoá.

Mạng nơ-ron là một mô hình học sâu nên có thể học được những đặc trưng phức tạp 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 cho từng trạng thái của game. Ngoài ra, do qua 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). Nhờ đặc điểm này mà quá trình học tăng cường 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, thuật toán học tăng cường 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. 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 khá dễ dàng.

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" (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 ví 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 sẽ giảm; còn nếu bước đi quá ngắn thì mỗi bước chỉ giảm J một ít, khi đó ta sẽ tốn rất nhiều thời gian để J 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 xấp xỉ tốt

véc-tơ đạo hàm riêng trung bình trên toàn bộ tập huấn luyện. 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:

$$\begin{aligned}
\theta_{t+1} &= \theta_t - \Delta\theta_t && (3.9)\\
&= \theta_t - \alpha\nabla_{\theta_t} J(\theta_t) && (3.10)\\
&= \theta_t - \alpha\nabla_{\theta_t}\left(\frac{1}{B}\sum_{i=1}^{B}(q_\pi(S_i, A_i) - \hat{q}(S_i, A_i; \theta_t))^2\right) && (3.11)\\
&= \theta_t - \alpha\frac{1}{B}\sum_{i=1}^{B}(q_\pi(S_i, A_i) - \hat{q}(S_i, A_i; \theta_t))\nabla_{\theta_t}\hat{q}(S_i, A_i; \theta_t) && (3.12)
\end{aligned}$$

Ở đây:

- $\Delta\theta$ là giá trị cập nhật tại cho 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 khi thực hiện theo chính sách π
- $\hat{q}(S_i, A_i; \theta_t)$ là giá trị **xấp xỉ** của hành động A_i tại trạng thái S_i khi thực hiện theo chính sách π
- $\nabla_{\theta_t}\hat{q}(S_i, A_i; \theta_t)$ là véc-tơ đạo hàm riêng 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:

$$\theta_{t+1} = \theta_t - \alpha(q_\pi(S, A) - \hat{q}(S, A; \theta_t))\nabla_{\theta_t}\hat{q}(S, A; \theta_t) \tag{3.13}$$

3.1.5 Học tăng cường kết hợp với xấp xỉ hàm

Áp dụng thuật toán SGD để tối ưu hàm lỗi J theo công thức (3.12) 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_i 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:

$$q_\pi(s, a) = \mathbb{E}_\pi \left[R_{t+1} + \gamma R_{t+2} + ... + \gamma^{T-t-1}R_T \middle|\ S_t = s, A_t = a\right] \tag{3.14}$$

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 (3.13) được thay bằng:

$$\theta_{t+1} = \theta_t - \alpha(G_t - \hat{q}(S, A; \theta_t))\nabla_{\theta_t}\hat{q}(S, A; \theta_t) \tag{3.15}$$

G_t ở đây là tổng điểm thưởng (tức "Returns") 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 (3.14) 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}, R_{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) càng 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 học chậm 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 (3.13) được thay bằng:

$$\theta_{t+1} = \theta_t - \alpha(R_{t+1} + \gamma \max_a \hat{q}(S_{t+1}, a; \theta) - \hat{q}(S, A; \theta_t))\nabla_{\theta_t} \hat{q}(S, A; \theta_t) \tag{3.16}$$

Mã giả của thuật toán "Q-learning" với xấp xỉ hàm được trình bày ở (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 $\hat{q}(; \theta)$ 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 (3.16) 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

3.2.1 Kỹ thuật làm tăng tính ổn định

3.2.2 Vấn đề "overestimation" của thuật toán Q-learning

Chương 4

Kết quả thực nghiệm

Trong chương này sẽ trình bày chi tiết về cấu trúc mô hình mà chúng em đã thiết lập. Đồng thời đề cập đến những phương pháp để đánh giá mô hình học và những kết quả thực nghiệm đã nhận được. Qua đó so sánh với các phương pháp đã đề xuất trước đây để thấy được tính hiệu quả của mô hình này.

4.1 Giới thiệu Arcade Learning Environment

4.2 Giới thiệu cấu trúc mạng và các siêu tham số đã chọn

4.3 Kết quả thực nghiệm

Chương 5

Kết luận và hướng phát triển

TÀI LIỆU THAM KHẢO

[1] I. G. Y. Bengio and A. Courville, "Deep learning," 2016, book in preparation for MIT Press. [Online]. Available: http://www.deeplearningbook.org 24

[2] G. J. Gordon, "Stable function approximation in dynamic programming," in *Proceedings of the twelfth international conference on machine learning*, 1995, pp. 261–268. 18

[3] 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. 4

[4] R. S. Sutton and A. G. Barto, *Introduction to reinforcement learning*. MIT Press Cambridge, 1998, vol. 135. 14, 22