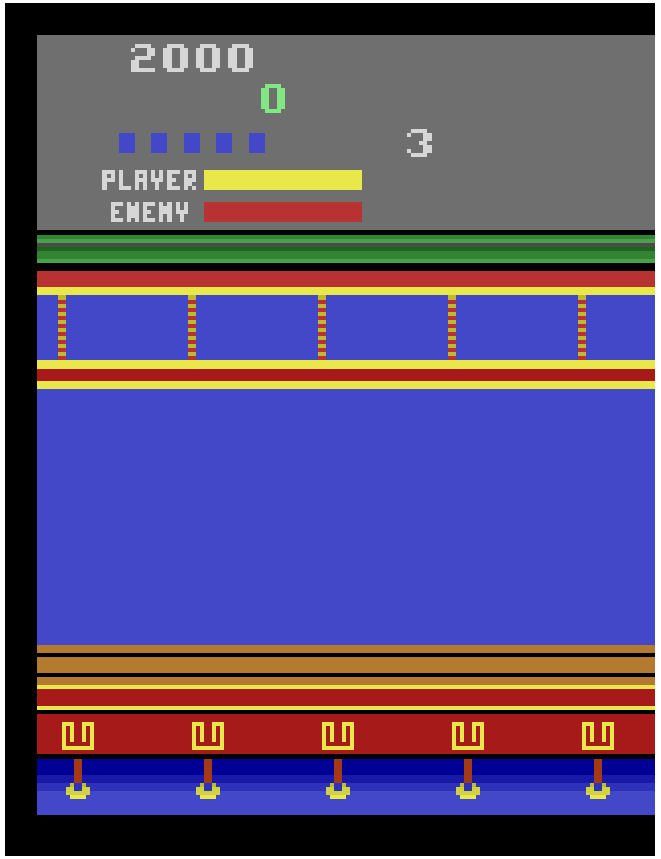
USING REINFORCEMENT LEARNING TO PLAY KUNG FU MASTER

Instructor: PhD Luong Ngoc Hoang

Phạm Trần Anh Tiên - 20522012
Lương Lý Công Thịnh - 20521960
Đào Trần Anh Tuấn - 20522107
Trần Phú Vinh - 20522161

Kung-Fu Master

Với game này bạn sẽ nhập vai là một võ sư Kung-Fu đang chiến đấu xuyên qua ngôi đền của Pháp sư ác độc để cứu công chúa Victoria, trên đường đi bạn phải đánh bại các kẻ thù khác nhau.



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

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 trạng thái là một ảnh RGB có dạng (250, 160, 3)

Tiền xử lý

- Ảnh màn hình trò chơi KungFu Master trước khi đưa vào mạng học sâu của các agent đều được xử lý như sau:
  - Chuyển thành ảnh xám.
  - Resize về kích thước (84, 84)

Trích xuất đặc trưng như thế nào?

Sử dụng các lớp conv2d kết hợp với lớp relu và pooling như hình để trích xuất đặc trưng

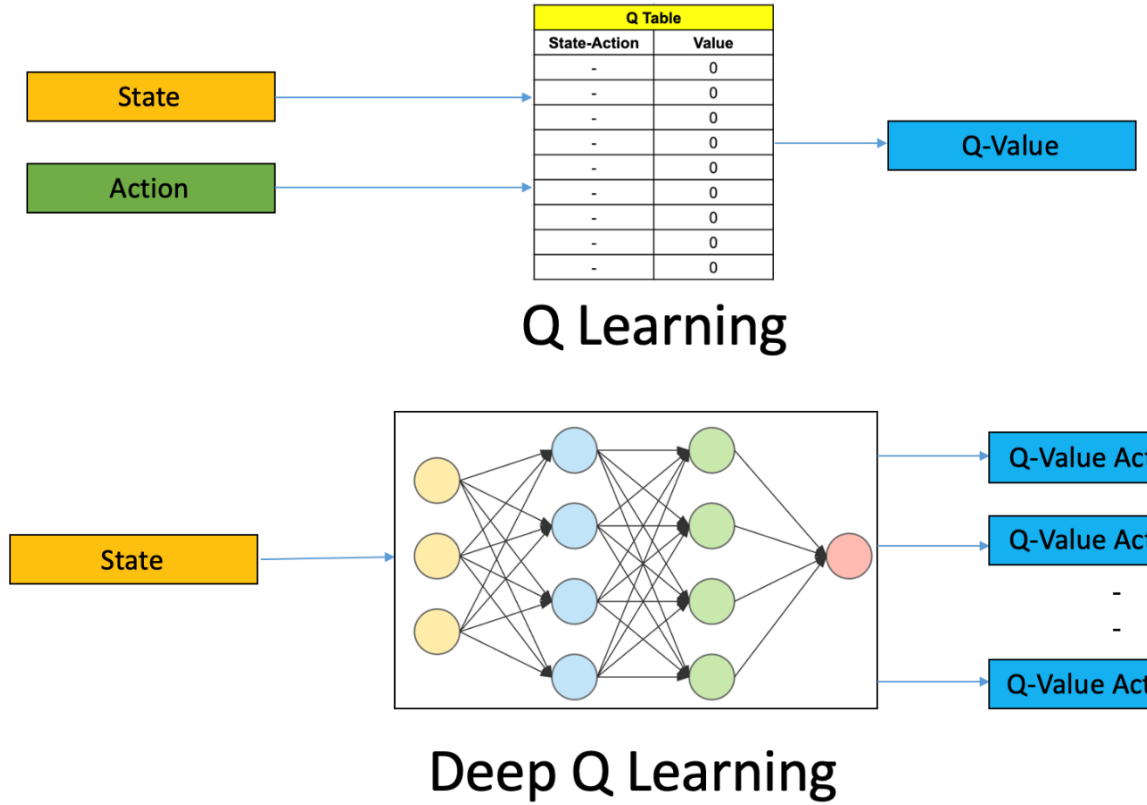
```
# Extraction
nn.Conv2d(in_channels=1, out_channels=16, kernel_size=3, stride=1),
nn.ReLU(),
nn.MaxPool2d(kernel_size=2, stride=2),

nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3, stride=1),
nn.ReLU(),
nn.MaxPool2d(kernel_size=2, stride=2),
```

Deep Q Network

Deep Q Network

- DQN là một cải tiến từ Q-learning.
- DQN sử dụng một Neural Network để thay thế cho Q-Table
- DQN nhận vào một trạng thái s và output là một vector giá trị action-value Q(s, ·, $\boldsymbol{\theta}$)



Deep Q Network – Fix Q-Targets

- DQN có ý tưởng sử dụng hai mạng neural network nhằm tránh sự tương tác giữa các hành động và các giá trị ước lượng của chúng:
  - **Q-network** để thực hiện việc chọn các hành động (actions).
  - **Target-network** chỉ được dùng để ước lượng giá trị của các hành động.
- **Target-network** có tham số $\boldsymbol{\theta}^{-}$ như Q-network và $\boldsymbol{\theta}^{-}$ được cập nhật sau mỗi τ steps

=> Sử dụng **Target-network** giúp cho mô hình học ổn định hơn và dễ dàng tiến đến điểm hội tụ

Deep Q Network – Experience replay

- DQN có vấn đề là **catastrophic forgetting,** hiện tượng mô hình quên mất các thông tin đã được huấn luyện trước đó.

$\Rightarrow$ Sử dụng một kỹ thuật là **experience replay**

- Mỗi step sẽ có 1 tuple **(s, a, r, s', done)** được lưu vào **experience replay**
- Khi training ta sẽ lấy random một minibatch từ **experience replay** để training mô hình

Deep Q Network

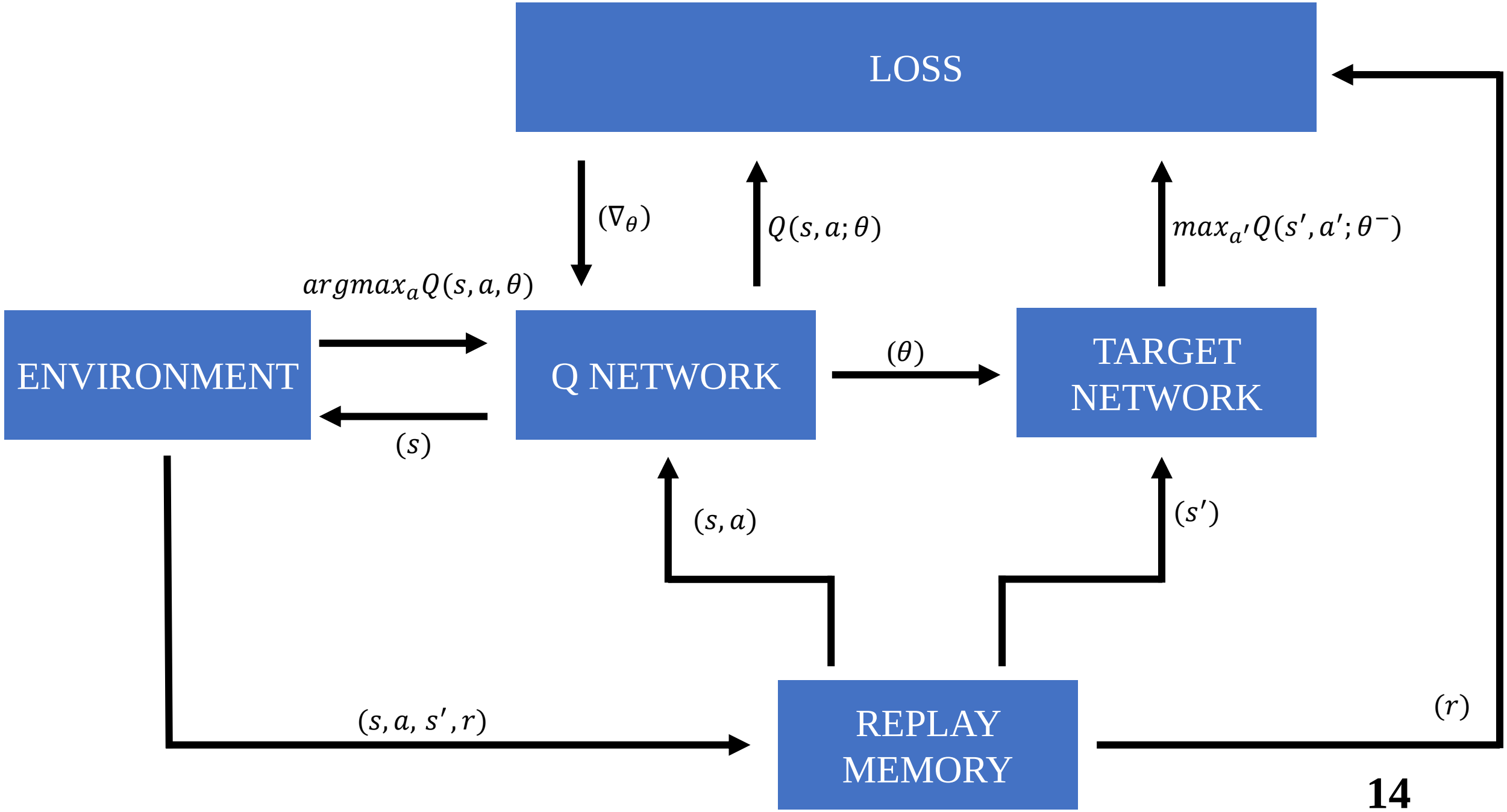
- Tham số mô hình Q-Network sẽ được cập nhật như sau

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

Với $Y_t = R_{t+1} + \gamma \max_{a} Q(S_{t+1},\ a;\ \boldsymbol{\theta_t^-})$

Deep Q Network – Summary

- DQN sử dụng Fix-Q Targets và Experience Replay
- Lưu trữ **(s, a, r, s', done)** trong replay memory
- Lấy mẫu bằng cách random một minibatch **(s, a, r, s', done)** từ **replay** memory để training mô hình
- Tối ưu MSE giữa Q-network và Target-network
- Sử dụng stochastic gradient descent để cập nhật trọng số cho các mạng


LOSS
(∇_θ)
$Q(s,a;\theta)$
$max_{a'}Q(s',a';\theta^-)$
$argmax_a Q(s,a,\theta)$
ENVIRONMENT
Q NETWORK
(θ)
TARGET NETWORK
(s)
(s,a)
(s')
(s,a,s',r)
REPLAY MEMORY
(r)

DOUBLE DQN

Double Deep Q Network

- Double DQN là một phương pháp cải tiến của DQN nhằm khắc phục vấn đề **overestimation** bởi vì vấn đề này dẫn đến xu hướng tối đa hóa trong khi huấn luyện.
- DQN do việc sử dụng hàm **max** tính **target** để cập nhật tham số nên thường bị tình trạng **overestimation**
  - mô hình xấp xỉ điểm số (reward) của một trạng thái (state) quá cao so với điểm số thực

$$Y_t = R_{t+1} + \gamma \max_{a} Q(S_{t+1},\ a;\ \boldsymbol{\theta_t^-})$$

Double Deep Q Network

- DDQN cố gắng giải quyết vấn đề đó bằng cách sử dụng **argmax** kết quả đầu ra của Q-Network để chọn action thay vì chọn action có giá trị cao nhất từ Target-Network như DQN
  - **DDQN**: $Y_t = R_{t+1} + \gamma Q(s'_{t+1}, \underset{a}{\text{argmax}}\, Q(S_{t+1}, a; \boldsymbol{\theta_t}), \boldsymbol{\theta_t^-})$
  - **DQN**: $Y_t = R_{t+1} + \gamma \max_{a} Q(S_{t+1}, a; \boldsymbol{\theta_t^-})$

PRIORITIZED DQN

- **Normal Experience Replay**:
  - Trong các experience được lưu, có một số experience quan trọng cho việc training, và tỉ lệ các experience này được chọn để training không cao

- **Prioritized Experience Replay (PER)** được giới thiệu đầu tiên vào năm 2015 bởi **Tom Schaul**.

- **PER** sử dụng **TD error** để tính độ ưu tiên cho từng experience để các experience quan trọng có thể được sử để huấn luyện

- experience có **TD error** lớn thì sẽ có độ ưu tiên cao,

=> chứng tỏ chúng ta cần học lại nhiều từ experience đó.

- Độ ưu tiên sẽ được tính như sau

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

- Trong đó:
  - $|\delta_t|$: độ lớn của TD error

$$\delta_t = r_{t+1} + \gamma Q(s_{t+1}, \underset{a}{\text{argmax}}\, Q(s_{t+1}, a), \theta^-) - Q(s_t, a_t\theta)$$

  - e: tham số để tránh priority bằng 0

- Các experience được lấy để training sẽ được cập nhật lại priority

- **Problem:**
  - experience có TD error thấp có thể sẽ không được replay
  - experience có TD error cao được replay thường xuyên.

$\Rightarrow$ overfitting khi chỉ có thể học từ các experience này

- **Solution**: sử dụng một phương pháp lấy mẫu kết hợp **từ pure greedy prioritization** và **uniform random sampling**

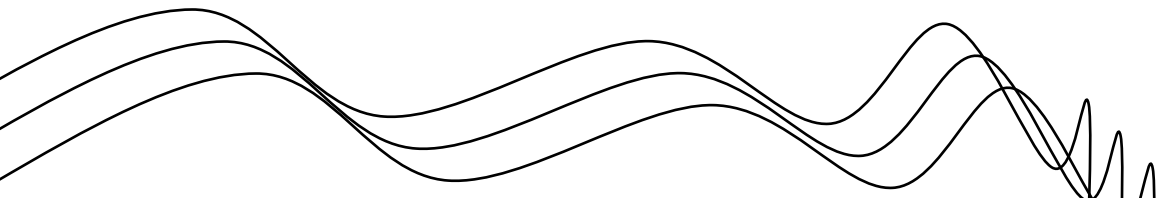
$$P_i = \frac{p_i^a}{\sum_k p_k^a}$$

- Trong đó:
  - p_i: độ ưu tiên của experience
  - a: tham số điều chỉnh mức độ sử dụng độ ưu tiên của experience
    - a = 0: chọn random uniform
    - a = 1: chỉ chọn những experience với độ ưu tiên cao nhất

- Lấy một batch gồm các experience với một phân phối xác suất và huấn luyện model.

- **Problem**: khi sử dụng PER => tạo ra một **bias** khiến experience có độ ưu tiên cao sẽ có xác suất được chọn cao hơn.

  => Điều đó dẫn đến việc model không đi đúng đến điểm hội tụ.



- **Solution**: sử dụng **importance samping (IS) weights** để điều chỉnh độ ảnh hưởng của của các experience

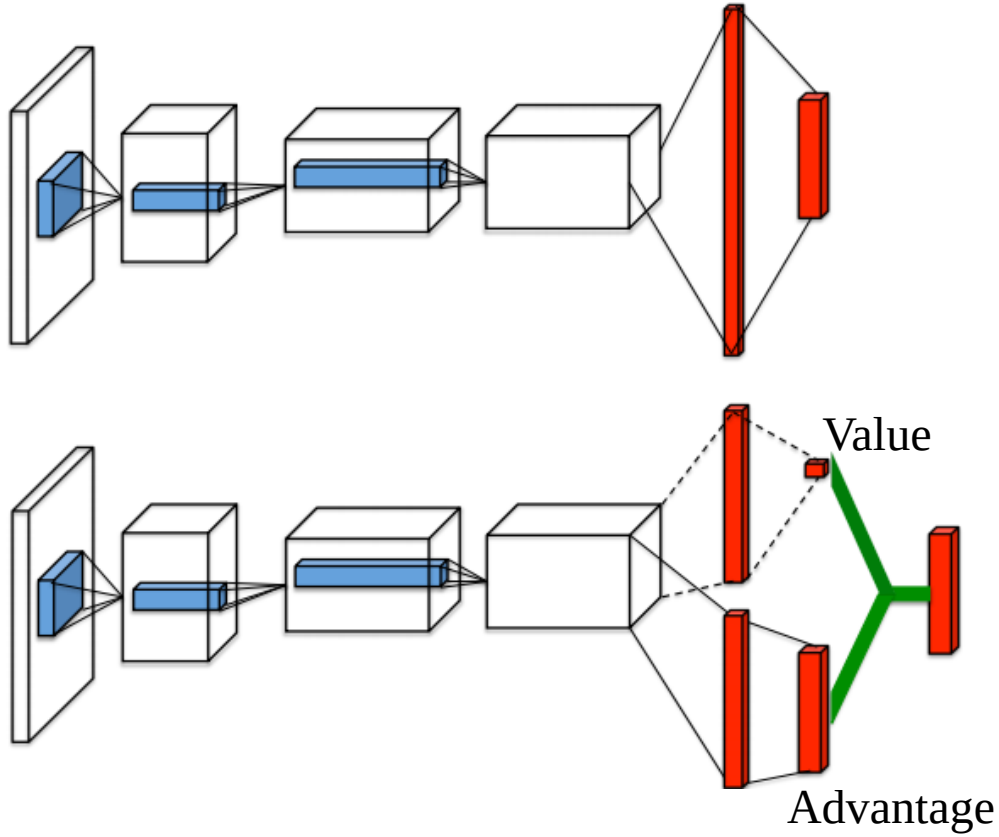
$$w_i = \left(\frac{1}{N} * \frac{1}{P(i)}\right)^b$$

- Trong đó:
  - N: số experience có trong memory
  - $P(i)$: xác suất của experience
  - b: tham số để điều chỉnh **IS weight**, b sẽ tăng lên dần khi training.

DUELING DQN

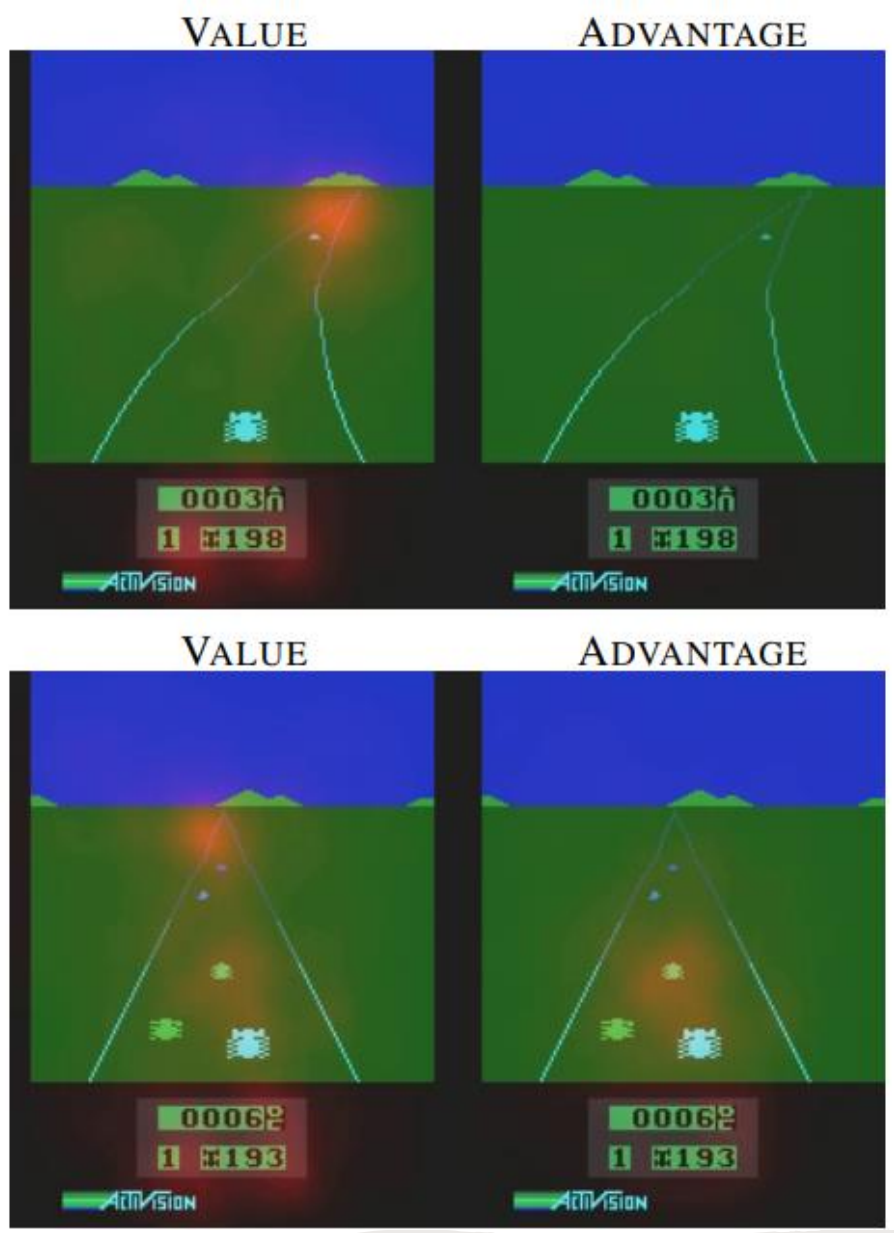
Dueling DQN

- Dueling DQN được giới thiệu lần đầu bởi nhóm nghiên cứu Google DeepMind, trong đó có **Tom Schaul** và **van Hasselt.**
- 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.



Dueling DQN

- Ý tưởng cho 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.



Dueling DQN

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

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

Dueling DQN

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

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

- Từ đó, suy ra:

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

- Trong đó:

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

Dueling DQN

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

- Vấn đề: không xác định Q(s,a1) = 100 thì có vô số cặp V, A(s,a1) thoả mãn.
- Giải quyết:

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

$$\text{Với } max_{a' \in |\mathcal{A}|} A(s,a';\theta,\alpha) = 0$$

- Giả sử: Q(s,a1) = 100, Q(s,a2) = 200.
  - Tại a^* = a2 => V = Q(s,a2) = 200
  - Từ đó suy ra: A(s,a2) = 0, A(s,a1) = -100. Xác định được các cặp A và V tương ứng.

Dueling DQN

- Ngoài ra, nhóm tác giả còn đề xuất:

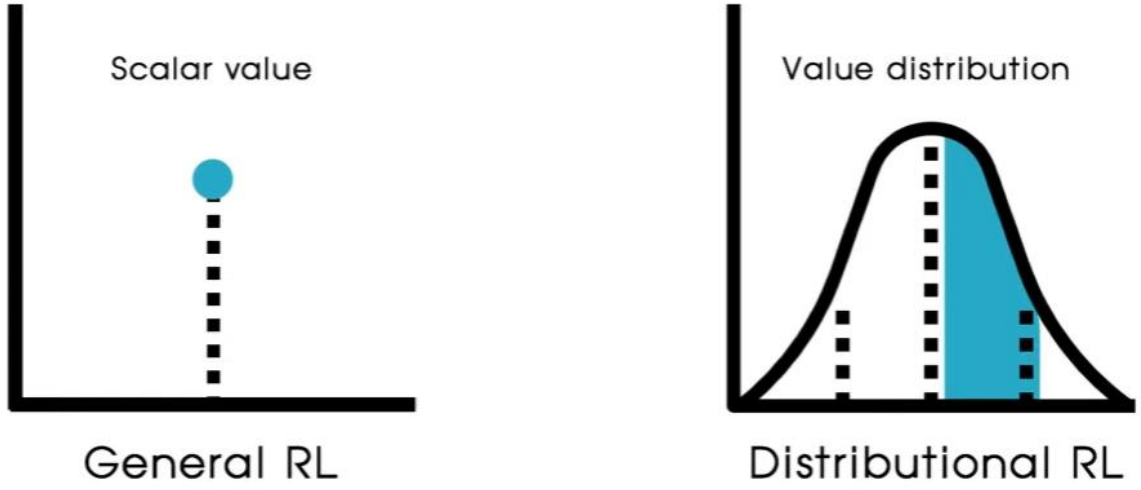
$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + \left(A(s,a;\theta,\alpha) - \frac{1}{|\mathcal{A}|}\sum_{a'} A(s,a';\theta,\alpha)\right)$$

- Công thức này tốt hơn công thức trước đó trong việc tăng sự ổn định trong quá trình huấn luyện.

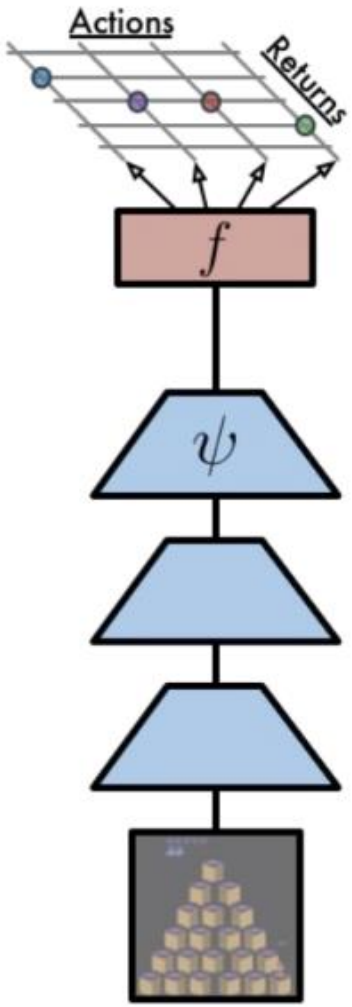
CATEGORICAL DQN

Categorical DQN

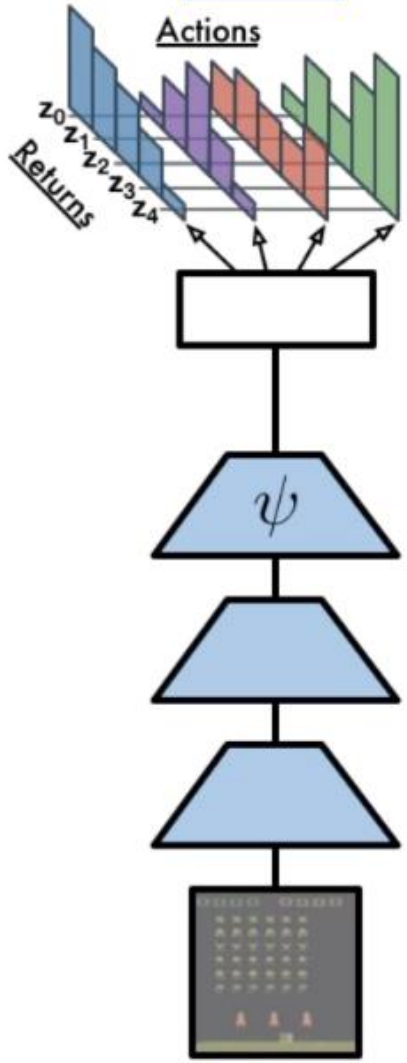
- Categorical DQN được giới thiệu vào năm 2017 bởi Marc G. Bellemare và các cộng sự.
- Mục tiêu: Học phân phối của $Q(s, a)$ thay vì xấp xỉ giá trị kỳ vọng của chúng như các thuật toán trước đây.



DQN



C51



Categorical DQN

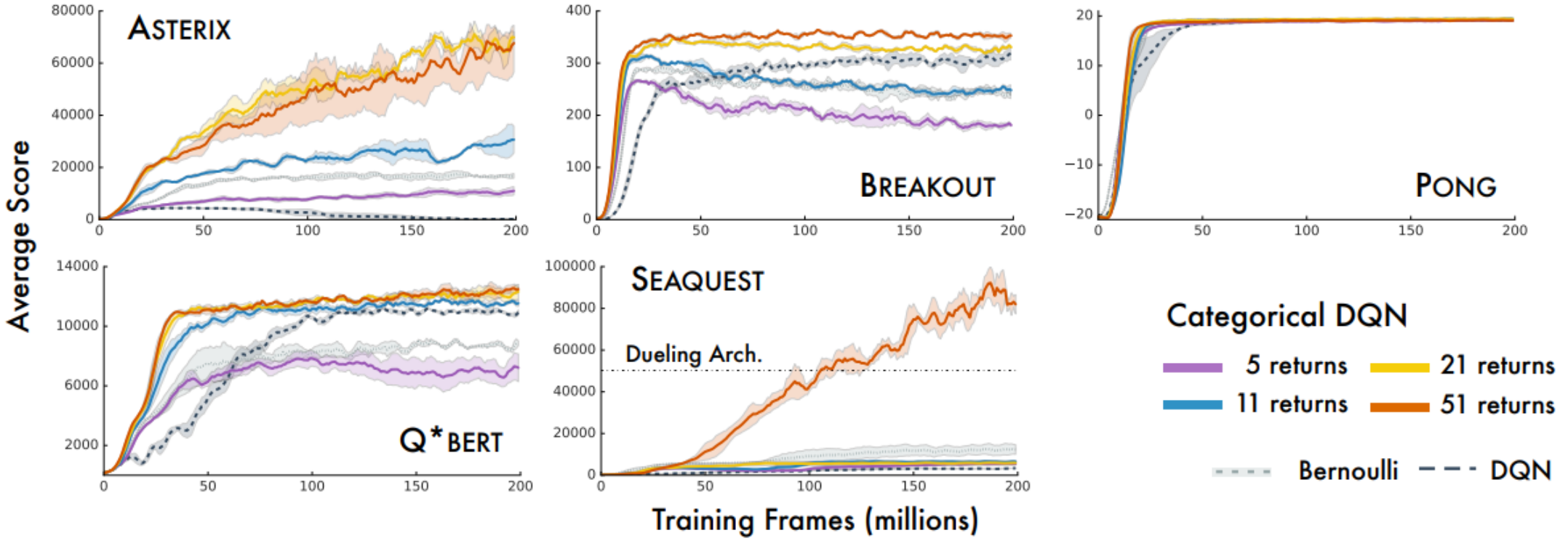
- Sử dụng phân phối rời rạc gồm các support/ atom (giá trị rời rạc) biểu diễn phân phối giá trị Z. Với $N_{atoms} \in \mathbb{N}^+$ và $i \in \{1, \dots, N_{atoms}\}$, các atom được xác định như sau:

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

- Từ kết quả thực nghiệm N = 51 cho kết quả tốt hơn nhiều so với các giá trị N khác.

=> C51

Categorical DQN



Categorical DQN

- 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.

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

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

Categorical DQN

- 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:

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

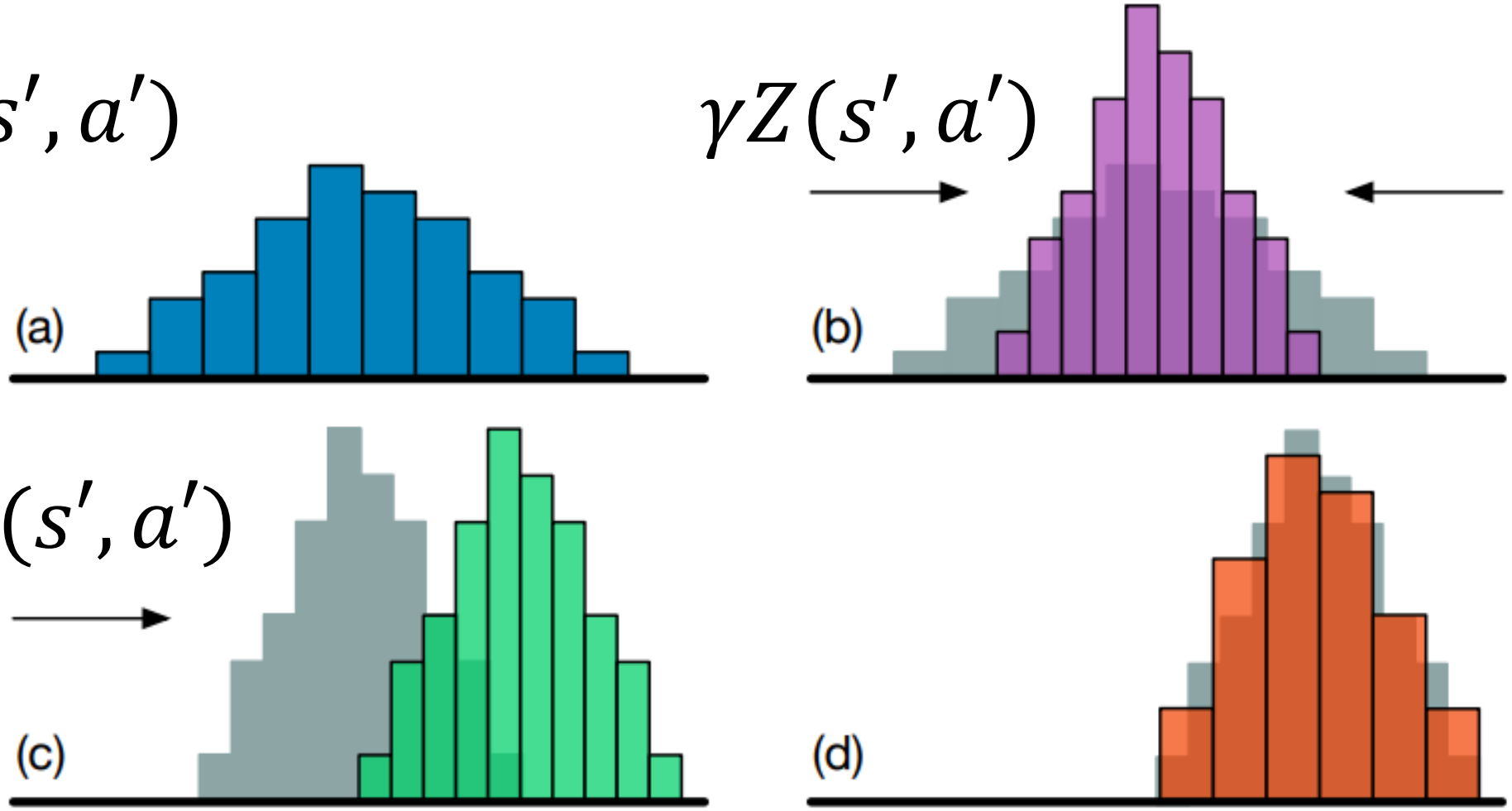
chúng ta sẽ cực tiểu hóa độ lỗi phân phối (distributional error) là khoảng cách giữa hai phân phối $r + \gamma Z(s',a')$ và $Z(s,a)$. Cũng như DQN, chúng ta vẫn dùng giá trị kỳ vọng để chọn hành động.

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

Discount / Shrink

$Z(s', a')$

$\gamma Z(s', a')$

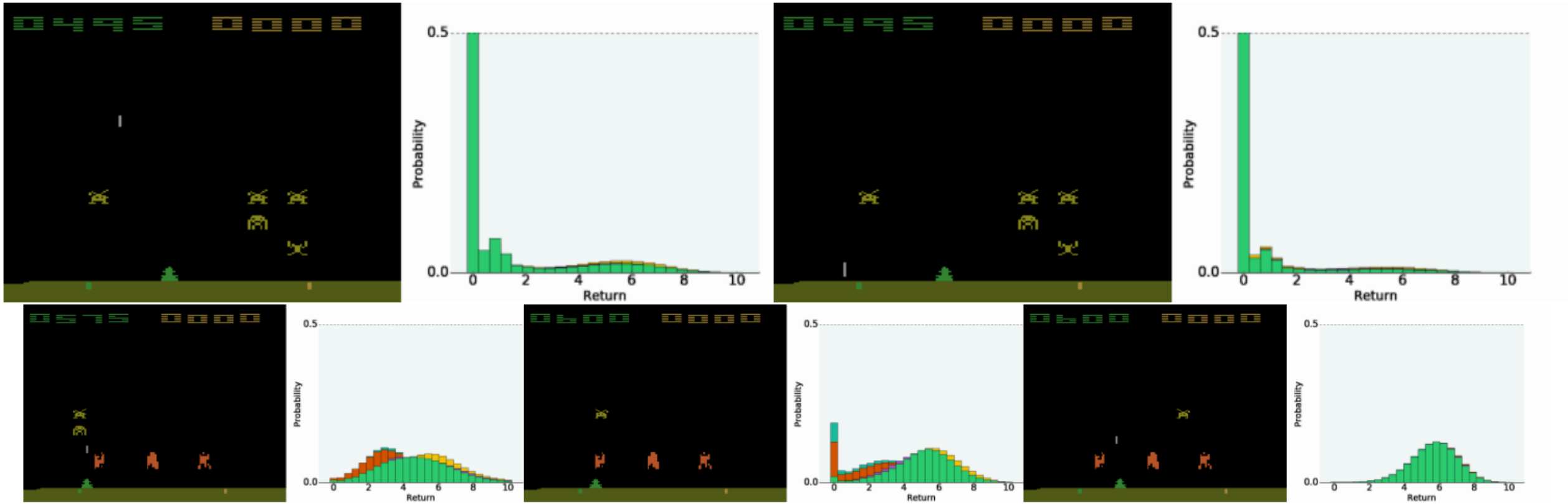


$r + \gamma Z(s', a')$

Reward / Shift

Fit/Project

Categorical DQN

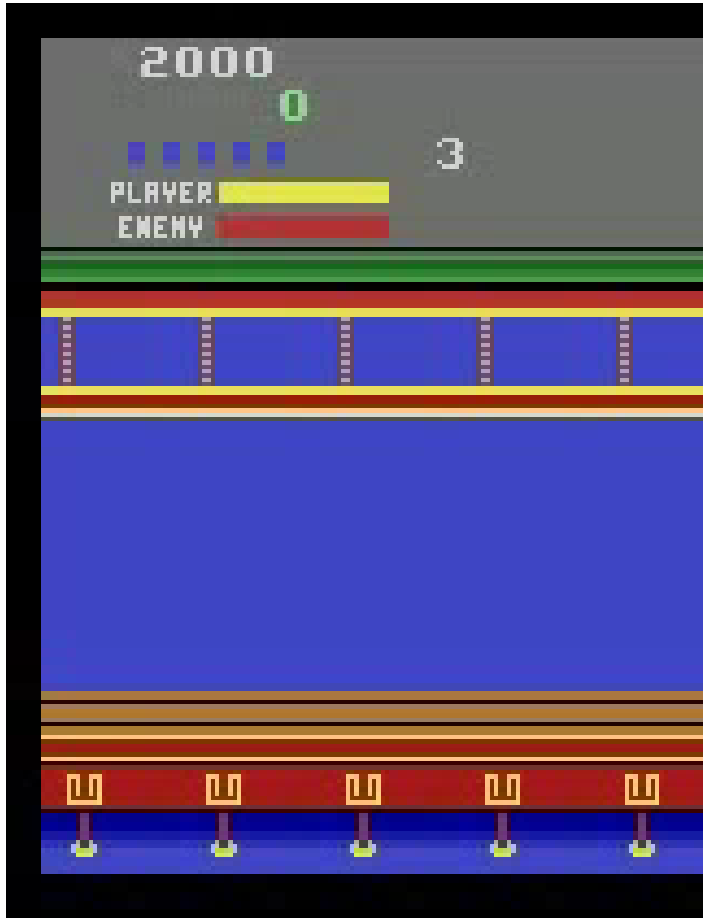


Categorical DQN

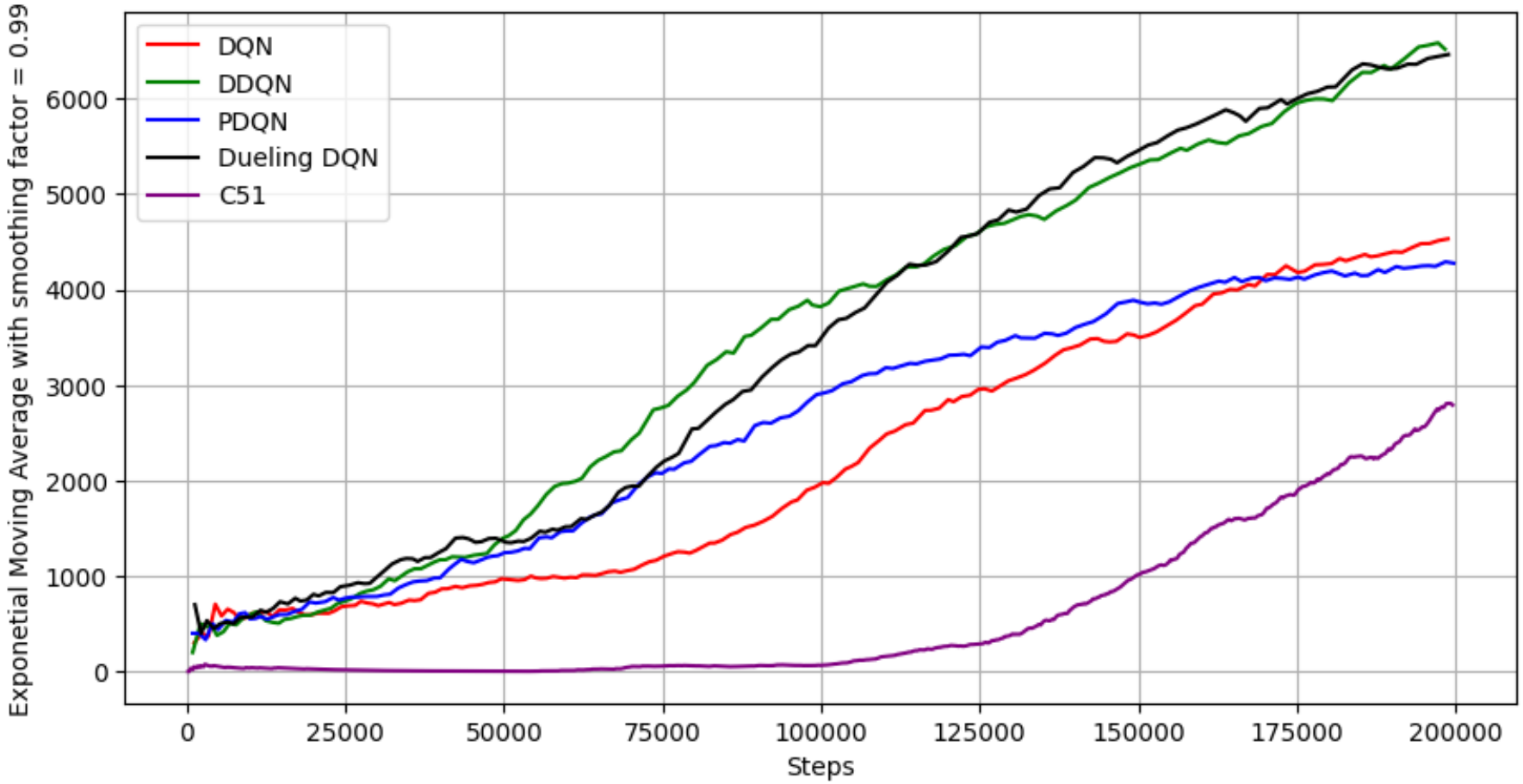
- Sử dụng CleanRL trên môi trường Kaggle với GPU P100.
- Training:
  - learning rate = 2.5e-4.
  - n_atom = 51.
  - $v_{max} = -v_{min} = 10$.
  - gamma = 0.99.
  - buffer_size = 10000.
- Thời gian train:
  - 200,000 step: 23 phút.
  - 500,000 step: 58 phút.

Result

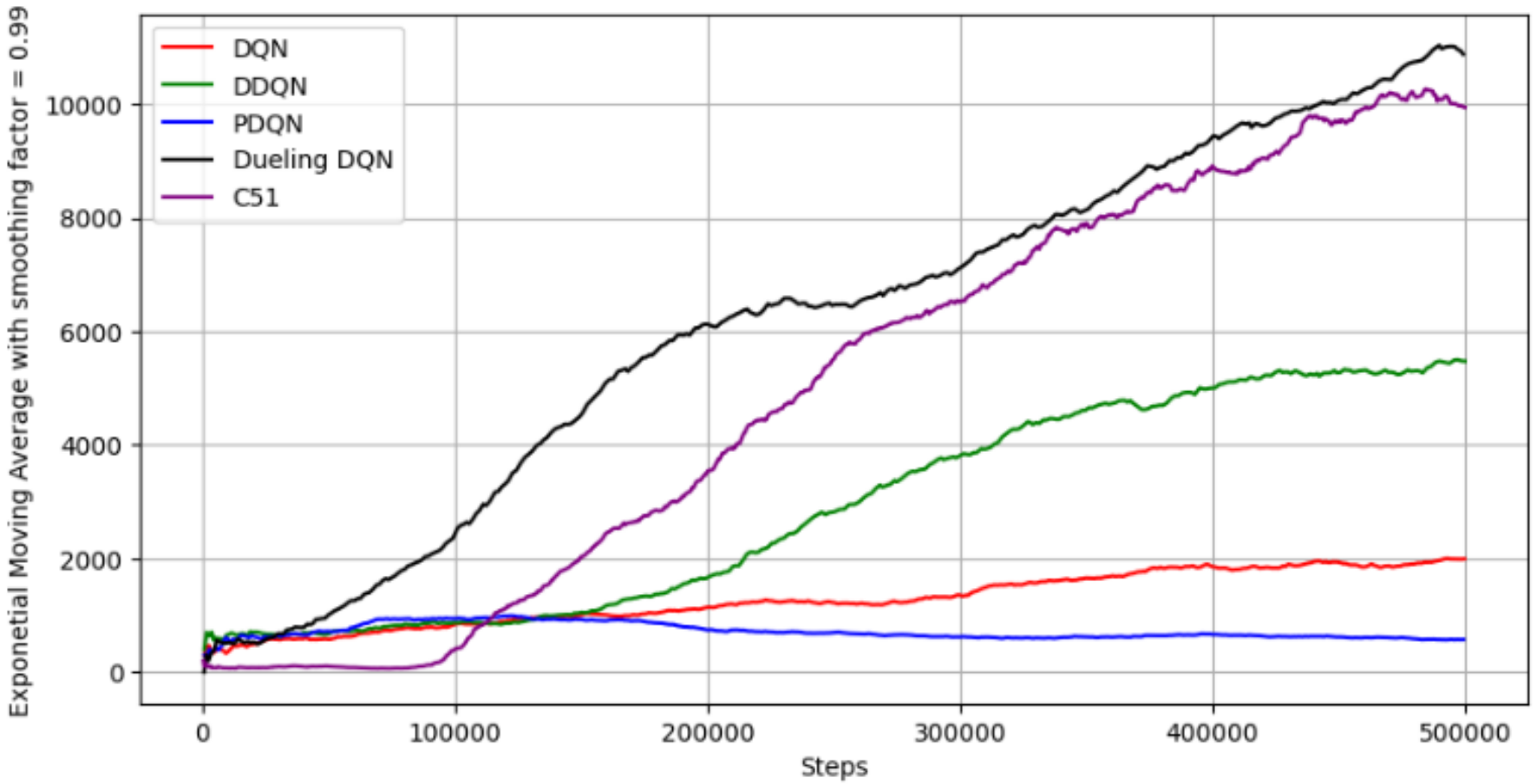
Video demo



Result


DQN
DDQN
PDQN
Dueling DQN
C51
Exponetial Moving Average with smoothing factor = 0.99
0
1000
2000
3000
4000
5000
6000
0
25000
50000
75000
100000
125000
150000
175000
200000
Steps

Result


DQN
DDQN
PDQN
Dueling DQN
C51
Exponetial Moving Average with smoothing factor = 0.99
10000
8000
6000
4000
2000
0
0
100000
200000
300000
400000
500000
Steps

Cám ơn thầy và các bạn đã lắng nghe.