*TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI*

*Viện Công nghệ thông tin và Truyền thông*

# Sử dụng mô hình Double Deep Q Learning để chơi các trò chơi trên hệ máy Atari 2600

Môn: Học sâu và Ứng dụng

*Trần Minh Hiếu (20170075)*

*Trương Quang Khánh (20170083)*

*Trương Ngọc Giang (20170067)*

*Hà Nội, ngày 7 tháng 1 năm 2021*

# Contents

## Tóm tắt kết quả

| No | Câu hỏi | Trả lời | Chi tiết (nếu có) |
|---|---|---|---|
| 1 | Ý tưởng đề tài mới không? Nhiều nhóm khác làm chưa? | Ý tưởng đề tài mới, không giống với các nhóm khác | |
| 2 | Có thu thập và gán nhãn dữ liệu không? Mô tả thông tin chi tiết nếu có. | Không | Dữ liệu của Reinforcement Learning chính là các sự kiện xảy ra trong quá trình luyện tập. |
| 3 | Dùng tập dữ liệu benchmark có sẵn không? | Không | |
| 4 | Có tiền xử lý dữ liệu không? | Có | Dữ liệu đầu vào là màn hình của game, resize về kích cỡ 84 * 84 và đưa về greyscale. Các hành động trong quá trình chơi chỉ được đánh điểm -1, 0 hoặc 1 dựa theo hành động đó có hại hay có lợi. |
| 5 | Giải quyết vấn đề bằng nhiều phương pháp khác nhau không? | Không | |
| 6 | Có đề xuất mô hình mới hay chỉnh sửa/cải tiến mô hình có sẵn nào để giải quyết vấn đề không? | Có | Thay hàm thiệt hại MSE trong paper gốc bằng hàm Huber, và thay optimizer RMSProp bằng Adam. |
| 7 | Huấn luyện mô hình mới hay chỉ dùng mô hình sẵn có cùng bộ weight sẵn có (pre-trained model)? | Huấn luyện mô hình mới | |
| 8 | Nếu huấn luyện lại thì huấn luyện từ đầu (from scratch) hay dùng pre-trained weights? | Huấn luyện từ đầu | |
| 9 | Có cài đặt mô hình không? Hay chỉ clone từ github? | Cài đặt lại mô hình | |

| | | | |
|---|---|---|---|
| 10 | Nhóm tự lập trình bao nhiêu phần trăm code trong tổng số những phần code chính (không tính những code phụ trợ) | Tự lập trình 100% | |
| 11 | Có deploy mô hình lên đâu không (ví dụ web, mobile app…)? | Không | |
| 12 | Có backend, CSDL gì không? | Không | |
| 13 | Công việc từng thành viên | Trần Minh Hiếu: Trưởng nhóm, lập trình mô hình, luyện tập mô hình<br>Trương Quang Khánh: Viết báo cáo, làm slide<br>Trương Ngọc Giang: Tìm hiểu paper, luyện tập mô hình | |
| 14 | Phần trăm đóng góp của các thành viên | Trần Minh Hiếu: **40%**<br>Trương Quang Khánh: **30%**<br>Trương Ngọc Giang: **30%** | |
| 15 | Tự chấm điểm (thang điểm 10) | Trần Minh Hiếu: **8**<br>Trương Quang Khánh: **7**<br>Trương Ngọc Giang: **7** | |
| 16 | Các vấn đề khác nếu có | Mô hình chưa chạy tố với một số game như Pong hay Skiing | |

## Lời mở đầu

Reinforcement Learning (Học tăng cường) là một lĩnh vực của Machine Learning xoay quanh việc tạo ra các mô hình thông minh, có khả năng đưa ra quyết định nhằm tối đa một giá trị kết quả được cho trước. Cùng với Supervised Learning (Học giám sát) và Unsupervised Learning (Học không giám sát), Reinforcement Learning là một trong ba mô hình cơ bản của Machine Learning. Khác với hai mô hình còn lại, Reinforcement Learning không cần đến dữ liệu đã được gán nhãn, thay vào đó chú trọng vào việc trải nghiệm trong môi trường luyện tập và cân bằng hai công việc:

- Exploration – Khám phá các khả năng có thể thực hiện trong môi trường luyện tập.
- Exploitation – Lợi dụng các kinh nghiệm đã tích luỹ được để tối đa mục tiêu.

Với sự phát triển của các mô hình Deep Learning (Học sâu) trong Supervised Learning, một câu hỏi được đặt ra: Liệu chúng ta có thể áp dụng các mô hình này để cải tiến các mô hình Reinforcement Learning sẵn có?

Để giải đáp câu hỏi này, báo cáo sẽ giới thiệu Double Deep Q Learning – một thuật toán tiêu biểu cho việc áp dụng Deep Learning vào Reinforcement Learning – và chỉ ra những cải tiến đã đạt được cùng với tư tưởng nằm sau những cải tiến này. Cuối cùng, chúng ta sẽ áp dụng thuật toán vào việc huấn luyện một số mô hình chơi game trên hệ máy Atari 2600 – một hệ máy chơi game cổ điển.

## Định nghĩa bài toán

Mô hình bài toán Reinforcement Learning cơ bản được định nghĩa giống như một mô hình quyết định Markov (Markov Decision Process – MDP), bao gồm các yếu tố sau:

- Tập hợp $S$ các trạng thái có thể xảy ra của môi trường và mô hình thông minh.
- Tập hợp $A$ các hành động mà mô hình có thể thực hiện.
- $P_a(s,s') = \Pr(s_{t+1} = s' \mid s_t = s, a_t = a)$ - xác suất để vào thời điểm $t$, từ trạng thái $s$ có thể chuyển sang trạng thái $s'$ bằng việc thực hiện hành động $a$.
- $R_a(s,s')$ - phần thưởng đạt được khi từ trạng thái $s$ chuyển sang trạng thái $s'$ bằng việc thực hiện hành động $a$.

Mô hình Reinforcement Learning tương tác với môi trường theo từng timestep một. Vào thời điểm $t$, mô hình nhận vào trạng thái hiện tại $s_t$. Mô hình quyết định đưa ra hành động $a_t$ tới môi trường, biến đổi trạng thái từ $s_t$ thành $s_{t+1}$ và tăng giá trị mục tiêu tích luỹ lên $R_a(s,s')$.
Mục tiêu của mô hình là tìm ra một policy (chiến lược) $\pi : A \times S \to [0,1]$,
$\pi(a,s) = \Pr(a_t = a \mid s_t = s)$ có thể đặt được giá trị mục tiêu tích luỹ cao nhất.

Với bài toán chơi game trên hệ máy Atari 2600, tập trạng thái $S$ tương ứng với trạng thái hiện tại trên màn hình game, tập hành động A tương ứng với các nút điều khiển của máy chơi game

và $P_a(s,s')$ và $R_a(s,s')$ được quy định theo từng game. Mục tiêu của việc luyện tập model Reinfocement Learning khi đó, rất dễ hiểu, trở thành tìm ra policy đạt điểm cao nhất trong game.


*Figure 1: Máy chơi game Atari 2600*

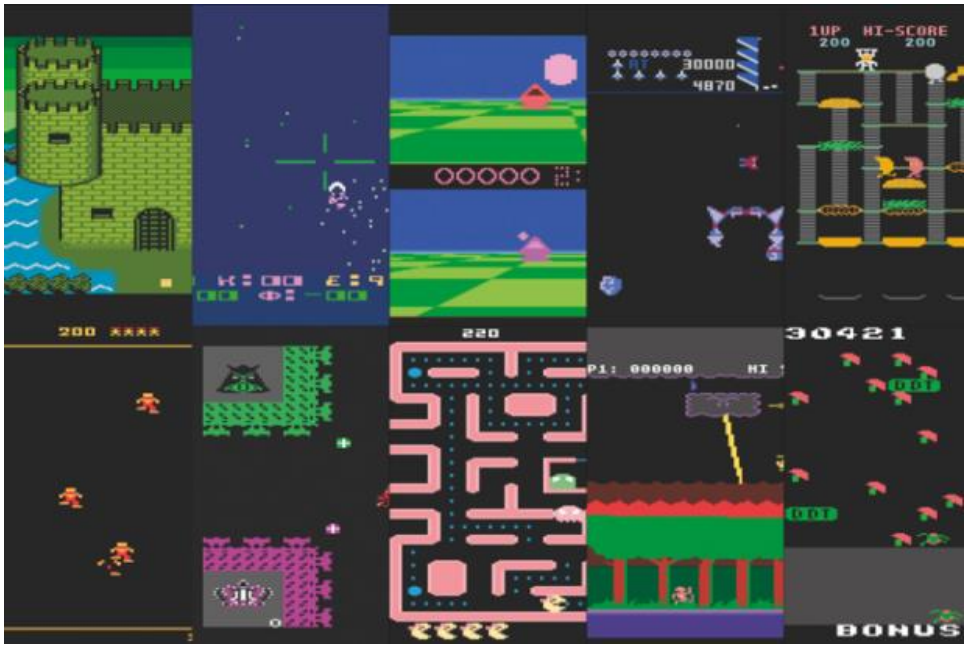



*Figure 2: Một số game trên hệ máy Atari 2600*

Trong báo cáo này, nhóm sử dụng thư viện Gym của OpenAI để giả lập môi trường Atari 2600.

# Thuật toán Double Deep Q Learning

## Q Learning

Thuật toán Q Learning hướng tới việc tìm ra hàm $Q(s,a)$ ước lượng giá trị kì vọng (quality) của việc thực hiện hành động $a$ tại trạng thái $s$. Khi đã có một ước lượng đủ tốt của hàm $Q(s,a)$, mô hình có thể tuân theo chiến lược cho ra giá trị kì vọng cao nhất.

### Hàm giá trị kì vọng Q

Việc tính toán giá trị của hàm $Q$ có thể được thực hiện như sau:

- Tại các trạng thái kết thúc $s_{end}$, ta có thể định sẵn một giá trị cố định cho mọi hành động có thể $a$.
- Tại các trạng thái khác, nếu biết được trạng thái kế tiếp $s'$ sau khi thực hiện hành động, ta có thể tính giá trị của $Q(s,a)$ theo công thức:

$$Q(s,a) = \mathrm{E}_{s'\sim\varepsilon}\left[r_a(s,s') + \gamma \max_{a'\in A} Q(s',a')\right]$$

với:

- $r_a(s,s')$ là giá trị phần thưởng khi chuyển từ trạng thái $s$ sang $s'$ sau khi thực hiện hành động $a$.
- $\gamma$ là hệ số phần thưởng. Giá trị này quyết định mô hình đánh giá các phần thưởng trong tương lai quan trọng nhiều hay ít. Ta thường đặt giá trị này trong khoảng $[0.9,1]$, với $\gamma = 1$ tương ứng với việc hàm $Q(s,a)$ có giá trị bằng đúng tổng phần thưởng còn có thể đạt được từ thời điểm hiện tại tới hết lần chạy.
- $\mathrm{E}_{s'\sim\varepsilon}[...]$ là giá trị kì vọng với mọi $s'$ kế tiếp có thể xảy ra.

Trong thực tế, việc tìm ra hàm $Q(s,a)$ là bất khả thi – chúng ta chỉ có thể tìm hàm ước lượng gần đúng $Q^*(s,a)$. Trong quá trình huấn luyện, ta để mô hình học máy thao tác trong môi trường, lưu lại các trạng thái đã xảy ra và các hành động được thực hiện vào bộ nhớ, và sử dụng những thông tin này để tối ưu hàm ước lượng $Q^*(s,a)$, sao cho sai số với giá trị thực tế là nhỏ nhất.

Ta có thể áp dụng các mô hình học máy, đặc biệt là học sâu vào đây – các mô hình này làm rất tốt công việc ước lượng hàm trong không gian đầu vào nhiều chiều, đúng như yêu cầu của bài toán. Với mỗi bước cập nhật mô hình, ta sử dụng các bộ ba $(s,a,s')$ và mô hình hiện tại để đưa ra ước lượng $Q^*(s',a)$, tính toán giá trị $Q^*(s,a)$ đúng theo công thức được nêu ra ở trên, sau đó tối ưu $Q^*(s,a)$ tới giá trị này. Nói cách khác, ta sử dụng mô hình học máy để tối ưu chính nó.

## Exploration và Exploitation

Một vấn đề quan trọng cần được giải quyết bởi mọi thuật toán Reinforcement Learning là việc cân bằng giữa exploration và exploitation. Trong quá trình huấn luyện, một mặt, ta cần phải dành thời gian khám phá môi trường hiện tại để tìm hiểu hết tất cả khả năng có thể. Mặc khác, ta cần phải tập trung lợi dụng các kinh nghiệm đã học được để đưa ra chiến lược đạt kết quả cao nhất.

Để giải quyết vấn đề này, thuật toán Deep Q Learning sử dụng một hệ số exploration $\varepsilon$ - xác suất để mô hình đưa ra hành động ngẫu nhiên (exploration) hoặc đưa ra hành động với giá trị $Q*(s,a)$ cao nhất (exploitation). Khi bắt đầu huấn luyện, $\varepsilon$ sẽ được đặt ở mức cao ($\varepsilon = 1$) để mô hình có thể tự do khám phá và tích luỹ những khả năng có thể thực hiện. Qua thời gian, $\varepsilon$ giảm dần và mô hình chuyển dần sang trạng thái lợi dụng những hiểu biến đã có để tối ưu policy hiện có.

## Experience Replay

Một vấn đề nữa cần đề cập trong thuật toán Q Learning là cách lựa chọn cách bộ kinh nghiệm $(s,a,s')$ thích hợp để tối ưu hàm $Q*(s,a)$. Mục tiêu của ta khi luyện tập các mô hình là nhằm tìm ra hàm $Q*(s,a)$ dự đoán chính xác nhất, do đó ta có thể nghĩ ngay tới việc sử dụng toàn bộ dữ liệu trong bộ nhớ để tối ưu. Cách làm này đơn giản, song lại nảy sinh ra hai vấn đề:

- Thứ nhất, các bộ $(s,a,s')$ trong khoảng thời gian liên tiếp thường có giá trị rất giống nhau, do đó việc sử dụng các dữ liệu này để luyện tập không đem lại nhiều giá trị cho việc tối ưu mô hình.
- Thứ hai, nội dung của các bộ $(s,a,s')$ trong khoảng thời gian liên tiếp cũng thường giống nhau. Việc luyện tập trên toàn bộ bộ nhớ sẽ khiến cho dữ liệu luyện tập bị mất cân bằng, các kinh nghiệm mới sẽ ít có cơ hội ảnh hưởng lên mô hình dự đoán. Lấy ví dụ đơn giản – một mô hình đang có nhiều kinh nghiệm rẽ xe sang bên trái trong bộ nhớ sẽ khó học được thêm các kinh nghiệm mới về rẽ sang bên phải.

Thay vì tối ưu hàm $Q*(s,a)$ sử dụng toàn bộ các giá trị trong bộ nhớ, ta chỉ tối ưu sử dụng một số lượng kinh nghiệm nhỏ được lấy ngẫu nhiên. Phương án này khắc phục được nhược điểm thứ nhất do ít có khả năng các kinh nghiệm liên tiếp rất giống nhau được sử dụng để luyện tập mô hình, và khắc phục được nhược điểm thứ hai vì số lượng kinh nghiệm nhỏ khiến cho các kinh nghiệm cũ không thể áp đảo được các kinh nghiệm mới.

## Thuật toán

1. Khởi tạo bộ nhớ kinh nghiệm $D$ với kích cỡ $N$.
   Khởi tạo mô hình dự đoán $Q*(s,a)$ với trọng số ngẫu nhiên.
2. For $episode = 1, M$ do:
   a. Đưa về trạng thái khởi tạo $s_1$.

b. For $t = 1, T$ do:
   i. Với xác suất $\varepsilon$ đưa ra hành động ngẫu nhiên $a_t$, ngược lại đưa ra hành động $a_t = \max_a Q^*(s_t, a; \theta)$.
   ii. Nhận lại phần thưởng $r_i$ và trạng thái mới $s_{i+1}$.
   iii. Đặt $s_i = s_{i+1}$.
   iv. Lưu kinh nghiệm $\phi_t = (s_i, a, s_{i+1})$ vào bộ nhớ $D$.
   v. Lấy ngẫu nhiên tập hợp kinh nghiệm $\{\phi_{j_1}, \phi_{j_2}, ..., \phi_j\}$ từ $D$.
   vi. Đặt $y_j = \begin{cases} r_j \; if \; \phi_j \; is \; end \; of \; episode \\ r_j + \gamma \max_{a'} Q^*(s_{j+1}, a'; \theta) \; otherwise \end{cases}$
   vii. Thực hiện tối ưu $Q^*(s, a)$ tới các giá trị $y_j$.

## Vấn đề overestimation và Double Q Learning

Deep Q Learning được đề xuất vào năm 2013 bởi các nhà nghiên cứu tại DeepMind Technology. Trong nghiên cứu này, mô hình Deep Q Learning đạt được chất lượng State-of-the-Art trong 6 trên 7 trò chơi được thí nghiệm, và vượt qua mức độ của con người trong 3 trong số đó. Tuy nhiên, Deep Q Learning thừa hưởng nhược điểm overestimation (dự đoán quá) từ Q Learning phiên bản gốc, như đã được chỉ ra bởi Hado Van Hasselt vào năm 2010 và xác nhận trên Deep Q Learning vào năm 2015 bởi DeepMind.

### Overestimation

Trong giai đoạn exploitation của thuật toán Q Learning, chúng ta ưu tiên thực thi các hành động có giá trị $Q^*(s, a)$ cao nhất. Điều này có thể khiến mô hình đưa ra dự đoán cao hơn thực tế, trên các giá trị không đúng với thực tế, và sai số này sẽ liên tục bị gia tăng do ta ưu tiên thực hiện các hành động có giá trị cao hơn các hành động chưa được khai thác.

Lấy ví dụ đơn giản: Tại trạng thái $s$, các giá trị $Q^*(s, a)$ ban đầu bằng 0 với mọi $a$. Sau một bước tối ưu hàm $Q^*(s, a)$, sẽ có một số giá trị lớn hơn 0 và một số giá trị nhỏ hơn 0. Các giá trị này có thể chưa chính xác, nhưng từ thời điểm đó trở đi, các giá trị lớn hơn 0 sẽ được ưu tiên theo đuổi hơn các giá trị nhỏ hơn 0, và chúng ta có rất ít cơ hội để sửa chữa về sau.

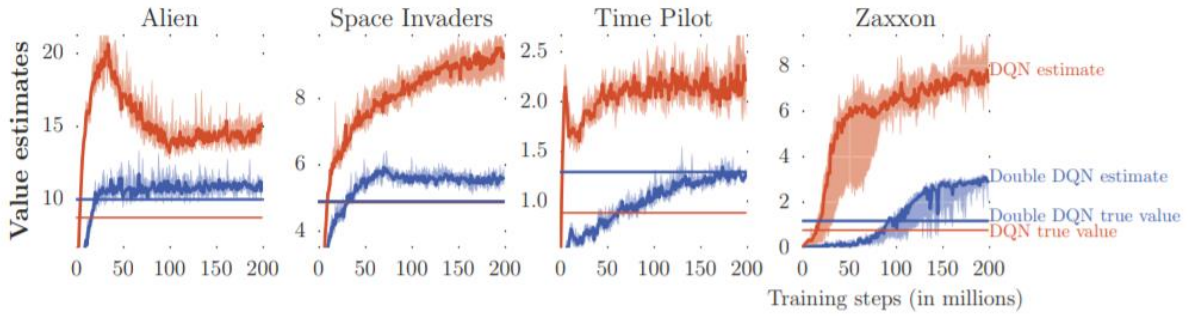



*Figure 3: Biểu đồ được đưa ra trong bài nghiên cứu "Deep Reinforcement Learning with Double Q-Learning" của DeepMind. Các đường biểu đồ biểu diễn giá trị Q được dự đoán bởi mô hình. Các đường thẳng ngang biểu diễn giá trị Q thực tế vào kết thúc trò chơi.*

## Phân tách mô hình luyện tập và mô hình dự đoán

Để khắc phục tình trạng overestimation của Q Learning, một phương án đơn giản được đề xuất trong nghiên cứu "Double Q Learning" của Hado Van Hasselt – sử dụng hai mô hình độc lập trong quá trình luyện tập.

Khi khởi tạo thuật toán, hai mô hình dự đoán - gọi là $Q_A^*(s,a)$ và $Q_B^*(s,a)$ - được đặt trọng số giống nhau, do đó đưa ra dự đoán giống nhau. Trong mỗi bước luyện tập, ta lấy ngẫu nhiên một mô hình làm mô hình dự đoán, và thực hiện tính toán như thuật toán Q Learning cổ điển để tối ưu mô hình còn lại. Khi chạy trên thực tế, ta sẽ đưa ra dự đoán sử dụng giá trị $Q*(s,a)$ trung bình của cả hai mô hình.

Phương pháp này hiệu quả vì một mô hình dự đoán sai vẫn có thể được sửa chữa lại thông qua mô hình còn lại. Trong bài báo gốc, Hasselt chứng minh được mô hình Double Q Learning có xu hướng underestimate (dựa đoán thấp) so với giá trị gốc, thay vì dự đoán cao. Điều này có thể được thấy thông qua kiểm nghiệm thực tế trong các phần sau.

## Thuật toán

Thuật toán sau đây là phiên bản được cải tiến trong bài báo "Deep Reinforcement Learning with Double Q-Learning" vào năm 2015 bởi DeepMind – thay vì sử dụng hai mô hình độc lập liên tục đổi chỗ cho nhau, phiên bản cái tiến sử dụng một mô hình luyện tập và một mô hình dự đoán. Sau một khoảng thời gian nhất định, mô hình dự đoán được sao chép lại bằng với mô hình luyện tập, cho phép mô hình có thể tiến tới giá trị tối ưu.

1. Khởi tạo bộ nhớ kinh nghiệm $D$ với kích cỡ $N$.
   Khởi tạo mô hình luyện tập $Q*(s,a)$ với trọng số ngẫu nhiên, và mô hình dự đoán $Q'(s,a)$ với trọng số bằng $Q*(s,a)$.
2. For $episode = 1, M$ do:
   a. Đưa về trạng thái khởi tạo $s_1$.
   b. For $t = 1, T$ do:

i. Với xác suất $\varepsilon$ đưa ra hành động ngẫu nhiên $a_t$, ngược lại đưa ra hành động $a_t = \arg\max_a Q^*(s_t, a; \theta)$.

ii. Nhận lại phần thưởng $r_i$ và trạng thái mới $s_{i+1}$.

iii. Đặt $s_i = s_{i+1}$.

iv. Lưu kinh nghiệm $\phi_t = (s_i, a, s_{i+1})$ vào bộ nhớ $D$.

v. Lấy ngẫu nhiên tập hợp kinh nghiệm $\{\phi_{j_1}, \phi_{j_2}, ..., \phi_j\}$ từ $D$.

vi. Đặt $y_j = \begin{cases} r_j \; if \; \phi_j \; is \; end \; of \; episode \\ r_j + \gamma \max_{a'} Q'(s_{j+1}, a'; \theta) \; otherwise \end{cases}$

vii. Thực hiện tối ưu $Q^*(s, a)$ tới các giá trị $y_j$.

viii. If shouldUpdateTarget() then $Q'(s, a) = Q^*(s, a)$

## Mô hình học sâu được sử dụng

Dữ liệu đầu vào của mô hình Reinforcement Learning là hình ảnh trên giao diện trò chơi, có kích thước $210 \times 160$ với 128 màu sắc khác nhau. Xử lý toàn bộ lượng thông tin này yêu cầu tài nguyên tính toán lớn, do đó bước đầu tiên của mô hình tính toán là tiền xử lý hình ảnh này, bằng cách đưa về định dạng đen trắng và giảm kích cỡ xuống còn $110 \times 84$. Sau đó ta cắt lấy phần đáy của hình ảnh – phần cung cấp toàn bộ thông tin cần thiết cho trò chơi – và thu lại một hình ảnh kích cỡ $84 \times 84$. 4 khung hình liên tiếp của trò chơi được gộp vào làm một nhằm thể hiện dòng thời gian trôi qua trong game. Kết quả nhận được là dữ liệu đầu vào có kích cỡ $84 \times 84 \times 4$.

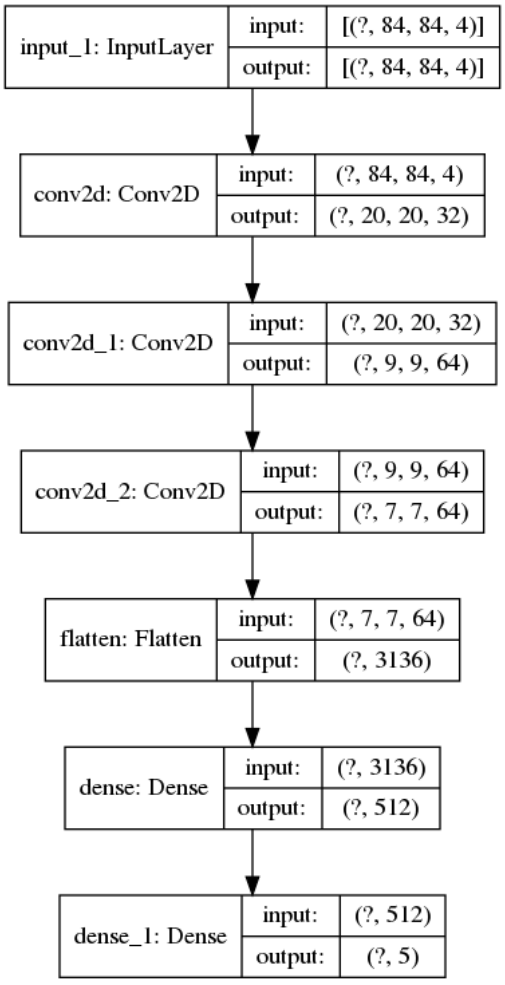


*Figure 4: Mô hình Học sâu được sử dụng*

Dữ liệu đầu vào này được đưa vào mô hình học sâu được mô tả trong hình bên cạnh. Tất cả các lớp sử dụng hàm kích hoạt ReLU, trừ lớp cuối cùng không sử dụng hàm kích hoạt. Đầu ra của mô hình là 5 giá trị số thực, tương ứng với 5 giá trị $Q^*(s, a)$ dự đoán cho 5 hành động có thể xảy ra.

Giá trị phần thưởng của trò chơi cũng được tiền xử lý qua hai bước:

- Mọi giá trị $r_i$ đều được thay thế bằng $sign(r_i)$ -1, 0 hoặc 1 cùng dấu với $r_i$. Bằng cách này, mô

hình chỉ quan tâm tới việc một hành động có lợi hay có hại cho mục tiêu, mà không quan tâm tới mức độ có lợi hay có hại ra sao. Điều này sẽ giúp ích trong một số trò chơi như Breakout – có các mức điểm số khác nhau tương ứng với việc phá các ô ở các vị trí khác nhau, nhưng ta quan tâm tới việc phá được nhiều ô hơn là việc tập trung câu điểm tại một số vị trí điểm cao.

- Nếu hành động dẫn tới kết thúc trò chơi, $r_i = -1$. Điều này không được cài đặt trong môi trường, ta cần phải thực hiện bên ngoài vì ta muốn mô hình tránh các nước đi dẫn tới thua cuộc.

## Luyện tập mô hình

Mô hình được luyện tập trong 4 trò chơi khác nhau - Breakout, MsPacman, Alien và Assault – qua 3.2 triệu khung hình (tương đương với 37 ngày chơi game liên tục). Các hyperparameter được sử dụng được tham khảo từ bài báo "Playing Atari with Deep Reinforcement Learning" từ DeepMind Technologies:

- $\gamma = 0.99$
- $\varepsilon = 1 \rightarrow 0.1$ trong 1 triệu khung hình. Mô hình bắt đầu cho phép exploitation sau khi đã trải qua 50000 khung hình.
- Mô hình $Q^*(s,a)$ được tối ưu lại sau mỗi 4 khung hình, mỗi lần sử dụng 32 bộ kinh nghiệm.

Nhóm có thay đổi một số yếu tố so với các nghiên cứu đi trước:

### Hàm sai số

Bài báo "Playing Atari with Deep Reinforcement Learning" từ DeepMind Technologies sử dụng hàm sai số Mean Square Error:

$$L_i(\theta_i) = \mathrm{E}_{s,a}\left[\left(y_i - Q(s,a;\theta_i)\right)^2\right]$$

với $\theta_i$ là các giá trị trọng số của mô hình tại thời điểm $i$. Tuy nhiên trong quá trình thực hành thực tế, nhóm sử dụng hàm sai số Huber:

$$L_i(\theta_i) = \begin{cases} \frac{1}{2}a^2 \ for|a| \le \delta \\ \delta(|a| - \frac{1}{2}\delta) otherwise \end{cases}$$
$$with\, a = y_i - Q(s,a;\theta_i)$$

Kiểm nghiệm thực tế cho thấy hàm sai số này giúp việc luyện tập ổn định hơn trong các bước đầu tiên, do sai số của hàm Huber gia tăng ở mức tịnh tiến với các giá trị sai lệch lớn, thích hợp với giai đoạn đầu khi mô hình còn dự đoán hàm giá trị không chính xác.

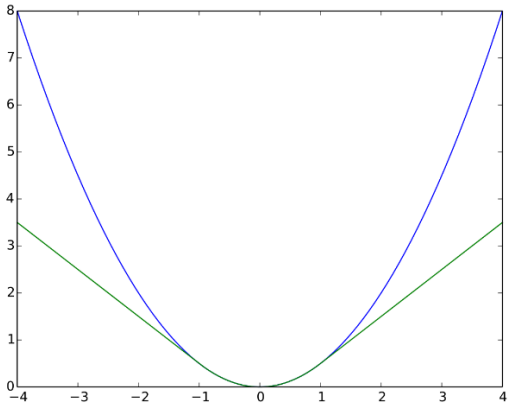



*Figure 5: Hàm Huber (xanh lá cây) và hàm MSE (xanh dương) ở các mức độ chênh lệch khác nhau*

### Thuật toán tối ưu

Thay vì sử dụng thuật toán RMSProp như bài báo của DeepMind, nhóm sử dụng thuật toán tối ưu Adam với learning rate 0.00025, vì thuật toán này cho thời gian hội tụ nhanh hơn. Nhóm cũng sử dụng gradient clipping để tránh bùng nổ gradient trong các bước exploration.

### Kích thước bộ nhớ

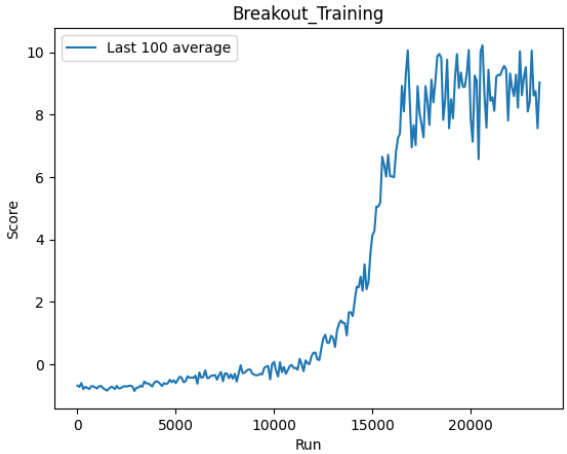
Thay vì sử dụng bộ nhớ có kích thước 1 triệu kinh nghiệm gần nhất, nhóm chỉ lưu trữ 100 nghìn. Thay đổi này không làm ảnh hưởng quá nhiều tới kết quả luyện tập và cho phép nhóm có đủ bộ nhớ thể luyện tập hai mô hình cùng một lúc.

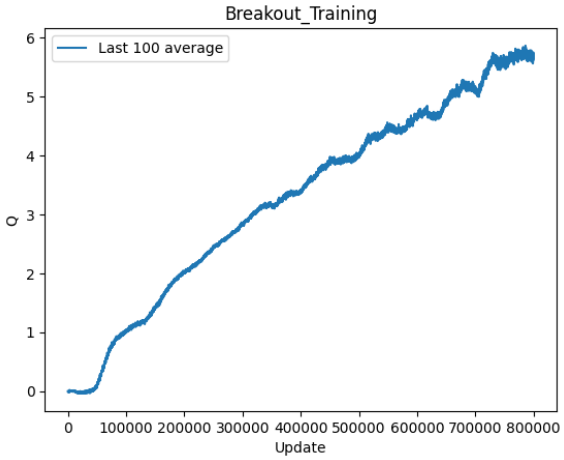
## Thử nghiệm thực tế

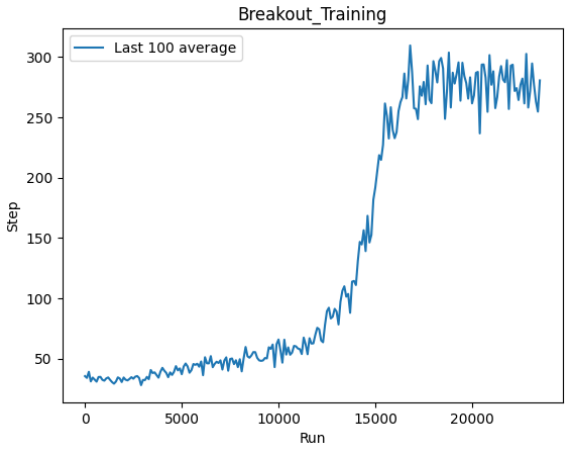
Quan sát đồ thị hàm sai số ghi lại được trong quá trình chạy, ta thấy có sự giao động rất lớn giữa các lần chạy do yếu tố ngẫu nhiên của thuật toán, song nhìn chung thì sai số có chiều hướng tăng lên theo thời gian. Để có thể đánh giá chính xác hơn hiệu quả của thuật toán, ta cần phải sử dụng các tham số khác như giá trị Q dự đoán trung bình và điểm số thực tế, thay vì sử dụng sai số như với các thuật toán học máy bình thường.

Quan sát đồ thị giá trị Q dự đoán trung bình qua thời gian, ta thấy xu hướng tăng biểu diễn rõ ràng qua qua trình học, đi theo đúng sự gia tăng của điểm số theo thời gian. Ta cũng thấy rõ hiện tượng underestimate được chỉ ra bởi Hasselt  khi giá trị hàm Q được dự đoán luôn thấp hơn giá trị điểm số thực tế.

Dưới đây là biểu đồ biểu diễn các giá trị theo dõi được trong quá trình luyện tập mô hình chơi game Breakout. Từ trên xuống dưới, từ trái qua phải bao gồm giá trị hàm sai số qua mỗi khung hình, điểm số đạt được trong các lần chơi, giá trị hàm Q dự đoán trung bình qua mỗi khung hình và số lượng khung hình chơi được trong mỗi lần chơi.


## Tài liệu tham khảo


- Mnih, Volodymyr & Kavukcuoglu, Koray & Silver, David & Graves, Alex & Antonoglou, Ioannis & Wierstra, Daan & Riedmiller, Martin. (2013). Playing Atari with Deep Reinforcement Learning.
- Van Hasselt, Hado. (2010). Double Q-learning.. 2613-2621.
- Van Hasselt, Hado & Guez, Arthur & Silver, David. (2015). Deep Reinforcement Learning with Double Q-learning.