GAME 2048 ÁP DỤNG DEEP Q-NETWORK

Lê Trung Hiếu, Hoàng Thế Kỷ, Lê Anh Minh

{18520738, 18520964, 18521098}@gm.uit.edu.vn

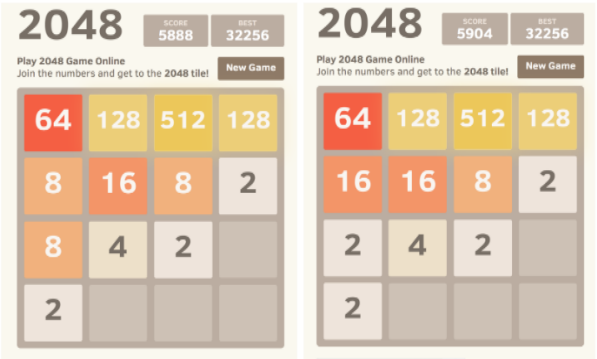
Trường Đại học Công nghệ Thông tin - ĐHQG Thành phố Hồ Chí Minh

TỔNG QUAN

2048 là một tựa game đơn giản, có lối chơi đơn giản, những người từng trải nghiệm sẽ thấy nó thú vị nhưng nó sẽ rất khó cho đến 1 mức độ nào đó sẽ không dễ dàng để chiến thắng . Ở đồ án này, chúng tôi xây dựng lại game 2048 dựa trên thư viện có sẵn OpenGym ở mức console. Chúng tôi tạo ra một agent để tự động chơi game và tối ưu hóa điểm số giành được bằng thuật toán Deep Q-Network.

I. GIỚI THIỆU

2048 là một trò chơi giải đố các khối vuông dành cho một người chơi được thiết kế bởi lập trình viên trẻ tuổi người Ý, Gabriele Cirulli vào tháng 3 năm 2014. Mục tiêu chính của trò chơi là trượt các khối được đánh số trên lưới để kết hợp chúng và tạo ra một khối có giá trị 2048. Game 2048 đã làm mưa làm gió trong suốt thời gian của năm đó và cho đến tận bây giờ thì tựa game này vẫn luôn lọt top các trò chơi trí tuệ, hấp dẫn nhất mọi thời đại. Mục tiêu của chơi là tối ưu hóa số điểm nhận được, việc đó cần đưa các khối giá trị lớn về các góc, cạnh của màn hình trò chơi để có ưu thế giành chiến thắng. Nhưng lúc này các khối xuất hiện ngẫu nhiên (mang giá trị 2 hoặc 4) sẽ là những rủi ro chặn các khối có giá trị lớn rời khỏi các góc, cạnh, viên vì không có cách di chuyển nào khác. Các khối có số điểm bằng nhau mới có thể kết hợp để tạo ra một khối có điểm số lớn hơn. Bên cạnh đó, kích thước của không gian trạng thái của trò chơi là rất lớn. 10^{16} là số trạng thái có thể có trước khi nhận được khối 2048 (2^{11}), và thậm chí còn nhiều hơn nếu chúng ta tiếp tục trò chơi sau tối ưu điểm số.



Hình 1: Ví dụ một nước đi trong game 2048. Hình bên trái là trạng thái hiện tại, hình bên phải là trạng thái sau khi người chơi di chuyển lên trên.

Chúng tôi sẽ dùng công cụ OpenGym với thư viện có sẵn để xây dựng agent Trí tuệ nhân tạo nhằm tối ưu hóa điểm số game 2048 bằng thuật toán Deep Q-Network.

Ở thuật toán Deep Q-Network chúng tôi cài đặt cho Agent đi vào những ô trên Board đã được tính toán bởi Deep Q- Network một cách hợp lí.

II. CÁC PHƯƠNG PHÁP GIẢI QUYẾT

1. Phương pháp Q-learning

Q-learning sẽ học hàm giá trị của hành động $Q(s, a)$: thực hiện đánh giá mức độ hiệu quả để thực hiện một hành động trong một trạng thái cụ thể. Chúng ta sẽ gán một giá trị vô hướng dựa trên lợi ích của việc thực hiện một bước đi. Q được gọi là hàm giá trị của hành động hay hàm giá trị Q.

Trong phương pháp Q-learning, chúng ta xây dựng một bảng bộ nhớ $Q(s, a)$ để lưu trữ các giá trị Q cho tất cả các kết hợp có thể có giữa s và a. Và đó sẽ là một bảng gian lận mà chứa các bước đi tốt nhất.

Về mặt kỹ thuật mà nói, chúng ta sẽ lấy mẫu một hành động từ trạng thái hiện tại, rồi tìm ra phần thưởng r (nếu có) và trạng thái mới s' (vị trí mới). Sau đó, từ

bảng bộ nhớ chúng ta sẽ xác định thực hiện hành động tiếp theo a' mà có giá trị $Q(s', a')$ là cao nhất.

2. Thuật toán Q-learning

Giả sử chúng ta biết trước phần thưởng mong đợi của mỗi hành động ở mỗi bước. Điều này về cơ bản sẽ giống như một bảng gian lận cho các Agent. Agent của chúng tôi sẽ biết chính xác hành động nào cần thực hiện.

Nó sẽ thực hiện chuỗi hành động mà cuối cùng sẽ tạo ra tổng phần thưởng tối đa. Tổng phần thưởng này còn được gọi là Q-value với công thức chiến lược như sau:

$$Q(s,a) = r(s,a) + \gamma \max_{a'} Q(s',a')$$

Phương trình trên cho biết Q-value sinh ra khi ở trạng thái s và thực hiện hành động a là phần thưởng tức thì $r(s,a)$ cộng với Q-value cao nhất có thể từ trạng thái tiếp theo s'. Gamma ở đây là hệ số chiết khấu kiểm soát sự đóng góp của phần thưởng trong tương lai.

$Q(s',a')$ phụ thuộc vào $Q(s'',a'')$, trong khi $Q(s'',a'')$ sẽ có hệ số gamma được bình phương lên. Vì vậy, Q-value phụ thuộc vào các Q-value của các trạng thái trong tương lai như sau:

$$Q(s,a) \rightarrow \gamma Q(s',a') + \gamma^2 Q(s'',a'') + ... \gamma^n Q(s''^{...n}, a''^{...n})$$

Điều chỉnh giá trị của gamma sẽ làm giảm hoặc tăng sự đóng góp của phần thưởng trong tương lai.

Vì đây là một phương trình đệ quy, chúng ta có thể bắt đầu với việc đưa ra các giả thiết tùy ý cho tất cả các Q-value. Với kinh nghiệm nhận được, nó sẽ hội tụ thành chính sách tối ưu, điều này được triển khai dưới dạng:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha \left[r_{t+1} + \gamma \max_{a_{t+1}} Q(s_{t+1},a_{t+1}) - Q(s_t,a_t) \right]$$

Trong đó alpha là tốc độ học. Công thức trên chỉ đơn giản là xác định mức độ thông tin mới mà chúng ta có được sẽ ghi đè thông tin cũ.

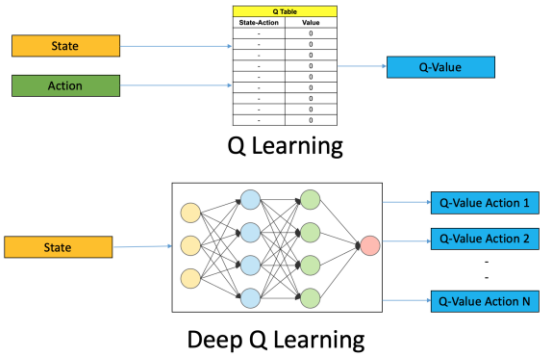
3. Greedy Epsilon

Greedy Epsilon giúp cân bằng giữa *exploration* (khám phá) và *exploitation* (khai thác). Chúng ta sẽ phát sinh ngẫu nhiên một xác suất, nếu xác suất đó nhỏ hơn hoặc bằng *epsilon* thì chúng ta sẽ chọn ngẫu nhiên một hành động, nếu xác suất đó lớn hơn *epsilon* thì chúng ta sẽ chọn hành động có giá trị $Q(s,a)$ lớn nhất. *Epsilon* sẽ giảm dần qua các episode, nên khi *epsilon* càng cao thì agent sẽ ưu tiên *exploration* hơn, để có thể khám phá ra các trạng thái mới, biết được các hành sẽ dẫn tới trạng thái tương lai tốt. Về sau, khi epsilon giảm dần, agent sẽ ưu tiên exploitation hơn để có thể tối ưu hóa tổng điểm thưởng, bởi vì agent đã khám phá đủ nhiều rồi, nên bây giờ cần ưu tiên tổng điểm thưởng hơn là *exploration*.

$$\text{Action at time (t)} = \begin{cases} \max Q_t(s,a), & \text{with probility 1- } \in \\ \text{any action(a), with probility } \ni \end{cases}$$

4. Deep Q-Learning

Trong Deep Q-learning, chúng tôi sử dụng mạng nơ-ron để tính gần đúng hàm Q-value. Trạng thái sẽ làm đầu vào và Q-value của tất cả các hành động có thể được tạo ra sẽ làm đầu ra. Dưới đây là minh họa sự so sánh giữa Q-learning và Deep Q-learning:



Hình 2: Sự khác nhau giữa Q Learning và Deep Q Learning.

Mục đích của ta là bắt Neural Network học được cách ước lượng Q-Value cho các actions dựa trên làm Loss. Nó sẽ ước lượng sai số giữa dự đoán và thực tế.

Công thức hàm Loss:

$$L = (r + \gamma \, max_{a'} Q(s',a') - Q(s,a))^2$$

5. Deep Q-Network

Deep Q-Network được phát triển bởi nhóm DeepMind vào năm 2015. Nó có thể giải quyết một loạt các trò chơi như 2048 bằng cách kết hợp Reinforcement Learning và các Deep Neural Network với quy mô lớn. Thuật toán được phát triển bằng cách nâng cao thuật toán Reinforcement Learning cổ điển, được gọi là Q-Learning, với các Deep Neural Network và một kỹ thuật là Experience replay.

Cấu trúc của Deep Q-Network:

Khởi tạo replay memory
Khởi tạo Q-network với trọng số ngẫu nhiên
Tạo một Target-network là bản sao của Q-network
Với mỗi episode:
Khởi tạo trạng thái bắt đầu (state)
Với mỗi lần lặp:
Chọn action
Thực hiện action
Quan sát reward và state tiếp theo
Lưu experience vào replay memory
Lấy sample ngẫu nhiên từ replay memory
Xử lý state ở bước 5
Chuyển các state đó vào Q-network
Tính loss giữa Q-Value(được tính từ state của Q-Network) và Target Q-Network

6. Experience replay

Experience sẽ lưu trữ và lấy mẫu từ replay memory sau này.

experience = (state, action, reward, next_state)

Các state có nguy cơ thực hiện những action giống nhau một cách liên tục dẫn đến việc overfitting. Kỹ thuật này giúp giảm thiểu việc đó, giúp cho thuật toán trở nên hiệu quả hơn.

7. Mạng mục tiêu (Target-network)

Chúng ta sẽ tạo ra hai mạng học sâu θ- và θ, sử dụng mạng đầu tiên để lấy ra các Q-value trong khi mạng thứ hai chứa tất cả các cập nhật trong quá trình đào tạo. Sau quá trình cập nhật, chúng ta sẽ đồng bộ hóa θ- với θ để giữ nguyên cố định các Q-value mục tiêu tạm thời.

$$L_i(\theta_i) = E_{s,a,s',r\sim D}(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i))^2$$

Trong đó, $r + \gamma \max_{a'} Q(s', a'; \theta_i^-)$ là biểu thức của mục tiêu.

8. Gradient Descent

Gradient Descent nhằm mục đích giúp cho Q-value hội tụ, tức là Q-value sẽ đạt được một ngưỡng reward tối đa nào đó nếu hàm Loss ở trên tiệm cận 0.

9. Hàm mất mát Mean Square Error

Các thành viên trong nhóm đều không có nhiều kinh nghiệm đối với mạng học sâu (Deep Learning). Do đó, chúng tôi chọn hàm mất mát MSE, một trong các hàm mất mát khá phổ biến.

MSE sẽ tính sai số giữ Q-value và Target.

10. Thuật toán Optimization Adam

Thuật toán này tương tự Gradient Descent nhưng nó sẽ tìm được nghiệm Global (tức là giá trị Q-Value đạt được sẽ tốt hơn)

Đối với thuật toán optimization, chúng tôi dùng Adam vì thuật toán này được sử dụng khá nhiều

III. THỰC NGHIỆM

Trò chơi 2048 có không gian trạng thái rất lớn, mỗi ô đều có giá trị là lũy thừa bậc n của 2, với n nguyên dương. Do đó, chúng tôi quyết định áp dụng 3 kiến trúc mạng do chúng tôi tự thiết kế để thử nghiệm, sau đó sẽ so sánh kết quả của 3 kiến trúc mạng khi huấn luyện với 1000 episode và 2000 episode.

1. Các kiến trúc mạng

Cả 3 kiến trúc mạng đều có lớp đầu vào là một vector 16 chiều và đầu ra là một vector 4 chiều. Vector 16 chiều chứa giá trị của 16 ô của trò chơi, còn vector 4 chiều là 4 giá trị $Q(s, a)$ của 4 hành động lên, xuống, trái và phải.

- Kiến trúc đầu tiên: có 1 hidden layer gồm 64 node.
- Kiến trúc thứ hai: 2 hidden layer, mỗi hidden layer gồm có 64 node.
- Kiến trúc thứ ba: 3 hidden layer, mỗi hidden layer gồm có 64 node.

Vì vấn đề tài chính và kỹ thuật nên 3 kiến trúc mạng trên đều được huấn luyện bằng CPU trên máy tính cá nhân. Cấu hình phần cứng và phần mềm chạy thực nghiệm:

- CPU: Intel® Core™ i7-7820HQ CPU @ 2.90GHz (CPUs), ~2.9GHz
- RAM: 8 GB
- Ngôn ngữ lập trình: Python 3.8.5

2. Hyperparameters

Giá trị epsilon tối đa	1.0
Giá trị epsilon tối thiểu	0.1
Giá trị gamma	0.99
Kích thước bộ nhớ experience replay	5,000
Kích thước batch	64
Tần suất cập nhật Target-network	1,000
Số lần chơi để huấn luyện mô hình (hoặc episodes)	1,000; 2,000
Số lần suy giảm epsilon	episodes

Số hành động tối đa trong một episode (khi huấn luyện)	10,000

Bảng 1: Các siêu tham số (hyperparameter) được dùng khi huấn luyện các kiến trúc mạng. "Tần suất cập nhật Target-network" là số hành động cần thực hiện (có thể cộng dồn của nhiều episode) để có thể cập nhật Target-network.

Chúng tôi có thử nghiệm vài bộ hyperparameter nhưng trong số đó thì bộ hyperparameter phía trên cho kết quả tốt nhất, sau đây là kết quả của bộ tham số trên kết hợp với 3 kiến trúc mạng.

3. Kết quả

	Số lượng giá trị đạt được			
Kiến trúc	**< 128**	**128**	**256**	**512**
1 hidden layer	62	26	12	0
2 hidden layer	46	44	10	0
3 hidden layer	60	31	9	0

Bảng 2: Số lượng các giá trị "nhỏ hơn 128", 128, 256 và 512 của các mạng trong 100 lần chơi, khi được huấn luyện với 1000 episode.

	Số lượng giá trị đạt được			
Kiến trúc	**< 128**	**128**	**256**	**512**
1 hidden layer	62	31	6	1
2 hidden layer	58	32	10	0
3 hidden layer	53	36	11	0

Bảng 3: Số lượng các giá trị "nhỏ hơn 128", 128, 256 và 512 của các mạng trong 100 lần chơi, khi được huấn luyện với 2000 episode.

"Giá trị đạt được" là giá trị của ô lớn nhất trong 16 ô của trò chơi 2048. Chúng tôi chia ra 4 mốc giá trị như trên là vì các giá trị nhỏ hơn 128 đều có thể dễ dàng đạt được bằng việc thực hiện các hành động ngẫu nhiên. Hai mốc giá trị 128 và 256 là các giá trị cao nhất mà cả 3 kiến trúc mạng có thể đạt được khá dễ dàng. Và mốc cuối cùng là 512, giá trị này cũng may mắn xuất hiện được khoảng dưới 10 lần trong khi huấn luyện và thử nghiệm các bộ hyperparameter, nhưng do chỉ là may mắn nên giá trị này nhằm để biểu thị giới hạn hiện tại của cả 3 mô hình và bộ hyperparameter.

Và có một điều rất thú vị khi chúng tôi dùng các kiến trúc mạng đã được huấn luyện để chơi trò chơi. Cả 3 kiến trúc mạng, dù là bộ tham số nào thì cũng đều gặp hiện tượng bị kẹt lại ở một trạng thái bất kì, bị kẹt là vì hành động có giá trị $Q(s,a)$ lớn nhất lại không thể thay đổi trạng thái của trò chơi, trong khi vẫn còn các hành động khác có thể thay đổi trạng thái bị kẹt. Nên chúng tôi đã quyết định sẽ chọn một hành động ngẫu nhiên để thực hiện khi hiện tượng này xảy ra. Lý do hiện tượng này không xuất hiện trong quá trình huấn luyện là vì cả 2 mạng Q-network và Target-network đều được cập nhật liên tục, nên cùng một trạng thái thì lúc này có thể là hành động này có giá trị $Q(s,a)$ lớn nhất, lúc khác thì hành động khác sẽ có giá trị $Q(s,a)$ lớn nhất. Khi huấn luyện xong thì Q-network không được cập nhật trọng số nữa, nên sẽ gặp hiện tượng bị kẹt lại ở một hoặc nhiều trạng thái nào đó.

Qua 2 bảng trên, với 6 dòng kết quả cho 3 kiến trúc mạng, được huấn luyện với 1000 episode và 2000 episode, chúng ta có thể nhận thấy rằng kết quả của kiến trúc có "2 hidden layer" được huấn luyện trong "1000 episode" có kết quả tốt nhất so với 5 kết quả còn lại. Vì các giá trị lớn như 128 và 256 đều có số lượng (44 và 10) trên mức trung bình (33.33 và 9.6) khi so với 6 kết quả. Dù kiến trúc "1 hidden layer" được huấn luyện trong "2000 episode" có đạt được 1 lần giá trị 512 nhưng chỉ chiếm 1% trong 100 lần chơi nên có thể coi đây là một kết quả không đáng kể.

IV. Kết luận và hướng phát triển

Trong đồ án này, chúng tôi đã huấn luyện 3 kiến trúc mạng theo thuật toán Deep Q-Learning để chơi trò chơi 2048. Kết quả dần tốt lên khi tăng thêm đồng thời số lượng hidden layer và số lượng episode dùng để huấn luyện. Vì không gian trạng thái của trò chơi là vô tận dựa trên lý thuyết, nên cần một kiến trúc mạng lớn và số lượng episode dùng để huấn luyện đủ nhiều. Nhưng đáng tiếc là cả 3 kiến trúc mạng đều chỉ đạt được các kết quả từ 256 trở xuống, theo như chúng tôi thì đây là kết quả của việc các kiến trúc mạng và bộ hyperparameter chưa được tối ưu, hoặc có thể là thuật toán này không phù hợp với trò chơi này.

Trong tương lai, chúng tôi dự định sẽ nghiên cứu và áp dụng các thuật toán học tăng cường khác để có thể so sánh với kết quả hiện tại, từ đó cải thiện được kết quả.

V. LỜI CÁM ƠN

Xin chân thành cám ơn TS. Lương Ngọc Hoàng – giảng viên khoa Khoa học máy tính, Trường Đại học Công nghệ thông tin, Đại học Quốc gia Thành phố Hồ Chí Minh, đồng thời là giảng viên giảng dạy lớp Trí tuệ nhân tạo – CS106.L21.KHCL đã giúp đỡ trong lúc nhóm chúng tôi thực hiện bài báo này.

VI. TÀI LIỆU THAM KHẢO

[1] Yun Nie, Yun Nie, and Yicheng An, "AI Plays 2048" Stanford University, 2016.

[2] Johnathan Amar, Antoine Dedieu, "Deep Reinforcement Learning for 2048", Operations Research Center Massachusetts Insitute of Technology, 2016

[3] https://www.tensorflow.org/agents/tutorials/0_intro_rl

[4] https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/

[5] https://tek4.vn/deep-q-network-dqn-hoc-tang-cuong-phan-5/