TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

——————o0o——————

PROJECT I

BÁO CÁO CUỐI KÌ

ĐỀ TÀI: REINFORCEMENT LEARNING

Giáo viên hướng dẫn: **Trịnh Văn Chiến**
Họ và tên: **Nguyễn Khánh Trường Lộc**
Chuyên ngành: **Kỹ thuật máy tính(IT2)**
*MSSV:***20204763**

Mục lục

1 Lời nói đầu

Trí tuệ nhân tạo hay trí tuệ được thể hiện qua máy móc là một thành tựu, một bước phát triển đánh dấu sự phát triển mạnh mẽ của nền văn minh nhân loại. Tiềm năng ứng dụng của nó là rất lớn trong đời sống thường ngày và trong nhiều lĩnh vực khác nhau như kinh tế, quân sự, giải trí...Điển hình cho tiến bộ vượt bậc của trí tuệ nhân tạo là sự ra đời của AlphaGo-phần mềm máy tính đầu tiên đánh bại một kì thủ cờ vây chuyên nghiệp. AlphaGo đã tự chơi với chính nó và cải thiện dần dần các mạng neural được sử dụng để đánh giá bàn cờ, đưa ra chiến lược và quyết định các nước đi. Sự thành công này không thể không nói đến "học tăng cường",phương pháp được sử dụng để cải thiện khả năng của AlphaGo, nó đã cho con người một cái nhìn mới về AI cũng như sự hiệu quả của các phương pháp công nghệ như "học tăng cường".

2 Khái niệm

2.1 Khái niệm "Học tăng cường"(reinforcement learning)

Học tăng cường (Reinforcement learning) là một lĩnh vực học máy liên quan đến cách các tác nhân thông minh(intelligent agents) phải thực hiện hành động trong môi trường để tối đa hóa khái niệm về phần thưởng tích lũy và giảm sự hối tiếc. Học tăng cường là một trong ba mô hình học máy cơ bản, bên cạnh học có giám sát và học không giám sát.

Học tăng cường khác với học có giám sát ở chỗ không cần trình bày các cặp đầu vào/ đầu ra được gắn nhãn và không cần đánh giá đúng sai một cách tường minh.Một đầu vào sẽ có nhiều hơn một đầu ra. Các hành động hiện tại được quan tâm, trọng tâm là sự cân bằng giữa khai thác kiến thức hiện tại và khám phá những gì chưa biết.

Môi trường thường được biểu diễn dưới dạng "quá trình quyết định Markov (MDP)", vì nhiều thuật toán học tăng cường cho ngữ cảnh này sử dụng các kỹ thuật quy hoạch động.Sự khác biệt chính giữa các phương pháp quy hoạch động cổ điển và các thuật toán học tăng cường là phương pháp sau này không giả định kiến thức về một mô hình toán học chính xác của MDP và chúng nhắm tới mục tiêu vào các MDP lớn nơi các phương pháp chính xác trở nên không khả khi.

2.2 Thuật ngữ cơ bản

Tác nhân(agent): được định nghĩa là "anything that can be viewed as perceiving its environment through sensor and acting upon that environment through actuators" (máy quan sát môi trường và đưa ra hành động tương ứng). Nhiệm vụ của tác nhân là đưa ra các quyết định phản ứng với môi trường.

Hành động(Action):Kí hiệu a. Là danh sách các hành động mà tác nhân có thể thực hiện.

Trạng thái(state):Kí hiệu S. Là tình hình hiện tại của tác nhân trong môi trường.

Môi trường(environment): thể hiện của "vấn đề", là mọi thứ xảy ra sau quyết định của tác nhân.

Phần thưởng(reward):Kí hiệu R. Với mỗi hành động được chọn bởi tác nhân, môi trường sẽ đưa ra phần thưởng , phần thưởng có thể tích cực hoặc tiêu cực.Đây là động lực giúp máy thực hiện các hành động, cơ sở để thay đổi chính sách.Trong quá trình khai thác, tác nhân sẽ lựa chọn hành động có phần thưởng cao hơn đã biết trong quá khứ .

Chính sách(policy):Kí hiệu π. Là tác nhân chuẩn bị chiến lược(ra quyết định) để ánh xạ các tình huống thành hành động. Chính sách là cốt lõi của tác nhân trong việc xác định hành vi.Trong một số trường hợp, chính sách có thể là một hàm hoặc bảng tra cứu đơn giản. Một số khác lại có thể là tính toán mở rộng.

2.3 Qúa trình quyết định Markov

Qúa trình quyết định Markov (MDP):là quá trình kiểm soát ngẫu nhiên theo thời gian rời rạc. Nó cung cấp một khung toán học để mô hình hóa việc ra quyết định trong các tình huống mà kết quả là một phần ngẫu nhiên và một phần nằm dưới sự kiểm soát của người ra quyết định. Nó bao gồm hữu hạn các trạng thái (và các hàm giá trị cho các trạng thái đó), một tập hữu hạn các hành động, một quy tắc chuyển trạng thái và hàm phần thưởng.

3 Các hàm

Hàm phản hồi định nghĩa mục tiêu trong bài toán quyết định Markov.Hàm phản hồi định nghĩa sự kiện nào là tốt hay xấu, hàm phản hồi chỉ ra cái gì tốt cho một ý thức tức thì.

Các bài toán có số bước hữu hạn:

$$R_t = r_t + r_{t+1} + ... + r_{t+k-1} \tag{1}$$

Trong đó k là số bước trước khi trạng thái kết thúc.

Các bài toán có só bước vô hạn:

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{2}$$

Hệ số γ ($0 \leq \gamma \leq 1$) xác định mức độ ảnh hưởng của các bước chuyển trạng thái thiếp theo đến giá trị phản hồi tại thời điểm đang xét. Vì không biết được số bước thực hiện trong tương lai, cũng có thể là vô số, γ được thêm vào để đảm bảo tính hội tụ. γ gần bằng 0, giá trị phản hồi của các hành động trong tương lai sẽ được giảm một cách nhanh chóng, điều này có nghĩa là agent sẽ tập trung vào việc đạt được phần thưởng ngay lập tức hơn là đạt được phần thưởng lâu dài, γ gần bằng 1, giá trị phản hồi của các hành động trong tương lai sẽ được giảm một cách chậm chạp, điều này có nghĩa là agent sẽ tập trung vào việc đạt được phần thưởng lâu dài hơn là đạt được phần thưởng ngay lập tức..

Các khung toán học được cung cấp bởi MPD sẽ lựa chọn các hành động để cực đại hóa R:

$$R = r_0 + \gamma r_1 + \gamma^2 r_2 + ... \tag{3}$$

Hàm giá trị theo chính sách kí hiệu V: được tính bằng kì vọng của hàm phản hồi theo thời gian, xác định mức độ thích hợp của chính sách π với tác nhân khi ở trạng thái s. Hàm giá trị sẽ đặc tả cái gì tốt trong suốt một thời gian dài. Trạng thái s_t lựa chọn hành động a_t dựa theo một chính sách π, được kí hiệu $a_t = \pi(s_t)$.

$$V_\pi(s) = E_\pi\{r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | S_t = s\} \tag{4}$$

Hàm giá trị hành động kí hiệu Q: Lợi nhuận kì vọng từ trạng thái S, tuân theo chính sách π và thực hiện hành động a. Hay có thể nói Q là phần thưởng mong đợi nhận được khí thực hiện một hành động.

$$Q_\pi(s, a) = E_\pi\{r_{t+1} + \gamma Q_\pi(S_{t+1}, A_{t+1}) | S_t = s, A_t = a\} \tag{5}$$

Hàm giá trị tối ưu $V^*(s)$ mang lại lợi tức mong đợi nếu bắt đầu ở trạng thái s và luôn hành động theo chính sách tối ưu trong môi trường.

$$V^*(s) = maxE_\pi[R_t | S_t = s] \tag{6}$$

Hàm giá trị hành động tối ưu $Q^*(s, a)$ mang lại lợi nhuận mong đợi nếu bắt đầu ở trạng thái s, thực hiện một hành động tùy ý a và sau đó hành động theo chính sách tối ưu trong môi trường.

$$Q^*(s, a) = maxE_\pi[R_t | S_0 = s, A_0 = a] \tag{7}$$

Hàm lợi thế $A^\pi(s, a)$: đôi khi trong học tăng cường, chúng ta không cần mô tả một hành động tốt như thế nào theo nghĩa tuyệt đối, mà chỉ cần xem nó tốt hơn bao nhiêu so với các hành động khác. Có nghĩa là chúng ta muốn biết lợi thế tương đối của hành động đó. Hàm lợi thế $A^\pi(s, a)$ tương ứng với một chính sách π mô tả mức độ tốt hơn nếu thực hiện một hành động cụ thể a ở trạng thái s, thay vì chọn ngẫu nhiên một hành động theo $\pi(|s)$.

$$A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s) \tag{8}$$

3.1 Mối quan hệ giữa V và Q

Mối quan hệ giữa V và Q theo chính sách ngẫu nhiên π có thể hiểu như là hàm giá trị là tổng xác suất lựa chọn hành động nhân với giá trị hành động của việc thực hiện từng hành động.

Hình 1: Liên hệ giữa V và Q

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

Mối quan hệ giữa Q và V:

Hình 2: Liên hệ giữa Q V

$$Q^{\pi}(s,a) = \sum_{s' \in \mathcal{S}} P(s'|s,a) \left[R(s,a,s') + \gamma V^{\pi}(s') \right]$$

P xác suất đạt đến trạng thái s' từ trạng thái s, R là phần thưởng ngay lập tức, V là giá trị trạng thái của trạng thái tiếp theo s'.

4 Hoạt động

Học tăng cường được mô hình hóa như một quyết định Markov(MPD):

+Một tập hợp các trạng thái môi trường S.

+Một tập hợp các hành động A của tác nhân.

$+P_a(s,s') = Pr(s_{t+1} = s'|s_t = s, a_t = a)$ là xác suất chuyển đổi (tại thời điểm t) từ trạng thái s đến trạng thái s' sau khi chọn hành động a.

$+R_a(s,s')$ là phần thưởng ngay sau khi chuyển trạng thái từ s đến s' sau khi chọn a.

Học tăng cường mục đích để một tác nhân thông minh tìm hiểu các chính sách có thể giúp nó tối đa hóa phần thưởng hoặc các tín hiệu tương tự. Tác nhân tương tác với môi trường theo các bước thời gian rời rạc. Ở mỗi thời điểm t, tác nhân ở trạng thái s_t và nhận được phần thưởng r_t. Sau mỗi a_t được chọn từ một tập hành động, môi trường phản hồi và đem lại cho tác nhân trạng thái mới s_{t+1} và phần thưởng r_{t+1}.Đây là cách tác nhân tìm hiểu chính sách : $\pi : A \times S \to [0,1], \pi(a,s) = Pr(a_t = a|s_t = s)$ tối đa hóa phần thưởng tích lũy dự kiến.

Hình thành một vấn đề như một giả định tác nhân trực tiếp quan sát trạng thái môi trường hiện tại,trường hợp này tác nhân *có đầy đủ khả năng quan sát*. Nếu tác nhân chỉ được truy cập vào một phần các trạng thái hoặc nếu các trạng thái bị nhiễu , tác nhân chỉ có *khả năng quan sát một phần*, vấn đề phải được xây dựng như một quá trình quyết định Markov có thể quan sát được một

phần(POMDP). Không giống như chức năng chính sách MDP ánh xạ các trạng thái cơ bản đến các hành động, chính sách của POMDP là một ánh xạ từ lịch sử quan sát hoặc trạng thái niềm tin đến các hành động.

5 Các thuật toán điển hình

5.1 SARSA

SARSA(trạng thái-hành động-phần thưởng-trạng thái-hành động) là một thuật toán để học chính sách "quy trình quyết định Markov". Thông qua tên thuật toán, có thể thấy chức năng chính để cập nhật giá trị Q phụ thuộc vào trạng thái hiện tại, hành động hiện tại, phần thưởng khi chọn hành động hiện tại, trạng thái sau đó và hành động chọn từ trạng thái mới.

5.1.1 THuật toán

$$Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma Q(S',a) - Q(S,A)] \tag{9}$$

α: Tỷ lệ học tập xác định mức độ thông tin mới thu được sẽ ghi đè thông tin cũ. Hệ số 0 sẽ khiến tác nhân không học được gì, trong khi hệ số 1 sẽ khiến tác nhân chỉ xem xét thông tin gần đây nhất.

Tác nhân SARSA tương tác với môi trường và cập nhật chính sách dựa trên các hành động được thực hiện. Gía trị phần thưởng Q nhận được trong bước thời gian tiếp theo khi thực hiện hành động a ở trạng thái s, cộng với phần thưởng chiết khấu trong tương lai nhận được từ lần quan sát hành động trạng thái tiếp theo.

5.2 Q-learning

Q-learning là một chính sách học tằng cường nhằm tìm ra hành động tốt nhất tiếp theo, với trạng thái hiện tại. Nó chọn hành động này một cách ngẫu nhiên và nhằm mục đích tối đa hóa phần thưởng.Tùy thuộc vào vị trí của tác nhân trong môi trường,nó sẽ quyết định hành động tiếp theo được thực hiện.

5.2.1 Thuật toán

$$Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma max_{\alpha} Q(S',a) - Q(S,A)] \tag{10}$$

6 Học tăng cường có mô hình , không có mô hình và Học tăng cường có chính sách và không có chính sách

6.1 Học tăng cường có mô hình , không có mô hình

Cách môi trường phản ứng với các hành động được gọi là mô hình, tác nhân có thể không biết mô hình hoặc biết mô hình. Khi tác nhân biết mô hình, đây là trường hợp học tăng cường dựa trên mô hình(Model-Based RL). Trường hợp này, chúng ta biết cách môi trường hoạt đông, phương pháp quy hoạch động sẽ giúp chúng ta tìm ra một giải pháp tối ưu. Khi tác nhân không biết mô hình, đây là trường hợp học tăng cường không có mô hình(model-free RL), tác nhân ra quyết định với thông tin không đầy đủ, nó sẽ tìm hiểu môi trường một cách từ từ.Hay nói cách khác Không có mô hình có nghĩa là tác nhân sử dụng các dự đoán về phản ứng mong đợi của môi trường để tiến lên phía trước. Nó không sử dụng hệ thống phần thưởng để học, mà là thử và sai.

Ưu điểm chính của việc có một mô hình là nó cho phép lập kế hoạch mà có sự suy tính trước, tác nhân có thể chắt lọc kết quả từ việc lập kế hoạch trước thành một chính sách đã học.

Nhược điểm của mô hình là phần lớn thế giới thực sẽ không có sẵn một môi trường có mô hình cho tác nhân. Nếu tác nhân muốn sử dụng mô hình, nó phải học dựa hoàn toàn trên kinh nghiệm, điều này lại tạo ra thách thức mới. Thách thức lớn nhất là tác nhân khai thác một cách thiên vị với một mô hình, dẫn đến tác nhân hoạt động tốt trong mô hình đã học, nhưng lại kém tối ưu trong môi trường thực.

6.2 Học tăng cường trong chính sách và ngoài chính sách

Các phương pháp trong chính sách cố gắng đánh giá hoặc cải thiện chính sách được sử dụng để đưa ra quyết định. Ngược lại, các phương pháp ngoài chính sách đánh giá hoặc cải thiện một chính sách khác với chính sách được sử dụng để tạo dữ liệu.
Tác nhân trong chính sách học giá trị dựa trên hành động hiện tại a của nó bắt nguồn từ chính sách hiện tại, tác nhân ngoài chính sách học giá trị đó dựa trên hành động a* thu được từ chính sách khác.
Trong học tăng cường trong chính sách, tác nhân sử dụng chính sách hiện tại

để lựa chọn hành động trong quá trình tương tác với môi trường, và sau đó cập nhật các tham số của chính sách dựa trên dữ liệu thu thập được. Vì vậy, các thuật toán học tăng cường trong chính sách thường có tính khái quát thấp và khó áp dụng vào các vấn đề thực tế.

Học tăng cường ngoài chính sách là khi tác nhân học và tối ưu hóa một chính sách khác với chính sách hiện tại của nó. Trong học tăng cường ngoài chính sách, tác nhân sử dụng dữ liệu thu thập từ một chính sách khác để cập nhật tham số của chính sách hiện tại của nó. Vì vậy, các thuật toán học tăng cường ngoài chính sách thường có tính khái quát cao và có thể áp dụng vào nhiều vấn đề khác nhau.
Khi chúng ta so sánh sự khác biệt giữa Q-learning và SARSA.

Q-learning sử dụng:

$$Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma max_{\alpha} Q(S',a) - Q(S,A)] \tag{11}$$

để cập nhật Q.
SARSA sử dụng:

$$Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma Q(S',a) - Q(S,A)] \tag{12}$$

để cập nhật Q.
Với chính sách cập nhật là cách tác nhân tìm hiểu chính sách tối ưu và chính sách hành vi là cách chúng ta hành sử.

Trong Q-learning , tác nhân học chính sách tối ưu bằng cách sử dụng chính sách tham lam tuyệt đối và hành sử bằng cách sử dụng các chính sách khác. Vì chính sách cập nhật khác chính sách hành vi, nên Q-learning nằm ngoài chính sách.Nói cách khác, nó ước tính phần thưởng cho các hành động trong tương lai và thêm một giá trị vào trạng thái mới mà không thực sự tuân theo bất kỳ chính sách tham lam nào.

Trong SARSA, tác nhân học chính sách tối ưu và hành sử bằng cách sử dụng cùng một chính sách.Vì cách cập nhật giống với chính sách hành vi nên SARSA nằm trong chính sách.

Về cơ bản, Sarsa tìm hiểu về một chính sách đôi khi thực hiện các hành động

tối ưu và đôi khi khám phá các hành động khác, trong khi Q-learning tìm hiểu về chính sách không khám phá và chỉ thực hiện các hành động tối ưu.

7 Thăm dò và khai thác

Càng nhiều kinh nghiện càng tốt vì ước lượng của các hàm sẽ chính xác. Tuy nhiên, có một vấn đề. Nếu thuật toán để cải thiện chính sách luôn cập nhật chính sách một cách tham lam, nghĩa là nó chỉ thực hiện các hành động dẫn đến phần thưởng ngay lập tức, thì các hành động và trạng thái không nằm trên con đường tham lam sẽ không được lấy mẫu đầy đủ và các phần thưởng tiềm năng tốt hơn sẽ bị ẩn khỏi quá trình học.

Về cơ bản , chúng ta buộc phải đưa ra lựa chọn giữa việc đưa ra quyết định tốt nhất với thông tin hiện tại hoặc bắt đầu khám phá và tìm thêm thông tin. Điều này còn được gọi là thế tiến thoái lưỡng nan giữa thăm dò và khai thác.

Chúng ta phải tìm cái gì đó giống như nền tảng trung gian giữa những thứ đó. Khám phá toàn diện có nghĩa là chúng ta sẽ cần nhiều thời gian để thu thập thông tin cần thiết và khai thác toàn diện sẽ khiến tác nhân bị mắc kẹt trong phần thưởng cục bộ tối đa. Có hai cách tiếp cận để đảm bảo tất cả các hành động được lấy mẫu đầy đủ đó là trong chính sách và ngoài chính sách.

8 Ứng dụng học tăng cường

Học tăng cường được nghiên cứu trong nhiều ngành, như lý thuyết trờ chơi, lý thuyết điều khiển, lý thuyết thông tin, tối ưu hóa dựa trên mô phỏng, trí thông minh bầy đàn...

Học tăng cường được ứng dụng rộng rãi, trong nhiều lĩnh vực đời sống . Bất kỳ vấn đề nào trong thế giới thực một tác nhân phải tương tác với một môi trường mà không rõ cách đáp ứng lại của môi trường đó với một mục tiêu cụ thể là một ứng dụng tiềm năng của học tăng cường.

-Robot: Robot có hành vi được lập trình sẵn rất hữu ích trong các môi trường có cấu trúc như dây chuyền lắp ráp, nơi nhiệm vụ có tính lặp đi lặp lại.Trong thế giới thực , nơi phản ứng của môi trường với hành vi của robot là không

chắc chắn, việc lập trình trước các hành động chính xác là không thể. Trong tình huống như vậy, học tăng cường giúp chế tạo một robot đa năng. Nó đã áp dụng thành công để lập kế hoạch đường đi cho robot, trong đó robot phải tìm đường đi ngắn, trơn tru và có thể điều hướng giữa hai vị trí, không có va chạm và tương thích với động lực học của robot:

-AlphaGo: Cờ vây, một trong những bộ môn trò chơi chiến lược phức tạp nhất với $10^2 70$ tổ hợp bàn cờ có thể xảy ra. Vào năm 2016, phần mềm AlphaGo (phần mềm chơi cờ vây sử dụng học tăng cường) đã đánh bại một trong những kì thủ cơ vây xuất sắc nhất. Giống như người chơi với người, nó đã tự học được kinh nghiệm, bằng cách tự chơi với chính mình và học kinh nghiệm từ đó.
-Lái xe tự động: Một hệ thống lái xe tự động phải thực hiện nhiều nhiệm vụ nhận thức và lập kế hoạch trong một môi trường không chắc chắn. Một số nhiệm vụ cụ thể mà học tăng cường có thể ứng dụng bao gôm lập kế hoạch cho đường đi và dự đoán chuyển động. Lập kế hoạch lộ trình cho phương tiện yêu cầu một số chính sách cấp thấp và cấp cao để đưa ra quyết định về các quy mô không gian khác nhau và thời gian khác nhau. Dự đoán chuyển động là nhiệm vụ dự đoán chuyển động của người đi bộ và các phương tiện khác,để hiểu tình huống có thể phát triển như thế nào dựa trên trạng thái hiện tại của môi trường.

-Cùng nhiều ứng dụng khác trong các lĩnh vực khác nhau như: Học tăng cường trong xử lý ngôn ngữ tự nhiên, y tế, tiếp thị và quảng cáo, tự động hóa, giao dịch tài chính

BÀI TOÁN CARTPOLE

9 Giới thiệu BÀI TOÁN CARTPOLE

Ý tưởng của CartPole là có một cái cột đứng trên đầu một chiếc xe đẩy. Mục tiêu là cân bằng cột này bằng cách di chuyển xe đẩy từ bên này sang bên kia để giữ cho cây gậy cân bằng thẳng đứng.

Mục tiêu là cân bằng gậy trong 500 khung hình và thất bại khi cột nghiêng hơn 15 độ so với phương thẳng đứng hoặc xe đẩy di chuyển hơn 2.4 đơn vị từ vị trí chính giữa.

Với mọi khung hình mà cột "cân bằng" (dưới 15 độ so với phương thẳng đứng), ta nhận được 1 điểm và mục tiêu là 500 điểm. CartPole là một trong

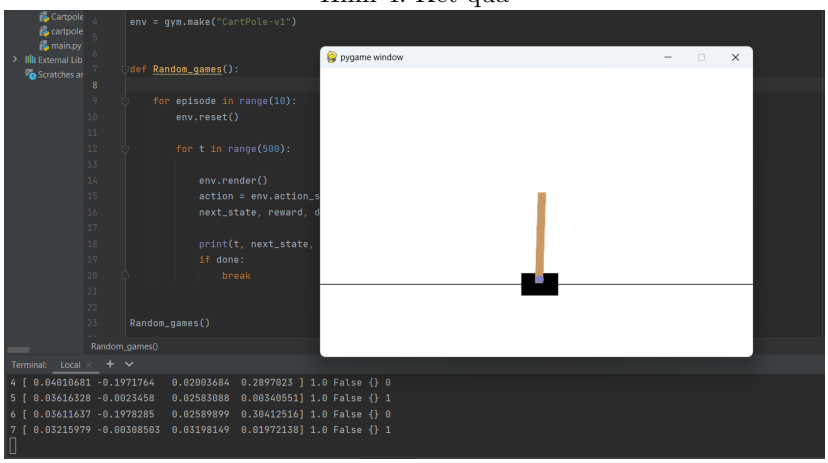
những môi trường đơn giản nhất trong OpenAI gym (các môi trường dùng để phát triển và thử nghiệm các thuật toán Rl).

10 Khởi tạo môi trường CartPole

```python
import gym
import random

env = gym.make("CartPole-v1")


def Random_games():

    for episode in range(10):
        env.reset()

        for t in range(500):

            env.render()
            action = env.action_space.sample()
            next_state, reward, done, info = env.step(action)

            print(t, next_state, reward, done, info, action)
            if done:
                break


Random_games()
```

Hình 3: Một môi trường CartPole hoạt động ngẫu nhiên

Hình 4: Kết quả



Mục tiêu của CartPole là cân bằng một cột được gắn trên một chiếc xe có thể di chuyển. Tác nhân thể hiện việc di chuyển xe sang trái hoặc phải qua số 0 và

1 .
Với môi trường mà ta đang tạo, nó sẽ hoạt động một cách ngẫu nhiên, giúp ta hiểu nó trước khi giúp nó có thể học.

11 Tạo mạng Nơ-ron bằng Keras

Mạng nơ-ron tìm hiểu dựa trên cặp dữ liệu đầu vào và đầu ra mẫu, đoán trước kết quả dựa vào đầu vào không nhìn thấy được.Chúng ta coi mạng nơ ron như một hộp đen ánh xạ dữ liệu đầu vào tới dữ liệu đầu ra, mà không bàn tới tạo sao lại vậy.

```python
import gym
import numpy as np
from collections import deque
from keras.models import Model, load_model
from keras.layers import Input, Dense
from keras.optimizers import Adam, RMSprop

def OurModel(input_shape, action_space):
    X_input = Input(input_shape)

    # 'Dense' là đơn vị cơ sở của lớp mạng nơ ron.
    # lớp ẩn với 512 nodes
    X = Dense(512, input_shape=input_shape, activation="relu", kernel_initializer='he_uniform')(X_input)

    # lớp ẩn với 256 nodes
    X = Dense(256, activation="relu", kernel_initializer='he_uniform')(X)

    # lớp ẩn với 64 nodes
    X = Dense(64, activation="relu", kernel_initializer='he_uniform')(X)

    # Lớp đầu ra
    X = Dense(action_space, activation="linear", kernel_initializer='he_uniform')(X)

    model = Model(inputs = X_input, outputs = X, name='CartPole_DQN_model')

    # Biên dịch mô hình:xác định optimizer để kiểm soát tốc độ học tập và hàm mất mát
    model.compile(loss="mse", optimizer=RMSprop(lr=0.00025, rho=0.95, epsilon=0.01), metrics=["accuracy"])

    model.summary()
    return model
```

Hình 5: Mạng Nơ-ron

Để mạng nơ-ron hiểu và dự đoán dựa trên dữ liệu từ môi trường, chúng ta sẽ khởi tạo mô hình và cung cấp thông tin cho nó. Sau đó, mô hình sẽ được đào tạo bằng dữ liệu đầu vào để tính toán gần đúng dữ liệu đầu ra.

Ta tạo ra các lớp mạng nơ ron bằng cách sử dụng lớp Dense trong thư viện Keras. Với các tham số:số lượng nơ-ron trong lớp, input shape(kích thước của đầu vào cho lớp mạng.), activation(hàm kích hoạt được sử dụng cho các nơ ron trong lớp.) kernel initializer(phương pháp khởi tạo trọng số cho các nơ-ron trong lớp. Ở đây, phương pháp khởi tạo được sử dụng là he uniform.)

12 Triển khai Deep Q Network (DQN)

Phần quan trọng nhất của mạng nơ-ron đó chính là hàm loss, nó được hiểu là sai số giữa Q thực tế và Q dự đoán, hàm cho biết giá trị dự đoán cách mục tiêu thực tế của chúng ta bao xa. Về căn bản chúng ta tính giá trị Q tối đa trong tương lai và chiết khấu nó rồi cộng với giá trị phần thưởng hiện tại trừ đi giá trị dự đoán hiện tại sẽ dẫn đến giá trị thua lỗ(loss). Phép bình phương chỉ để làm mất dấu âm.

$$loss = (r + \gamma max Q'(s, a') - Q(s, a))^2$$

Trong trò chơi CartPole ,ví dụ trường hợp khi cột nghiêng sang trái, phần thưởng dự kiến trong tương lai của việc nhấn nút bên trái sẽ cao hơn khi nhấn nút bên phải vì nó sẽ giúp cộ đứng lâu hơn. Để thực hiện điều này, hàm loss sẽ được sử dụng dưới dạng một công thức để tối ưu hóa.

Trong Keras, chúng ta không cần phải tính toán hàm mất mát (loss function) và đạo hàm của nó (gradient) để cập nhật trọng số của mạng nơ-ron. Thay vào đó, Keras sẽ tự động thực hiện các bước này cho chúng ta.

Khi sử dụng hàm fit() trong Keras để huấn luyện mạng nơ-ron, Keras sẽ tự động tính toán hàm mất mát bằng cách lấy sự khác biệt giữa giá trị thực tế (target) và giá trị dự đoán của mạng nơ-ron, sau đó bình phương khoảng cách này để đưa ra một giá trị mất mát (loss value).

Sau đó, Keras sử dụng thuật toán lan truyền ngược (backpropagation) để tính toán gradient của hàm mất mát đó. Gradient này được sử dụng để cập nhật trọng số của mạng nơ-ron thông qua quá trình tối ưu hoá (optimization process), như sử dụng thuật toán gradient descent để điều chỉnh các tham số trong mạng nơ-ron.

Khi chúng ta lặp lại quá trình cập nhật trọng số nhiều lần, giá trị mất mát sẽ giảm, và độ chính xác của mạng nơ-ron sẽ tăng lên. Khi đạt được một mức độ chính xác đủ cao, chúng ta có thể sử dụng mạng nơ-ron đã huấn luyện để dự đoán kết quả cho các dữ liệu mới mà nó chưa từng thấy trước đó.

13 Hàm ghi nhớ

Để thực hiện chức năng học tập, ta cần ghi nhớ kinh nghiệm với trải nghiệm cũ bên cạnh học trải nghiệm mới.Chúng ta sẽ gọi mảng trải nghiệm này là memory và sử dụng hàm memory() để thêm trạng thái, hành động, phần thưởng và trại thái tiếp theo vào memory.Chức năng ghi nhớ sẽ lưu trữ các trạng thái, hành động và kết quả phần thưởng vào memory:

Hình 6: Hàm ghi nhớ

```
def remember(self, state, action, reward, next_state, done):
    self.memory.append((state, action, reward, next_state, done))
    if len(self.memory) > self.train_start:
        if self.epsilon > self.epsilon_min:
            self.epsilon *= self.epsilon_decay
```

Điều kiện if self.epsilon > self.epsilon_min được sử dụng để kiểm tra xem giá trị của biến epsilon có nhỏ hơn giá trị tối thiểu epsilon_min hay không. Nếu điều kiện đúng, tức là epsilon vẫn còn lớn hơn giá trị tối thiểu, thì giá trị của epsilon sẽ được giảm dần theo một tỉ lệ xác định bởi epsilon_decay.

epsilon là giá trị sử dụng trong thuật toán epsilon-greedy để điều khiển việc lựa chọn hành động của mô hình. Khi epsilon lớn, mô hình sẽ có xu hướng lựa chọn các hành động ngẫu nhiên để khám phá không gian hành động. Khi epsilon nhỏ, mô hình sẽ có xu hướng lựa chọn các hành động tốt nhất được học từ dữ liệu để tối ưu hóa hàm mục tiêu.

Nếu epsilon không giảm dần theo thời gian, mô hình sẽ tiếp tục thực hiện các hành động ngẫu nhiên dù đã học được nhiều thông tin từ dữ liệu. Điều này có thể dẫn đến việc mô hình không hội tụ hoặc hội tụ chậm. Vì vậy, việc giảm dần giá trị của epsilon là cần thiết để mô hình có thể khám phá không gian hành động và tối ưu hóa hàm mục tiêu một cách hiệu quả.

14 Hàm Replay()

Để huấn luyện mạng nơ-ron với các trải nghiệm trong memory, chúng ta gọi replay(). Đầu tiên, lấy mẫu một số trải nghiệm từ memory(gọi là minibatch), đoạn code minibatch=random.sample(memory, min(len(memory),batch_size)) dùng để tạo minibatch.

Với Batch size là số lượng mẫu dữ liệu trong một lần huấn luyện

Hình 7: Hàm replay

```
def replay(self):         #  đào tạo NN với những trải nghiệm được lấy mẫu từ bộ nhớ
    if len(self.memory) < self.train_start:
        return
    # lấy mẫu ngẫu nhiên minibatch từ bộ nhớ
    minibatch = random.sample(self.memory, min(len(self.memory), self.batch_size))

    state = np.zeros((self.batch_size, self.state_size))
    next_state = np.zeros((self.batch_size, self.state_size))
    action, reward, done = [], [], []

    for i in range(self.batch_size):
        state[i] = minibatch[i][0]
        action.append(minibatch[i][1])
        reward.append(minibatch[i][2])
        next_state[i] = minibatch[i][3]
        done.append(minibatch[i][4])

    target = self.model.predict(state)
    target_next = self.model.predict(next_state)

    for i in range(self.batch_size):
        # hiệu chỉnh giá trị Q cho hành động được sử dụng
        if done[i]:
            target[i][action[i]] = reward[i]
        else:
            # Tiêu chuẩn - DQN
            # DQN chọn giá trị Q tối đa trong các hành động tiếp theo
            # lựa chọn và đánh giá hành động trên Q Network
            # Q_max = max_a' Q_target(s', a')
            target[i][action[i]] = reward[i] + self.gamma * (np.amax(target_next[i]))

    # Huấn luyện Neural Network với các lô
    self.model.fit(state, target, batch_size=self.batch_size, verbose=0)
```

Đoạn code trên sử dụng một vòng lặp để lưu trữ các trạng thái, hành động, phần thưởng, trạng thái tiếp theo và trạng thái kết thúc của các kinh nghiệm trong minibatch.

Sau đó, hàm thực hiện dự đoán giá trị Q cho các trạng thái trong state và lưu trữ vào biến target. Tương tự, giá trị Q cho các trạng thái tiếp theo trong next_state được dự đoán và lưu trữ vào biến target_next.

Nếu trạng thái hiện tại là trạng thái kết thúc (done[i] == True), giá trị Q cho hành động được thực hiện trong trạng thái đó được cập nhật thành phần thưởng (reward[i]). Nếu không, giá trị Q cho hành động đó được cập nhật bằng tổng của phần thưởng hiện tại và giá trị Q tối đa của trạng thái tiếp theo (reward[i] + self.gamma * (np.amax(target_next[i]))) theo công thức Q-learning.

15 Hàm Run()

Dưới đây là code chịu trách nhiệm đào tạo mô hình DQN. Chúng ta cho chạy 1000 tập của trò chơi để huấn luyện.Khi thực hiện bằng sai, mô hình vẫn tiếp

tục đào tạo. Chúng ta lưu kết quả từ mọi bước vào memory. Khi mô hình đạt 500 điểm, chúng ta giữ nó và dùng nó để thử nghiệm.

Hình 8: Hàm run

```
import matplotlib.pyplot as plt
def run(self):
    a=[]
    for e in range(self.EPISODES):
        state = self.env.reset()
        state = np.reshape(state, [1, self.state_size])
        done = False
        i = 0
        while not done:
            self.env.render()
            action = self.act(state)   #hành động là 0 hoặc 1;
            next_state, reward, done, _ = self.env.step(action)    # Tác nhân tương tác với Env, nhận phản hồi.
            next_state = np.reshape(next_state, [1, self.state_size])
            if not done or i == self.env._max_episode_steps-1:
                reward = reward
            else:
                reward = -100
            self.remember(state, action, reward, next_state, done) # Ghi nhớ trạng thái, hành động, phần thưởng của Timestep trước đó.
            state = next_state
            i += 1
            a.append(i)

            import matplotlib.pyplot as plt

            if done:
                print("episode: {}/{}, score: {}, e: {:.2}".format(e, self.EPISODES, i, self.epsilon))
                print(a)
                plt.plot(a)
                plt.show()
                if i == 500:
                    print("Saving trained model as cartpole-dqn.h5")
                    self.save("cartpole-dqn.h5")
                    return
            self.replay()
```

score (được tính bằng biến i) là số lần thực hiện hành động liên tiếp mà mô hình đã thực hiện trong một tập lặp (episode) của trò chơi.score được in ra màn hình để hiển thị số lần thực hiện hành động tối đa mà mô hình đã đạt được trong tập lặp đó.

16 Code hoàn chỉnh

Hình 9: Import các thư viện và môi trường cần thiết

```
# Tutorial by www.pylessons.com
# Tutorial written for - Tensorflow 1.15, Keras 2.2.4

import os
os.environ['SDL_VIDEODRIVER']="dummy"
os.environ['CUDA_VISIBLE_DEVICES'] = '-1'
import random
import gym
import numpy as np
from collections import deque
from keras.models import Model, load_model
from keras.layers import Input, Dense
from keras.optimizers import Adam, RMSprop
```

Hình 10: Mạng Nơ-ron

```
def OurModel(input_shape, action_space):
    X_input = Input(input_shape)

    # 'Dense' là đơn vị cơ sở của lớp mạng nơ ron.
    # lớp ẩn với 512 nodes
    X = Dense(512, input_shape=input_shape, activation="relu", kernel_initializer='he_uniform')(X_input)

    # lớp ẩn với 256 nodes
    X = Dense(256, activation="relu", kernel_initializer='he_uniform')(X)

    # lớp ẩn với 64 nodes
    X = Dense(64, activation="relu", kernel_initializer='he_uniform')(X)

    # Lớp đầu ra
    X = Dense(action_space, activation="linear", kernel_initializer='he_uniform')(X)

    model = Model(inputs = X_input, outputs = X, name='CartPole_DQN_model')
    model.compile(loss="mse", optimizer=RMSprop(lr=0.00025, rho=0.95, epsilon=0.01), metrics=["accuracy"])

    model.summary()
    return model
```

Hình 11: Khai báo DQNAgent

```
class DQNAgent:
    def __init__(self):
        self.env = gym.make('CartPole-v1')

        self.state_size = self.env.observation_space.shape[0]
        self.action_size = self.env.action_space.n
        self.EPISODES = 1000
        self.memory = deque(maxlen=2000)

        self.gamma = 0.95    # Hệ số chiết khấu
        self.epsilon = 1.0   # Tỷ lệ thăm dò: Bao nhiêu để hành động ngẫu nhiên
        self.epsilon_min = 0.001 # Số lượng khám phá ngẫu nhiên tối thiểu được phép
        self.epsilon_decay = 0.999 # Giảm số lượng khám phá ngẫu nhiên khi hiệu suất của tác nhân cải thiện theo thời gian
        self.batch_size = 64    #Xác định dung lượng bộ nhớ mà DQN sẽ sử dụng để huấn luyện;
        self.train_start = 1000

        self.model = OurModel(input_shape=(self.state_size,), action_space = self.action_size)
```

Hình 12: Hàm remember

```
    def remember(self, state, action, reward, next_state, done):
        self.memory.append((state, action, reward, next_state, done))
        if len(self.memory) > self.train_start:
            if self.epsilon > self.epsilon_min:
                self.epsilon *= self.epsilon_decay
```

Hình 13: Hàm act

```
    def act(self, state):
        if np.random.random() <= self.epsilon:    # Nếu hành động ngẫu nhiên, hãy thực hiện hành động ngẫu nhiên
            return random.randrange(self.action_size)
        else:                                     # Nếu không hành động ngẫu nhiên, hãy dự đoán giá trị phần thưởng dựa trên trạng thái hiện tại
            return np.argmax(self.model.predict(state)) #Chọn hành động sẽ cho phần thưởng cao nhất (tức là, đi bên trái hay phải )
```

Hình 14: Hàm replay

```
def replay(self):         #  đào tạo NN với những trải nghiệm được lấy mẫu từ bộ nhớ
    if len(self.memory) < self.train_start:
        return
    # lấy mẫu ngẫu nhiên minibatch từ bộ nhớ
    minibatch = random.sample(self.memory, min(len(self.memory), self.batch_size))

    state = np.zeros((self.batch_size, self.state_size))
    next_state = np.zeros((self.batch_size, self.state_size))
    action, reward, done = [], [], []

    for i in range(self.batch_size):
        state[i] = minibatch[i][0]
        action.append(minibatch[i][1])
        reward.append(minibatch[i][2])
        next_state[i] = minibatch[i][3]
        done.append(minibatch[i][4])

    target = self.model.predict(state)
    target_next = self.model.predict(next_state)

    for i in range(self.batch_size):
        # hiệu chỉnh giá trị Q cho hành động được sử dụng
        if done[i]:
            target[i][action[i]] = reward[i]
        else:
            # Tiêu chuẩn - DQN
            # DQN chọn giá trị Q tối đa trong các hành động tiếp theo
            # lựa chọn và đánh giá hành động trên Q Network
            # Q_max = max_a' Q_target(s', a')
            target[i][action[i]] = reward[i] + self.gamma * (np.amax(target_next[i]))

    # Huấn luyện Neural Network với các lô
    self.model.fit(state, target, batch_size=self.batch_size, verbose=0)
```

Hình 15: Hàm load và save

```
def load(self, name):
    self.model = load_model(name)

def save(self, name):
    self.model.save(name)
```

Hình 16: Hàm run

```
import matplotlib.pyplot as plt
def run(self):
    a=[]
    for e in range(self.EPISODES):
        state = self.env.reset()
        state = np.reshape(state, [1, self.state_size])
        done = False
        i = 0
        while not done:
            self.env.render()
            action = self.act(state)   #hành động là 0 hoặc 1;
            next_state, reward, done, _ = self.env.step(action)    # Tác nhân tương tác với Env, nhận phản hồi.
            next_state = np.reshape(next_state, [1, self.state_size])
            if not done or i == self.env._max_episode_steps-1:
                reward = reward
            else:
                reward = -100
            self.remember(state, action, reward, next_state, done) # Ghi nhớ trạng thái, hành động, phần thưởng của Timestep trước đó.
            state = next_state
            i += 1
            a.append(i)

            import matplotlib.pyplot as plt

            if done:
                print("episode: {}/{}, score: {}, e: {:.2}".format(e, self.EPISODES, i, self.epsilon))
                print(a)
                plt.plot(a)
                plt.show()
                if i == 500:
                    print("Saving trained model as cartpole-dqn.h5")
                    self.save("cartpole-dqn.h5")
                    return
            self.replay()
```

Hình 17: Hàm test

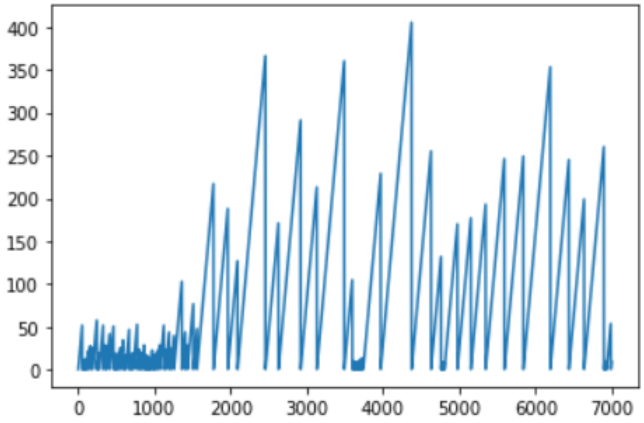
```
def test(self):
    a=[]
    self.load("cartpole-dqn.h5")
    for e in range(self.EPISODES):
        state = self.env.reset()
        state = np.reshape(state, [1, self.state_size])
        done = False
        i = 0
        while not done:
            self.env.render()
            action = np.argmax(self.model.predict(state))
            next_state, reward, done, _ = self.env.step(action)
            state = np.reshape(next_state, [1, self.state_size])
            i += 1
            a.append(i)
            if done:
                print("episode: {}/{}, score: {}".format(e, self.EPISODES, i))
                import matplotlib.pyplot as plt
                plt.plot(a)
                plt.show()
                break
```

17 Kết quả chạy code

- Ta dùng chế độ huấn luyện (agent.run()).Qúa trình huấn luyện dừng lại ở episode thứ 104 khi điểm đạt được bằng 500, lưu kết quả huấn luyện model_weights vào file cartpole_dqn.h5 .

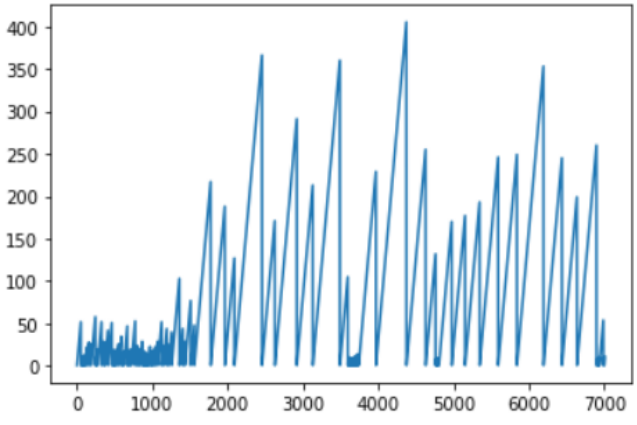
Hình 18: Ở episode 102, score đạt 10.

```
1/1 [==============================] - 0s 27ms/step
episode: 102/1000, score: 10, e: 0.0025
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,
```

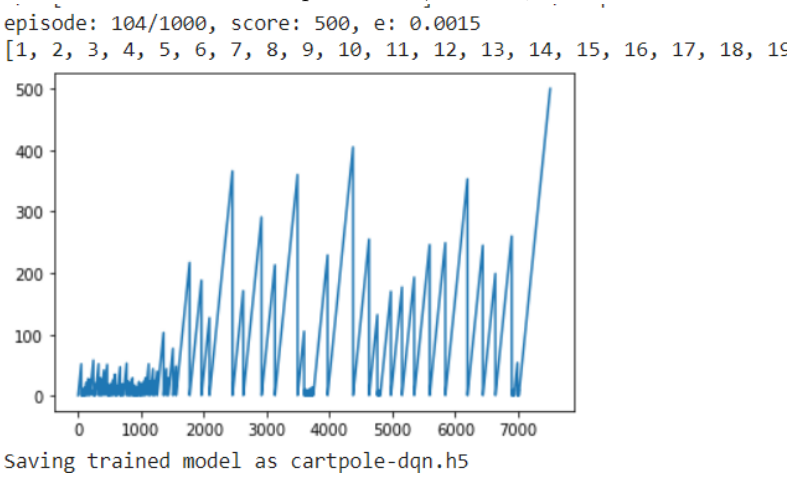


Hình 19: Ở episode 103, score đạt 11.

```
1/1 [==============================] - 0s 33ms/step
episode: 103/1000, score: 11, e: 0.0024
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,
```

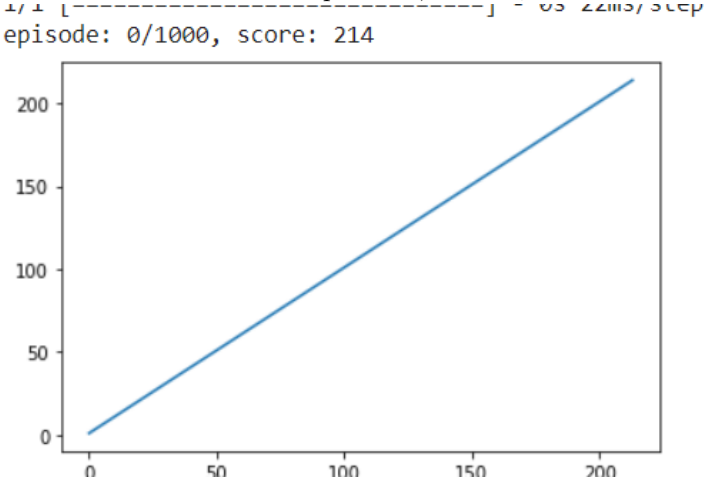


Hình 20: Ở episode 104, score đạt 500.

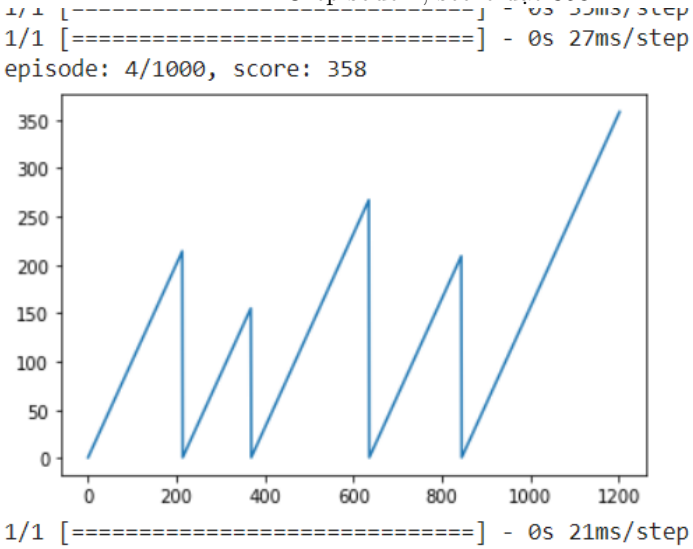


- Chuyển sang chế độ thử nghiệm(agent.test()). Với dữ liệu ta đã có được bằng cách huấn luyện tác nhân, điểm số đạt được qua mỗi episode của chế độ thử nghiệm tốt hơn rất nhiều.

Hình 21: Ở episode 0, score đạt 214.



Hình 22: Ở episode 4, score đạt 358.



18 Kết luận

Học tăng cường đã đi được con đường dài từ khi xuất hiện, phát triển và trưởng thành theo nhiều hướng. Nó đã trở thành một trong những lĩnh vực được nghiên cứu tích cực trong học máy, trí tuệ nhân tạo, mạng thần kinh . Học tăng cường đã phát triển được một nền tảng toán học hỗ trợ mạnh mẽ , trở thành một lĩnh vực rộng lớn với các nhà nghiên cứu đến từ nhiều quốc gia trên thế giới trong các lĩnh vực khác nhau như vật lý, lý thuyết điều khiển,...