

**BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA KHOA HỌC MÁY TÍNH**



**ĐỒ ÁN MÔN HỌC
TRÍ TUỆ NHÂN TẠO – CS106.M11**

**REINFORCEMENT LEARNING CHO BÀI TOÁN
SUPER MARIO BROS**

GVHD: TS. Lương Ngọc Hoàng

STT	Họ tên	MSSV
1	Phạm Quang Vinh	19522526
2	Huỳnh Đỗ Tấn Thành	19522227
3	Nguyễn Minh Trí	19522389
4	Trương Xuân Linh	19521759

Tp HCM, tháng 12 năm 2021

I. Tổng quan bài toán:

1. Giới thiệu chung:

Trong đồ án này chúng em tìm hiểu cách ứng dụng Reinforcement Learning để thiết kế một agent tự động chơi tựa game Supper Mario Bros – một trong những tựa game làm nên tuổi thơ của thế hệ 9x-10x, là một thử thách trong RL với lượng không gian trạng thái khá lớn.



2. Môi trường:

Cũng vì tính phổ biến của game mà OpenAi Gym đã tích hợp Super Mario Bros và Super Mario Bros 2 với nền tảng của trình giả lập Nes-py - một trình giả lập các game trên hệ máy NES bằng python, ta có thể dễ dàng cài đặt và sử dụng bằng lệnh:

`pip install gym_super_mario`

Bao gồm 6 môi trường, với 4 phiên bản (v0, v1, v2, v3). Trong mỗi môi trường có 8 world, mỗi world có 4 stage. Cấu trúc:

`SuperMarioBros-<world>-<stage>-v<version>`

Environment	Game	ROM	Screenshot
SuperMarioBros-v0	SMB	standard	
SuperMarioBros-v1	SMB	downsample	
SuperMarioBros-v2	SMB	pixel	
SuperMarioBros2-v0	SMB2	standard	
SuperMarioBros2-v1	SMB2	downsample	

Hình 1: Các môi trường của Gym-super-mario-bros

Không gian trạng thái: mỗi trạng thái là 1 frame ảnh RGB kích thước (240x256x3)

Với việc sử dụng giả lập joypad của hệ máy NES với 6 nút bấm ta có 7 hành động cơ bản sau:

- **Noop** (không di chuyển),
- **left** (đi sang trái),
- **right** (đi sang phải),
- **up** (vào cửa + vào cổng trên),
- **down** (cuối người + vào cổng dưới),
- **A** (nhảy + bơi),
- **B** (tăng tốc + bắn lửa).

Từ 7 hành động này original Supper-mario-bros có tới 256 hành động khả thi. Tuy nhiên, trong Nes-py đã tích hợp những tập hành động sẵn và chia thành 3 loại:

- **RIGHT_ONLY:** gồm 5 hành động:
['NOOP'], ['right'], ['right', 'A'], ['right', 'B'], ['right', 'A', 'B'].
- **SIMPLE_MOVEMENT:** gồm 7 hành động:
['NOOP'], ['A'], ['right'], ['left'], ['right', 'A'], ['right', 'B'], ['right', 'A', 'B'],
- **COMPLEX_MOVEMENT:** gồm 12 hành động sau:
['NOOP'], ['A'], ['right'], ['left'], ['right', 'A'], ['left', 'A'], ['right', 'B'],
['left', 'B'], ['right', 'A', 'B'], ['left', 'A', 'B'], ['up'], ['down']

3. Tính điểm:

Với mục tiêu đi càng xa và càng nhanh, môi trường đã mô hình hóa hàm điểm thưởng với 3 biến sau:

- v: khoảng cách agent đi được sau mỗi step:

$$v = x1 - x0 \quad \left(\begin{array}{l} x0: \text{vị trí của agent trước khi đi step} \\ x1: \text{vị trí của agent sau khi đi step} \end{array} \right)$$
- c: thời gian đi 1 step:

$$c = c0 - c1 \quad \left(\begin{array}{l} c0: \text{thời gian còn lại trước khi đi step} \\ c1: \text{thời gian còn lại sau khi đi step} \end{array} \right)$$
- d: hình phạt khi chết:

$$d = 0 \Leftrightarrow \text{agent còn sống.}$$

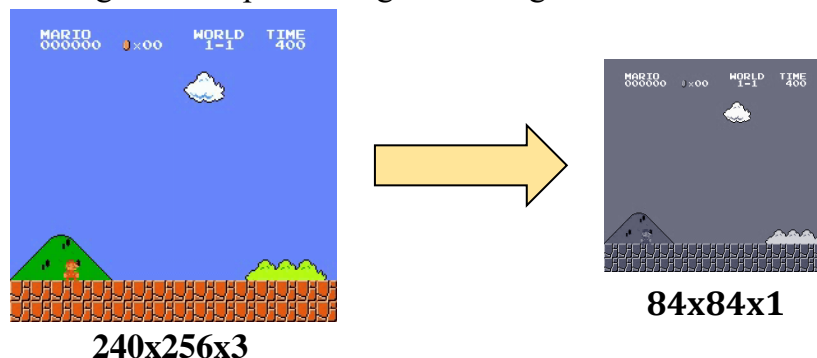
$$d = -15 \Leftrightarrow \text{agent chết.}$$

⇒ Hàm điểm thưởng: $r = v + c + d$

II. Tiền xử lý:

Để tăng tốc thời gian huấn luyện chúng em sử dụng hàm wrapper của OpenAi Gym để thực hiện một số tiền xử lý môi trường như sau:

- Cho agent thực hiện lặp lại 1 action trong 4 frame và xuất frame thứ 4. (vì trong 4 frame liên tiếp không khác biệt nhau quá nhiều)
- Giảm kích thước frame xuống 84x84x1 (chuyển về ảnh xám).
- Chuẩn hóa giá trị của pixel trong về khoảng từ 0 đến 1.



Hình 2: Tiền xử lý

III. Double Deep Q-Network (DDQN):**1. Q-learning với ε - greedy:**

Trước khi bắt đầu với DDQN, ta nhắc lại một vài điểm về Q-learning.

Chúng ta có công thức cập nhật bảng giá trị trạng thái từ công thức value iteration như sau:

$$Q_{k+1}(s, a) \leftarrow \sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma \max_{a'} Q_k(s', a') \right]$$

Tuy nhiên, để thực hiện công thức trên chúng ta cần có mô hình phân phối xác suất MDP. Nhưng trong bài toán này với không gian trạng thái quá lớn ta không có MDP, nên hướng đầu tiên để giải quyết là sử dụng các phương pháp học tăng cường mà điển hình là Q-learning.

Nhắc lại, về các bước thực hiện của Q-learning với ε - greedy:

1. Khởi tạo $Q(s, a) = 0$ cho mỗi cặp s, a và $t = 0$.
2. Thiết lập chiến lược $\pi_b(s) = \arg \max_{a'} Q(s, a)$ với xác suất $1 - \varepsilon$, else random.
3. Thực hiện vòng lặp:
 1. Chọn action a theo chiến lược π_b thu được trải nghiệm (s, a, r, s')
 2. Xét giá trị ước lượng $Q(s, a)$
 3. Sử dụng trải nghiệm thu được dựng công thức cho target

$$\text{sample} = r + \gamma \max_{a'} Q(s', a')$$
 4. Cập nhật $Q(s, a)$ dựa trên công thức sample

$$Q(s, a) \leftarrow Q(s, a) + \alpha (\text{sample} - Q(s, a)) = Q(s, a) + \alpha (r + \gamma \max_{a'} Q(s', a') - Q(s, a))$$

$$\pi_b(s) = \arg \max_{a'} Q(s, a) \text{ với xác suất } 1 - \varepsilon, \text{ else random.}$$
5. $s \leftarrow s'$

Đối với Q-learning, chính sách tối ưu của Agent sẽ luôn chọn những hành động có giá trị lớn nhất dù trong bất kỳ trạng thái nào. Tuy nhiên, Agent lúc đầu không hề biết gì về môi trường, nó cần ước tính Q-value lúc đầu và cập nhật chúng qua mỗi lần tương tác với môi trường. Q-value như vậy sẽ tồn tại rất nhiều nhiễu, những nhiễu này được coi là các nhiễu phân phối không đồng đều nên không thể chắc chắn được hành động có giá trị ước tính lớn nhất chọn ra có thực sự đã là hành động tốt nhất hay chưa. Khi đó, một vấn đề được phát sinh có tên là Overestimations, vấn đề này sẽ làm cho quá trình học trở nên vô cùng phức tạp và lộn xộn.

2. Double Q-Learning

Để giải quyết vấn đề Overestimations của Q-learning nên năm 2010 Hado Hasselt đã đề xuất ra Double Q-learning[]. Double Q-learning sẽ sử dụng hai hàm giá trị trạng thái là Q và Q', trong đó:

Hàm Q: được sử dụng để chọn ra action a có giá trị lớn nhất

$$a = \max_a Q(s_{t+1}, a)$$

Hàm Q': là hàm ước lượng giá trị hành động dựa trên hành động a đã chọn ở hàm Q

$$q_{\text{estimated}} = Q'(s_{t+1}, a)$$

Cuối cùng, ta sẽ cập nhật Q-value dựa vào hai hàm giá trị trạng thái đã đề cập ở trên là Q và Q' theo công thức:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(R_{t+1} + \gamma Q'(s_{t+1}, a) - Q(s_t, a_t))$$

So sánh giữa công thức giữa Q-learning và Double Q-learning, ta có thể thấy được sự khác biệt của công thức giữa hai thuật toán này. Trong khi Q-learning cập nhật giá trị Q-value cho nó dựa vào giá trị ước lượng được tính bởi chính nó thì Double Q-learning sẽ cập nhật Q-value dựa vào giá trị ước lượng được tính bởi Q'. Khi đó, kể cả khi Q và Q' bị nhiễu thì những nhiễu này có thể xem được là các nhiễu có phân phối đồng đều và chúng sẽ không ảnh hưởng tới sự khác biệt giữa Q và Q'.

Dưới đây là đoạn mã giả mô tả tổng quan thuật toán Double Q-learning

```

1: Initialize  $Q^A, Q^B, s$ 
2: repeat
3:   Choose  $a$ , based on  $Q^A(s, \cdot)$  and  $Q^B(s, \cdot)$ , observe  $r, s'$ 
4:   Choose (e.g. random) either UPDATE(A) or UPDATE(B)
5:   if UPDATE(A) then
6:     Define  $a^* = \arg \max_a Q^A(s', a)$ 
7:      $Q^A(s, a) \leftarrow Q^A(s, a) + \alpha(s, a)(r + \gamma Q^B(s', a^*) - Q^A(s, a))$ 
8:   else if UPDATE(B) then
9:     Define  $b^* = \arg \max_a Q^B(s', a)$ 
10:     $Q^B(s, a) \leftarrow Q^B(s, a) + \alpha(s, a)(r + \gamma Q^A(s', b^*) - Q^B(s, a))$ 
11:  end if
12:   $s \leftarrow s'$ 
13: until end

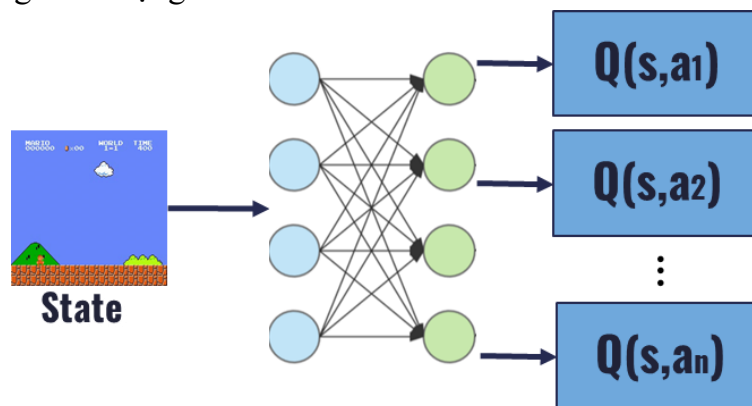
```

Dựa vào đoạn mã giả, có thể dễ dàng nhận thấy một điểm rất đặc biệt và đặc trưng của Double Q-learning, đó là thuật toán sẽ chọn ngẫu nhiên UPDATE(A) hoặc là UPDATE(B). Có nghĩa là vai trò của Q^A và Q^B sẽ được hoán đổi cho nhau trong quá trình học tập. Có khi Q^A sẽ được sử dụng để chọn ra hành động có giá trị ước lượng lớn nhất, cũng có khi Q^A sẽ là hàm ước lượng giá trị hành động có giá trị ước lượng lớn nhất được lấy ra từ Q^B và ngược lại.

Cả hai thuật toán là Q-learning và Double Q-learning đều sử dụng bảng để lưu trữ các giá trị trạng thái và hành động của Agent, bảng này được gọi là Q-table. Tuy nhiên, khi số lượng hành động và trạng thái quá lớn, việc lưu trữ vào bảng dường như là một điều không thể vì không đủ bộ nhớ lưu trữ. Vì vậy năm 2016 Hado Hasselt đã đề xuất ra việc sử dụng mạng Neural Network thay thế cho Q-table trong thuật toán Double Q-learning.

3. Double Deep Q-Network

Có nhiều cách để thiết kế mạng DDQN, trong đó có một cách sẽ có lợi hơn trong việc lập trình. Theo đó, ta sẽ truyền vào mạng một trạng thái, sau khi rút trích đặc trưng qua mạng, output sẽ cho biết tại trạng thái đó có bao nhiêu hành động có thể thực hiện và giá trị của từng hành động đó là bao nhiêu.



Hình 3: Cách thiết kế mạng Double Deep Q-Network

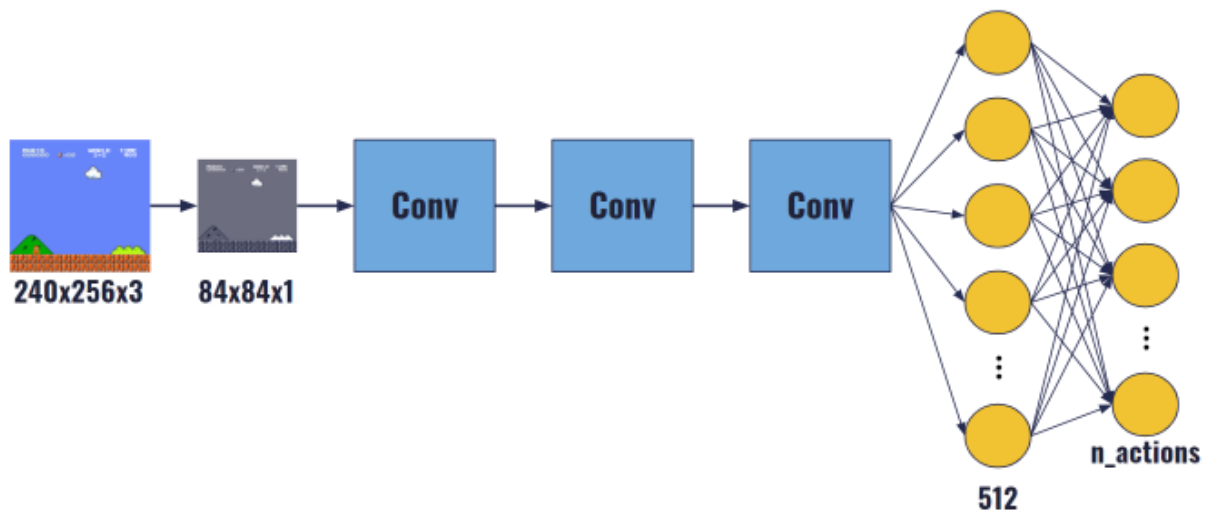
Bên cạnh sự khác biệt giữa Double Q-learning và Double Deep Q-Network là việc sử dụng mạng Neural Network thay thế cho bảng Q-table thì còn có một khác biệt khác ở công thức giữa hai thuật toán này là việc Double Q-learning sử dụng learning rate alpha khi cập nhật các giá trị Q-value còn Double Deep Q-Network sẽ không sử dụng learning rate alpha để cập nhật Q-value mà sử dụng nó trong giải đoạn tối ưu các tham số của mạng Neural Network. Hay nói một cách khác, mọi công thức của Double Deep Q-Network được lấy từ Double Q-learning với tham số học learning rate alpha được đặt cố định bằng 1.

Dưới đây là công thức cập nhật của Double Deep Q-Network

$$a = \max_a Q_{\text{qnet}}(s_{t+1}, a)$$

$$q_{\text{estimated}} = Q_{\text{tnet}}(s_{t+1}, a)$$

$$Q_{\text{qnet}}(s_t, a_t) \leftarrow R_{t+1} + \gamma Q_{\text{tnet}}(s_{t+1}, a)$$



Hình 4: Mô hình mạng sử dụng

Bức ảnh đầu vào của game có kích thước 240x256x3, sau đó được chuyển về ảnh bức xám và resize lại kích thước 84x84x1 và chuẩn hóa các giá trị pixel về giá trị từ 0 đến 1. Sau đó, nó được đưa qua 3 lớp Convolutional để rút trích đặc trưng và đưa qua một lớp hidden layer với số units là 512. Cuối cùng, là một lớp output với số unit bằng số lượng hành động có thể thực hiện của Agent.

Bên cạnh đó, một số tham số và thuật toán được sử dụng xuyên suốt trong quá trình học tập như:

- Optimizer: Adam
- Epsilon_max: 1
- Epsilon_min: 0.02
- Epsilon_decay: 0.999
- Batch_size: 32
- gamma: 0.9
- lr: 0.00025

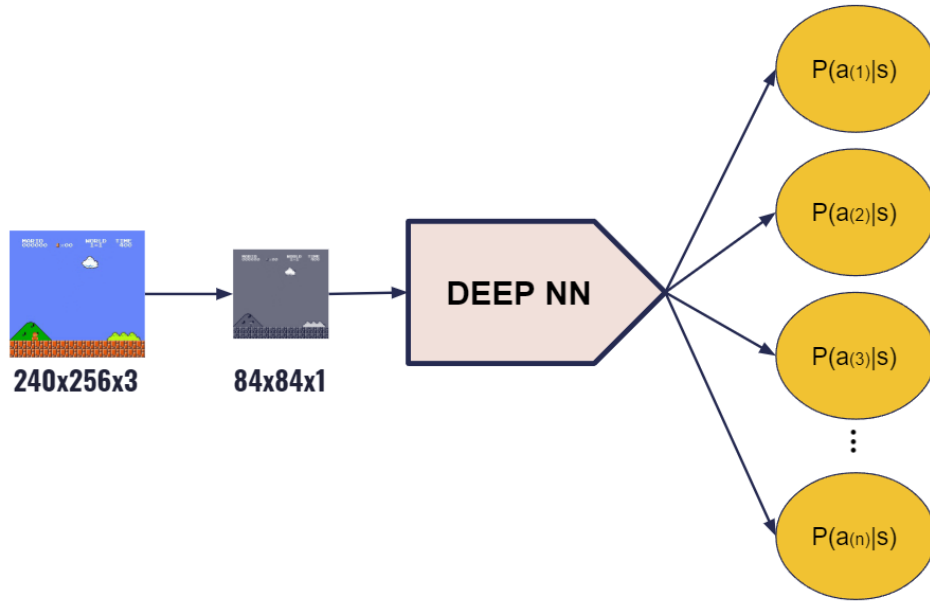
Và công thức cập nhật epsilon là:

$$\text{Epsilon_max} = \text{Epsilon_max} * \text{Epsilon_decay}$$

$$\text{Epsilon} = \max(\text{Epsilon_min}, \text{Epsilon_max})$$

IV. Proximal Policy Optimization

1. Mạng học sâu:



Hình : Mô hình PPO

Tương tự với DDQN, các bước tiền xử lý một frame cũng là SkipFrame với 4 frame. Sau khi tiền xử lý ta có được một frame hình 84x84x1, đưa frame đó qua một mạng học sâu tương tự như ở DDQN. Output của mạng học sâu sẽ là xác suất thực hiện hành động $a(i)$ ở trạng thái s và tổng các output bằng 1: $\sum_i P(a_i|s) = 1$

2. Hàm Custom Reward:

Ngoài reward được cung cấp từ gym-super-mario-bros, chúng em kết hợp thêm điểm tích lũy của Mario khi ăn những đồng coin và tiêu diệt quái vật, thêm vào đó là phần thưởng +50 nếu chiến thắng và -50 nếu thua cuộc.

$$R_t = \frac{(r_t + score_t)}{40}$$

if info["flag_get"]:

$R_t += 50$

else: $R_t -= 50$

3. Giá trị lợi thế kỳ vọng (Generalized Advantage Estimation - GAE):

$$\hat{A}_t = \delta_t + (\tau\gamma)\delta_{t+1} + \dots + (\tau\gamma)^{T-t+1}\delta_{T-1}$$

where $\delta_t = r_t + \tau V(s_{t+1}) - V(s_t)$

Với: r : giá trị điểm thưởng

γ, λ : siêu tham số.

V : giá trị trạng thái.

Ở bài toán này chúng em sử dụng $\gamma = 1.0$ và $\lambda = 0.9$.

4. Giá trị gradient trong bài toán Policy Gradient:

$$\hat{g} = \hat{E}_t[\nabla_t \log \pi_t(a_t|s_t) \hat{A}_t]$$

Với \hat{A} : giá trị lợi thế kỳ vọng.

π : chiến lược.

a : hành động.

s : trạng thái.

Ở bài toán Policy Gradient, giá trị gradient cũng tương đương với giá trị loss trong bài toán Supervised learning nhưng nhiệm vụ của chúng ta ở đây sẽ là cực đại hoá giá trị gradient này. Nếu giá trị lợi thế kỳ vọng mang giá trị dương hay còn gọi là lợi thế tích cực, điều đó dẫn đến gradient sẽ mang giá trị dương và hành động a là hành động tốt tại trạng thái s , vì vậy sẽ tăng xác suất thực hiện hành động a tại trạng thái s để có được chiến lược tối ưu hơn. Ngược lại thì sẽ giảm xác suất thực hiện hành động a nếu a là một hành động xấu tại trạng thái s .

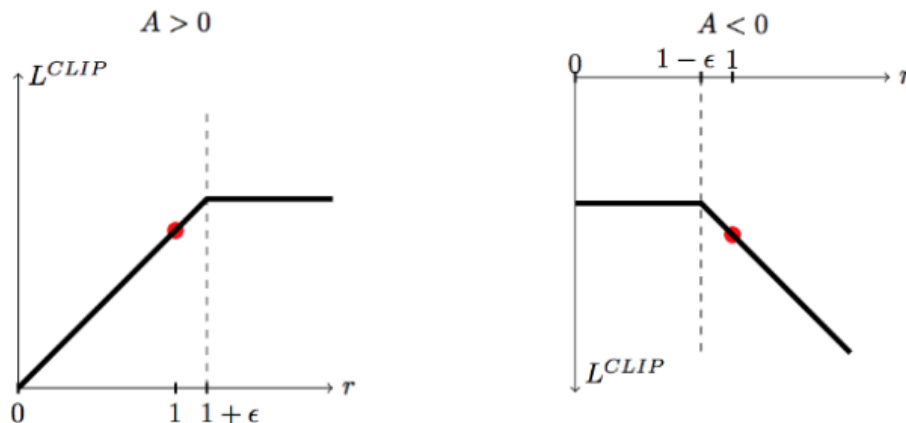
Tuy nhiên vấn đề của thuật toán Policy Gradient là không kiểm soát được mức độ cập nhật chiến lược mới, mức độ hội tụ của thuật toán có thể chậm do thực hiện các bước cập nhật quá lớn hoặc quá nhỏ. Vì vậy cần một ràng buộc hay một vùng tin cậy để giảm bớt mức độ cập nhật, nếu cập nhật quá nhiều thì cần thu hẹp vùng tin cậy lại.

5. Vùng tin cậy trong thuật toán PPO:

$$r(\theta) = \frac{\pi_{\theta}(a|s)}{\pi_{\theta_{old}}(a|s)} \quad \text{Với } r: \text{ tỉ lệ sai lệch giữa chiến lược mới và chiến lược cũ}$$

$$L^{CLIP}(\theta) = \hat{E}_t[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon)\hat{A}_t)] \quad \text{Với } \epsilon: \text{ siêu tham số.}$$

Sau khi có được tỉ lệ sai lệch, ta tiến hành tính loss CLIP qua công thức trên. Ở phần cắt giảm (clip), giá trị r được ràng buộc trong khoảng $[1-\epsilon, 1+\epsilon]$.



Biểu đồ 1: Biểu đồ thể hiện sự cắt giảm giá trị lợi thế kỳ vọng

Ở biểu đồ bên trái, nếu r có xu hướng tăng và vượt qua ngưỡng thì sẽ bắt đầu cắt giảm \hat{A} để các bước cập nhật không đi quá xa so với chiến lược cũ. Còn ở bên phải, nếu r có xu hướng giảm và thấp hơn ngưỡng thì sẽ giữ lại các chiến lược cũ vì chiến lược cũ tốt hơn.

6. Thuật toán PPO:

Mã giả:

```

Initialize  $\theta$ 
for i = 1, 2... do
  for j = 1, 2..., N do
    Run policy  $\pi_{\theta_{old}}$  in environment for T timesteps
    Compute advantage estimates  $\hat{A}$ 
  end for
  Optimize surrogate L wrt  $\theta$ , with K epochs and minibatch size  $M \leq NT$ 
   $\theta_{old} \leftarrow \theta$ 
end for

```

V. Nhận xét & đánh giá:**1. Cấu hình:**

Để so sánh 2 thuật toán trong việc huấn luyện mario thì chúng em huấn luyện trên cùng một cấu hình:

- GPU: NVIDIA GTX 1650.
- Ram: 8GB
- Thời gian huấn luyện tối đa: 20 tiếng (vì thời gian huấn luyện có hạn).
- Action: SIMPLE_MOVEMENT (SIMPLE_MOVEMENT chứa 7 tập hành động cần thiết giúp tối ưu cả về khả năng qua màn và thời gian huấn luyện).
- Thực hiện huấn luyện trên môi trường Super-mario-bros-v0 (môi trường cơ bản của tựa game mario).
- Thực hiện huấn luyện trên toàn bộ các stage của các world từ 1-1 đến 8-4

2. Thống kê kết quả:

Kết quả thống kê của của hai thuật toán được thể hiện dưới bảng bên dưới.

	Stage 1	Stage 2	Stage 3	Stage 4
World 1	WIN	WIN	WIN	WIN
World 2	WIN	WIN	WIN	WIN
World 3	WIN	WIN	WIN	WIN
World 4	WIN	WIN	WIN	WIN
World 5	WIN	WIN	WIN	WIN
World 6	WIN	WIN	WIN	WIN
World 7	WIN	WIN	WIN	WIN
World 8	WIN	WIN	WIN	WIN

DOUBLE DEEP Q-NETWORK

WIN : WIN KHÔNG WIN : KHÔNG WIN

	Stage 1	Stage 2	Stage 3	Stage 4
World 1	WIN	WIN	WIN	WIN
World 2	WIN	WIN	WIN	WIN
World 3	WIN	WIN	WIN	WIN
World 4	WIN	WIN	WIN	WIN
World 5	WIN	WIN	WIN	WIN
World 6	WIN	WIN	WIN	WIN
World 7	WIN	WIN	WIN	WIN
World 8	WIN	WIN	WIN	WIN

PROXIMAL POLICY OPTIMIZATION*Bảng 1: Thống kê kết quả Agent chơi thử nghiệm*

Kết quả cho thấy thuật toán PPO cho kết quả hơn vượt trội với DDQN.

3. Nhận xét:

- Với thời gian huấn luyện tối đa cho 1 màn bằng nhau và cấu hình ngang nhau, ta nhận thấy thuật toán PPO cho kết quả tốt hơn.
- Đối với DDQN: Input là một bức ảnh nên khó nhận biết được các con quái hay cột lửa di chuyển về hướng nào và khó nhận ra được hành động xấu để tránh.
- Đối với PPO:
 - Đã cải thiện điểm yếu của DDQN, việc huấn luyện của PPO dựa vào một chuỗi các hành động nên có thể đoán được hướng di chuyển của đối thủ.
 - Nhưng có một số màn có các vị trí làm cho mario trở lại điểm trước đó khiến mario rơi vào vòng lặp (màn 4-4 và màn 7-4).
 - Màn 8-4 là màn khá khó nên cả 2 thuật toán đều không qua được.
- Cả hai thuật toán đều không thể huấn luyện trên một màn và thử nghiệm trên màn khác hay phiên bản khác.

4. Đánh giá:

- PPO là một thuật toán hiện đại, cho kết quả tốt nhất trong khoảng thời gian huấn luyện bằng nhau.
- Thời gian huấn luyện của PPO nhanh hơn (Đối với màn 1-1, DDQN mất hơn 15 tiếng và PPO chỉ mất hơn 7 tiếng).
- Có thể quan sát thấy rằng PPO cung cấp tỷ lệ hội tụ và hiệu suất tốt hơn còn DDQN thì không ổn định và cho độ hội tụ kém.

5. Khó khăn:

- Tinh chỉnh các siêu tham số khó khăn và phải thực hiện nhiều lần.
- Tốn nhiều tài nguyên cho việc huấn luyện.
- Thời gian huấn luyện khá lâu (nhóm chúng em mất 2 tháng cho quá trình huấn luyện và điều chỉnh tham số).

VI. Tài liệu tham khảo.

[1] Gym-super-mario-bros github repo

Link: <https://github.com/Kautenja/gym-super-mario-bros>

[2] Playing Super Mario Bros with Deep Reinforcement Learning

Link: <https://www.analyticsvidhya.com/blog/2021/06/playing-super-mario-bros-with-deep-reinforcement-learning/>

[3] Deep q learning part 2: Double Deep Q Network (Double DQN)

Link: <https://medium.com/@qempsil0914/deep-q-learning-part2-double-deep-q-network-double-dqn-b8fc9212bbb2>

[4] Double Q-learning

Link: <https://papers.nips.cc/paper/2010/file/091d584fced301b442654dd8c23b3fc9-Paper.pdf>

[5] uvipen's repo – Super-mario-bros-PPO-pytorch

Link: <https://github.com/uvipen/Super-mario-bros-PPO-pytorch>

[6] Proximal policy optimization algorithms

Link: <https://arxiv.org/pdf/1707.06347.pdf>