USING REINFORMENT LEARNING TO PLAY KUNG FU MASTER

Phạm Trần Anh Tiên
20522012@gm.uit.edu.vn

Lương Lý Công Thịnh
20521960@gm.uit.edu.vn

Đào Trần Anh Tuấn
20522107@gm.uit.edu.vn

Trần Phú Vinh
20522161@gm.uit.edu.vn

1. GIỚI THIỆU

1.1.REINFOREMENT LEARNING

Reinforment Learning (học tăng cường) là một lĩnh vực mà trong đó sẽ nghiên cứu về cách một hệ thống có thể tự động học được cách đưa ra quyết định dựa trên việc tương tác với môi trường xung quanh nó. Trong Reinforment Learning, hệ thống hay được gọi là tác từ (agents) sẽ tương tác với môi trường (environemnt) để nhận phần thưởng (rewards). Các thuật toán học tăng cường sẽ tập trung vào việc tìm ra được một chiến lược (policy) phù hợp để có thể tối đa reward nhận được trong quá trình tương tác với môi trường.

Deep Reinforment Learning (Deep RL) là một kỹ thuật kết hợp deep learning (học sâu) với reinforment learning (học tăng cường). Deep RL thay vì sử dụng các thuật toán truyền thống thì sẽ sử dụng một mạng học sâu để có thể giúp hệ thống học được một chiến lược hiệu quả hơn.

1.2.ENVIRONMENT

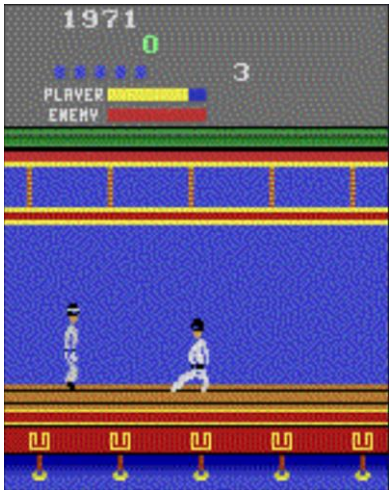
Môi trường được chọn để thực hiện đồ án là **Kung Fu Master**, là một game của **Atari** mà trong đó bạn sẽ đóng vai một bậc thầy Kung-Fu đang trên đường đi đến một ngôi đến của các pháp sư tà ác để giải cứu công chúa Victoria, bạn sẽ phải hạ gục các kẻ địch trên đường đến đó để có thể giải cứu công chúa.

Môi trường **Kung Fu Master** sẽ gồm 14 hành động:

Bảng 1 Các hành động trong môi trường Kung Fu Master

Num	Action
0	NOOP
1	UP
2	RIGHT
3	LEFT
4	DOWN
5	DOWNRIGHT
6	DOWNLEFT
7	RIGHTFIRE
8	LEFTFIRE
9	DOWNFIRE
10	UPRIGHTFIRE
11	UPLEFTFIRE
12	DOWNRIGHTFIRE
13	DOWNLEFTFIRE

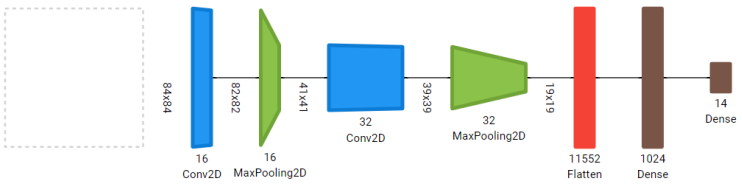
Mặc định môi trường **Kung Fu Master** sẽ trả về một ảnh RGB có dạng (210, 160, 3)



Hình 1. Ví dụ về một observations

2. TIỀN XỬ LÝ VÀ CẤU TRÚC MODEL

Mỗi observations có được từ trò chơi Kung Fu Master đầu tiên sẽ được chuyển thành ảnh xám và resize về kích thước (84, 84) và được đưa vào mô hình.

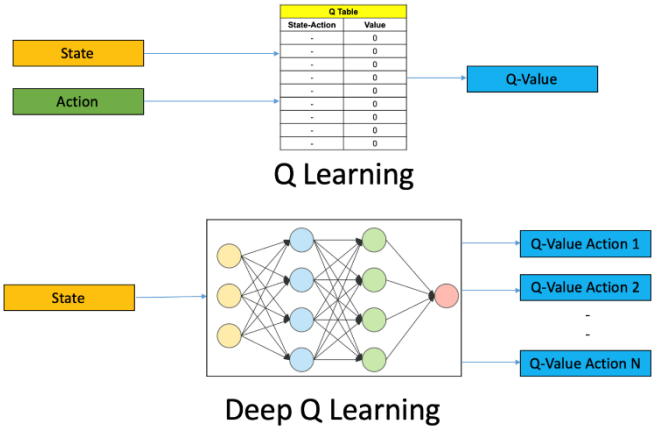


Hình 2. Cấu trúc backbone

3. PHƯƠNG PHÁP SỬ DỤNG

3.1.DEEP Q NETWORK (DQN)

DQN là một thuật toán được phát triển bởi DeepMind vào năm 2015, DQN là một cải tiến từ thuật tuán Q-learning. Hạn chế trước đó của Q-Learning là nếu môi trường có tính liên tục thì sẽ rất khó để có thể lập và học được bảng Q-Table (nếu được thì sẽ tốn rất nhiều bộ nhớ). DQN sẽ sử dụng một mạng DNN để thay thế cho Q-Table, một DQN sẽ nhận vào một state s và cho ra output là một vector gồm các giá trị action-value Q(s, ·, $\boldsymbol{\theta}$) với $\boldsymbol{\theta}$ sẽ là tham số của mô hình



Hình 3. Ảnh mô tả Q-Learning và Deep Q Learning

Tuy nhiên chỉ với mô hình DQN thì thuật toán rất dễ bị tình trạng overfit, để giải quyết vấn đề này ta sẽ sử dụng 2 mô hình DNN, DNN chính gọi là **Q-Network** và một DNN phụ gọi là **Target-Network**. Target-Network sẽ có bộ tham số $\boldsymbol{\theta}^-$ tương tự như của Q-Network và $\boldsymbol{\theta}^-$ sẽ được cập nhật sau mỗi τ steps, Việc sử dụng Target-Network có thể giúp cho mô hình có thể học một cách ổn định hơn và có thể dễ dàng tiến đến điểm hội tụ.

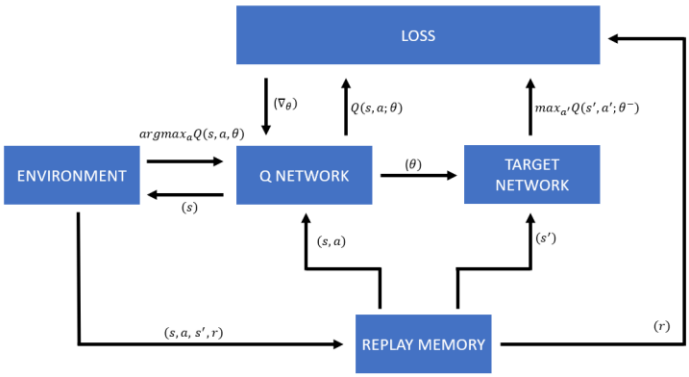
Tham số mô hình Q-Network sẽ được cập nhật như sau:

$$\boldsymbol{\theta_{t+1}} = \boldsymbol{\theta_t} + \alpha(Y_t - Q(S_t,\ A_t;\ \boldsymbol{\theta_t}))\nabla_{\theta_t} Q(S_t,\ A_t;\ \boldsymbol{\theta_t})$$

Với

- $Y_t = R_{t+1} + \gamma \max_{a} Q(S_{t+1},\ a;\ \boldsymbol{\theta_t^-})$
- $\boldsymbol{\theta_t^-}$**:** tham số target network ở step thứ t
- $\boldsymbol{\theta_t}$:tham số q network ở step thứ t

Một vấn đề nữa với DQN đó chính là **catastrophic forgetting**, **catastrophic forgetting** là hiện tượng khi các mô hình sẽ quên mất các thông tin đã được huấn luyện trước đó. Để giảm bớt vấn đề này thì ta sẽ sử dụng một kỹ thuật là **experience replay**. Với mỗi step thì ta sẽ có 1 tuple **(s, a, r, s', done)** (done cho biết trạng thái s' có phải là trạng thái kết thúc hay không) được lưu vào experience replay, khi thực hiện training chúng ta sẽ lấy random một minibatch từ experience replay để training mô hình.



Hình 4. Cấu trúc của DQN khi có Target-Network và Experience Replay

3.2.DOUBLE DEEP Q NETWORK (DDQN)

Do sử dụng hàm **max** khi tính toán giá trị target để cập nhật tham số nên DQN thường bị tình trạng **overestimation**, tình trạng khi mà mô hình xấp xỉ điểm số (reward) của một trạng thái (state) quá cao so với điểm số thực.

DDQN cố gắng giải quyết vấn đề đó bằng cách thay vì sử dụng hàm **max** thì DDQN sẽ chọn giá trị action-value dựa trên **argmax** kết quả đầu ra của Q-Network.

$$Y_t = R_{t+1} + \gamma Q(s'_{t+1}, \operatorname*{argmax}_a Q(S_{t+1}, a; \boldsymbol{\theta_t}), \boldsymbol{\theta_t^-})$$

3.3. PRIORITIZED DDQN

Prioritized Experience Replay (PER) được giới thiệu đầu tiên vào năm 2015 bởi **Tom Schaul**. Ý tưởng của PER là trong các experience được lưu trong memory, sẽ có một số experience quan trọng cho việc training, và các experience này có thể xuất hiện ít hơn. Như Experience Replay bình thường, chúng ta sẽ chọn experience một cách random đến khi đủ 1 batch, điều đó làm cho việc các experience quan trọng được lựa chọn cho việc training càng khó hơn.

PER sử dụng **temporal-difference (TD) error** để tính toán độ ưu tiên cho từng experience để xác suất các experience quan trọng có thể được sử để huấn luyện cao hơn (**pure greedy prioritization)**, qua đó giúp quá trình huấn luyện diễn ra tốt hơn.

Chúng ta sẽ các experience mà trong đó TD error lớn thì sẽ có độ ưu tiên cao, điều đó chứng tỏ chúng ta cần học lại nhiều từ experience đó. Độ ưu tiên sẽ được tính như sau

$$p_t = |\delta_t| + e$$

$$\delta_t = r_{t+1} + \gamma Q(s_{t+1}, \operatorname*{argmax}_a Q(s_{t+1}, a; \boldsymbol{\theta}); \boldsymbol{\theta}^-) - Q(s_t, a_t; \boldsymbol{\theta})$$

Trong đó:

- $|\delta_t|$: độ lớn của TD error
- e: tham số để tránh priority bằng 0

Việc chỉ ưu tiên chọn các experience có TD error cao cho việc training mà ít để ý đến các experience khác dẫn đến việc mô hình có xu hướng overfitting khi chỉ có thể học từ các experience TD error cao. Để có thể giải quyết vấn đề này, chúng ta sử dụng một phương pháp lấy mẫu kết hợp **từ pure greedy prioritization** và **uniform random sampling**

$$P_i = \frac{p_i^a}{\sum_k p_k^a}$$

Trong đó:

- p_i: độ ưu tiên của experience
- a: tham số điều chỉnh chúng ta sẽ sử dụng bao nhiêu độ ưu tiên của experience
  - a = 0: chọn random uniform
  - a = 1: chỉ chọn những experience với độ ưu tiên cao nhất
  - a thường có giá trị trong khoảng [0.6, 0.7]

Với mỗi step trong quá trình huấn luyện, chúng ta sẽ lấy một batch gồm các experience với phân phối xác suất như trên và huấn luyện mô hình.

Quay lại với Normal Experience Replay, chúng ta lấy mẫu một cách ngẫu nhiên tuy nhiên khi sử dụng PER thì ta đã tạo ra một **bias** mà trong đó các experience có độ ưu tiên cao sẽ có xác suất được chọn cao hơn và làm giảm sự đa dạng của dữ liệu huấn luyện. Điều đó dẫn đến việc model không đi đúng đến điểm hội tụ. Để giải quyết vấn đề này, chúng ta sử dụng importance samping (IS) weights để điều chỉnh độ ảnh hưởng của của các experience

$$w_i = \left(\frac{1}{N} * \frac{1}{P(i)}\right)^b$$

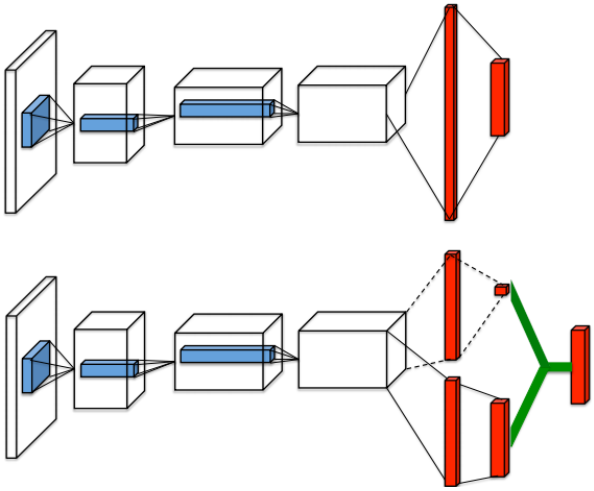
Trong đó:

- N: số experience có trong memory
- $P(i)$: xác suất của experience đã tính trước đó
- b: tham số điều chỉnh IS weight, b tăng dần dần trong quá trình. Tham số b sẽ điều khiển tầm quan trọng của IS weight ảnh hưởng lên quá trình training

3.4. DUELING DQN

Dueling DQN được giới thiệu lần đầu bởi nhóm nghiên cứu Google DeepMind, trong đó có **Tom Schaul** và **van Hasselt[3].** Thay vì sử dụng Q-network 1 luồng thì nhóm tác giả sử dụng Q-network 2 luồng để ước lượng giá trị V(s) cho mỗi trạng thái

và giá trị lợi ích A(s,a) cho mỗi cặp hành động – trạng thái. Sau đó 2 giá trị V(s) và A(s,a) được tổng hợp lại để ước lượng Q(s,a). Mục đích của cách làm trên là để Q-network có thể học được trạng thái nào có (hoặc không có) ý nghĩa trong việc huấn luyện mà không cần học sự ảnh hưởng của từng hành động lên một trạng thái.



Hình 5. Mạng Deep Q-network 1 luồng truyền thống (ở trên) và mạng Dueling Deep Q-network (ở dưới). Dueling DQN gồm 2 luồng, luồng ước tính giá trị trạng thái (đầu ra là 1) và giá trị lợi ích cho mỗi hành động trong trạng thái đó. Phần màu xanh là tổng hợp đầu ra 2 luồng. Cả 2 mô hình đều cho ra đầu ra là Q-value cho mỗi hành động.

Ý tưởng của Dueling DQN là trong một vài game Atari 2600, lấy ví dụ là game Enduro với các hành động là tập trung vào một vài điểm quan trọng trên hình ảnh trò chơi (xem hình 5), ở cặp hình ảnh phía trên luồng giá trị (Value) sẽ luôn tập trung vào bảng điểm số và một điểm trên con đường nơi có xe khác xuất hiện, luồng lợi ích sẽ không tập trung quá nhiều vào hình ảnh vì bất kì hành động nào trong trạng thái này cũng sẽ không ảnh hưởng đến điểm số. Ở cặp hình ảnh bên dưới, khi có xe ngay phía trước, thì luồng lợi ích sẽ tập trung ngay vào xe phía trước để đưa ra hành động phù hợp.

Nhóm tác giả định nghĩa mối liên hệ giữa A, V và Q bằng công thức sau

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s)$$

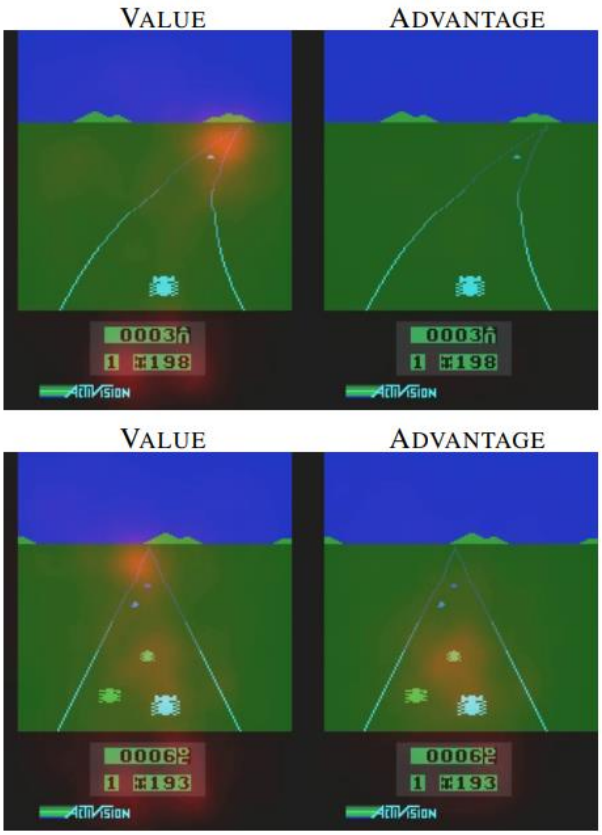
Trong đó, hàm giá trị V cho biết giá trị tốt hay không tốt của trạng thái s. Hàm Q cho biết giá trị của mỗi hành động được chọn trong trạng thái s. Từ đó, ta suy ra được công thức sau:

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + A(s,a;\theta,\alpha)$$

Trong đó:

- θ là trọng số của lớp rút trích đặc trưng
- α, β là trọng số của lớp Fully-connected.

Nhưng có một vấn đề khi sử dụng công thức trên là sẽ không xác định được khi cho Q để tính toán lại V và A. Điều này dẫn đến hiệu suất quá trình huấn luyện kém



Hình 6. Minh họa game Enduro.

Theo thực nghiệm của nhóm tác giả. Để giải quyết vấn đề này, nhóm tác giả sẽ ép cho hàm tính lợi ích A trả về giá trị 0 tại hành động được chọn trong trạng thái này, dẫn đến các giá trị lợi ích khác tại trạng thái này sẽ là âm.

Nhưng có một vấn đề khi sử dụng công thức trên là sẽ không xác định được khi cho Q để tính toán lại V

và A. Điều này dẫn đến hiệu suất quá trình huấn luyện kém theo thực nghiệm của nhóm tác giả. Để giải quyết vấn đề này, nhóm tác giả sẽ ép cho hàm tính lợi ích A trả về giá trị 0 tại hành động được chọn trong trạng thái này, dẫn đến các giá trị lợi ích khác tại trạng thái này sẽ là âm.

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + (A(s,a;\theta,\alpha) - max_{a'\in|\mathcal{A}|}A(s,a';\theta,\alpha))$$

Khi sử dụng công thức trên, khi chọn hành động tối ưu $a^* = argmax_{a'\epsilon\mathcal{A}}Q(s,a';\theta,\alpha,\beta) = argmax_{a'\epsilon\mathcal{A}}A(s,a';\theta,\alpha)$, chúng ta sẽ thu được $Q(s,a^*;\theta,\alpha,\beta) = V(s;\theta,\beta)$. Từ đó, xác định lại được các giá trị A(s,a) cho mỗi hành động trong trạng thái s.

Ngoài ra, nhóm tác giả đề xuất một công thức khác:

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + (A(s,a;\theta,\alpha) - \frac{1}{|\mathcal{A}|}\sum_{a'} A(s,a';\theta,\alpha))$$

Khi thực nghiệm, nhóm tác giả thấy công thức này tốt hơn công thức trước đó trong việc tăng sự ổn định trong quá trình huấn luyện, các giá trị lợi ích chỉ cần thay đổi với độ lớn là trung bình thay vì thay đổi với độ lớn của hành động tối ưu.

3.5. CATEGORICAL DQN

Categorical DQN được giới thiệu vào năm 2017 bởi Marc G. Bellemare và các cộng sự. Mục tiêu chính của thuật toán là học phân phối của phần thưởng nhận được trong tương lai (return) thay vì xấp xỉ giá trị kỳ vọng của chúng như các thuật toán trước đây. Với các môi trường phức tạp, phần thưởng có thể ngẫu nhiên, việc học giá trị kỳ vọng của phần thưởng bỏ qua các thông tin cần thiết (vd: phương sai của phân phối) để có thể chọn hành động tốt nhất.

Bài báo đề xuất sử dụng phân phối rời rạc gồm các support/ atom (giá trị rời rạc) biểu diễn phân phối giá trị Z. Với $N_{atoms} \in \mathbb{N}^+$ và $i \in \{1,\dots,N_{atoms}\}$, các atom được xác định như sau:

$$z_i = V_{min} + (i-1)\frac{(\mathrm{V}_{max} - \mathrm{V}_{\min})}{N-1}$$

Từ thực nghiệm thì nhóm tác giả nhận ra với $N = 51$ thì cho kết quả tốt nên Categorical DQN cũng được gọi là C51.

Gọi $Z(s,a)$ là phân phối giá trị (value distribution) phần thưởng nhận được khi xuất phát từ trạng thái s thực hiện hành động a sau đó tuân theo chiến thuật π.

$$Q(s,a) = \mathbb{E}[Z(s,a)] = \sum_{i=1}^{N} p_i z_i$$

Lúc này, ta có thể chuyển Bellman equation thành Distributional Bellman equation:

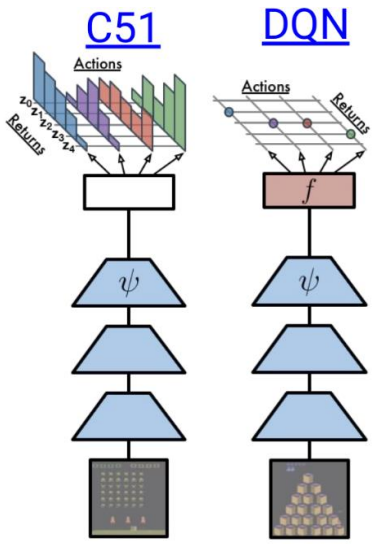
$$Z(s,a) \overset{D}{=} r + \gamma Z(s',a')$$

Thay vì cực tiểu hóa độ lỗi:

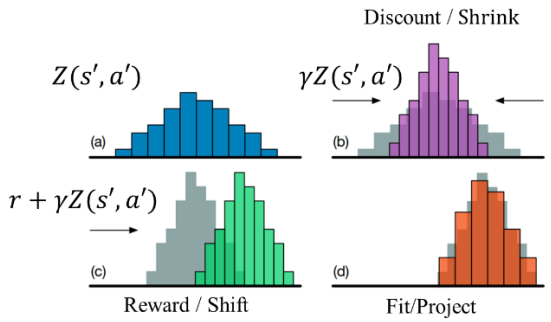
$$\mathbb{E}_{s,a,s'} = \left[\left(r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right)^2\right]$$

chúng ta sẽ cực tiểu hóa độ lỗi phân phối (distributional error) là khoảng cách giữa hai phân phối $r + \gamma Z(s',a')$ và $Z(s,a)$. Cũng như DQN, chúng ta vẫn dùng giá trị kỳ vọng để chọn hành động.

$$a^* = \arg\max_{a'} Q(s',a') = \arg\max_{a'} \mathbb{E}[Z(s',a')]$$



Hình 7. Minh họa so sánh C51 và DQN.



Hình 8. Minh họa quá trình cập nhật phân phối. (a) Phân phối của trạng thái kế tiếp, (b) Discount khiến phân phối thu hẹp lại về 0, (c) Điểm thưởng giúp phân phối dịch chuyển sang trái và (d) Bước chiếu (projection) phân phối Z' lên các support / atom của Z bằng cách cực tiểu hóa cross entropy loss giữa Z' và Z.

C51 được huấn luyện sử dụng phiên bản của CleanRL trên môi trường Kaggle với GPU P100. Các hyperparameter được cài đặt giống nhau ở hai lần chạy 200,000 và 500,000 step: learning rate = 2.5e-4; n_atom = 51; $v\_max = -v\_min = 10$; gamma = 0.99, buffer_size = 10000. Thời gian huấn luyện C51 sau 200,000 step mất khoảng 23 phút, sau 500,000 *step mất khoảng 58 phút.*

4. KẾT QUẢ THỰC NGHIỆM

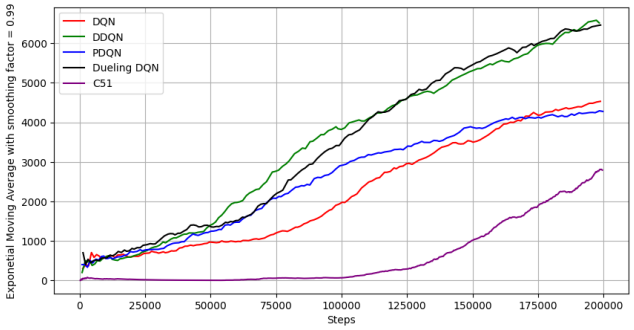
Kết quả thực nghiệm sẽ được thực hiện với 2 bộ hyperparameters chung nhất cho 4 thuật toán

Ý nghĩa các tham số được sử dụng:

- max_epsilon, min_epsilon: giá trị cao nhất và thấp nhất trong Epsilon Greedy Exploration
- epsilon_decay_intervals: tham số suy giảm của epsilon
- max_num_steps: số step dùng để training
- gamma: 0.99
- learning_rate: hệ số học
- memory_size: kích thước tối đa của bộ nhớ experience replay
- min_replay_size: thêm vào bộ nhớ số experience với giá trị tương ứng
- batch_size: số experience của từng batch

Bộ hyperparameter 1:

- max_epsilon: 1.0
- min_epsilon: 0.01
- max_num_steps: 200000
- epsilon_decay_intervals: 50000
- gamma: 0.99
- learning_rate: 5e-4
- memory_size: 20000
- min_replay_size: 1000
- batch_size: 32
- target_update_frequency: 1000

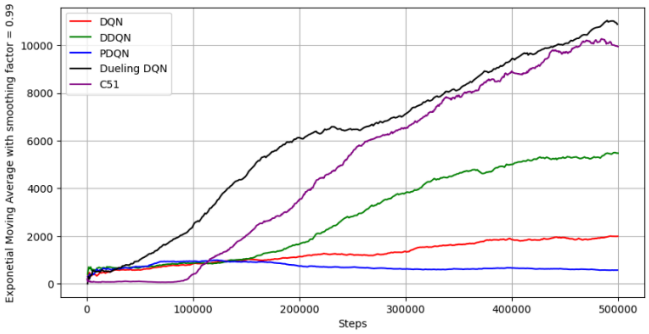


Hình 9. Kết quả quá trình huấn luyện các agent sau 200,000 step.

Có thể thấy khoảng từ step 50,000 đến 115,000 DDQN đạt điểm số cao nhất nhưng sau đó thì Dueling DQN có điểm số cao nhất. C51 có kết quả khá tệ khi phải sau 100,000 step thì agent mới có kết quả khả quan và tăng dần.

Bộ hyperparameters thứ hai:

- max_epsilon: 1.0
- min_epsilon: 0.01
- max_num_steps: 500000
- epsilon_decay_intervals: 200000
- gamma: 0.99
- learning_rate: 5e-4
- memory_size: 50000
- min_replay_size: 2000
- batch_size: 32
- target_update_frequency = 2000



Hình 10. Kết quả quá trình huấn luyện các agent sau 500,000 step.

Ở thử nghiệm 500,000 step, Dueling DQN đạt kết quả cao nhất so với 4 thuật toán còn lại, tuy kết quả có chững lại ở đoạn 200,000 đến 300,000. C51 đạt kết quả tốt ở thí nghiệm này và chỉ đứng sau Dueling DQN. PDQN có kết quả khá tệ khi kết quả không cải thiện sau 500,000 step.

5. TÀI LIỆU THAM KHẢO

[1] Tom Schaul, John Quan, Ioannis Antonoglou, David Silver. (2016) Prioritized Experience Replay. doi:10.48550/arXiv.1511.05952.

[2] Hado van Hasselt, Arthur Guez, David Silver. (2015) Deep Reinforcement Learning with Double Q-learning. doi:10.48550/arXiv.1509.06461

[3] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, Nando de Freitas (2016). Dueling Network Architectures for Deep Reinforcement Learning. doi:10.48550/arXiv.1511.06581

[4] Bellemare, M. G.; Dabney, W.; and Munos, R. 2017. A distributional perspective on reinforcement learning. doi.org/10.48550/arXiv.1707.06887

[5] van Hasselt, H., Guez, A. and Silver, D. (2015) Deep reinforcement learning with double Q-learning, arXiv.org. doi.org/10.48550/arXiv.1509.06461

[6] Lecture 6: CNNs and Deep Q Learning. 2019 https://web.stanford.edu/class/cs234/CS234Win2019/slides/lecture6.pdf