Continuous state spaces

Nhưng trong thực tế, nhiều hệ thống (robot, ô tô, máy bay...) có thể ở **vô số vị trí khác nhau** → tạo thành **không gian trạng thái liên tục (continuous):**

- **Continuous state space**: Trạng thái được biểu diễn bằng một **vector số thực**, thay vì chọn từ một tập hợp nhỏ rời rạc.
- Trong các bài toán có **continuous state**, thuật toán cần **xử lý các vector số thực** và đưa ra quyết định (hành động) dựa vào đó.
- Trạng thái có thể thuộc **không gian có hàng ngàn chiều** tùy theo độ phức tạp của hệ thống.

Deep Q-Network (DQN)

- Huấn luyện một **mạng nơ-ron** để xấp xỉ **hàm giá trị hành động** (Q-function)

→ từ đó chọn hành động tốt nhất trong từng trạng thái để tối ưu phần thưởng dài hạn.

- Đầu vào là các chiều biểu diễn trạng thái → đưa vào mạng nơ-ron → đầu ra là giá trị Q(s, a)
- Huấn luyện như bài toán hồi quy: **tối thiểu hóa sai số bình phương (MSE)** giữa dự đoán và giá trị mục tiêu y
- Dữ liệu huấn luyện được tạo từ quá trình huấn luyện từ môi trường, chỉ nên lưu **10,000 experience gần nhất** để tiết kiệm bộ nhớ (Replay Buffer)

Vòng lặp học Q-function

1. **Khởi tạo Q-network với trọng số ngẫu nhiên**
2. **Lặp lại nhiều lần**:
    - Chơi trong môi trường (dùng policy hiện tại hoặc random)
    - Lưu lại các experience tuples vào replay buffer
    - **Lấy mẫu batch** từ replay buffer
    - Tính toán x, y
    - Huấn luyện lại neural network để **xấp xỉ hàm Q**

Sau khi có mô hình Q-function:

$$\pi(s) = \arg\max_a Q(s, a)$$

Tức là tại mỗi trạng thái s, chọn hành động a có giá trị Q lớn nhất.

Nhược điểm

- Để chọn hành động tốt nhất tại mỗi trạng thái, cần chạy mạng **4 lần** (mỗi lần với 1 hành động khác nhau)
- Chậm và không hiệu quả khi huấn luyện hoặc suy luận

⇒ Cải tiến: output ra likelihood của các state, giảm được số lần chạy và số lượng input đầu vào.

ϵ greedy policy

- **Khi mới bắt đầu học**, mô hình chưa biết hành động nào là tốt nhất trong mỗi trạng thái.
- Tuy nhiên, **vẫn phải chọn hành động liên tục** để vừa điều khiển agent vừa thu thập dữ liệu huấn luyện.
- Nếu chỉ chọn hành động có reward cao nhất ngay từ đầu, mô hình sẽ có xu hướng luôn luôn bỏ qua một số hành động, mà đôi khi có thể tốt
- Do đó cần thực hiện chọn một cách ngẫu nhiên hơn. Đây gọi là **exploration** (khám phá). Ngược lại, chọn greedy được gọi là **exploitation** (khai thác kiến thức hiện tại)

Epsilon-Greedy Policy là gì?

- Với xác suất **ε (Epsilon)**: chọn hành động **ngẫu nhiên**
- Với xác suất **1 - ε**: chọn hành động tốt nhất theo Q(s, a)

Giảm dần Epsilon theo thời gian

- Ban đầu ε lớn (ví dụ 1.0): **chọn ngẫu nhiên hoàn toàn**
- Sau đó giảm dần (ví dụ xuống 0.01): gần như luôn greedy
- Giúp mô hình:
    - Ban đầu khám phá nhiều → thu thập kiến thức

- Sau đó khai thác hiểu biết → học ổn định

Mini batch and soft updates

Mini-Batching – Giảm thời gian huấn luyện

Vấn đề

- Trong học tăng cường, ta có thể lưu trữ **10.000 tuple** trong **replay buffer**
- Nếu dùng toàn bộ 10.000 để train mô hình ở mỗi bước → **quá tốn thời gian tính toán**

Giải pháp

- **Chỉ chọn ngẫu nhiên một phần nhỏ** – ví dụ **1.000 tuple** → gọi là **mini-batch**
- Dùng mini-batch để tạo tập dữ liệu huấn luyện (x,y)(x, y)(x,y) nhỏ gọn hơn
- Huấn luyện mô hình trên tập nhỏ này → **nhanh hơn**, dù **hơi nhiễu (noisy)**

Lợi ích

- **Tăng tốc huấn luyện**
- Giúp mô hình **cập nhật thường xuyên hơn**
- Áp dụng tốt cho cả **supervised learning** (hồi quy tuyến tính, logistic regression, mạng neural, v.v.)

Soft Updates – Giảm dao động mô hình

Vấn đề

- Trong phiên bản cũ, ta **thay hoàn toàn** Q
- Nếu Q **tình cờ học kém hơn** (do noise), ta có thể **làm hỏng toàn bộ mô hình**

Soft Update

- Cập nhật Q **từ từ, từng bước nhỏ**:

$$W \leftarrow \tau \cdot W_{\text{new}} + (1 - \tau) \cdot W$$

  - Với $\tau = 0.01$, nghĩa là chỉ **nhận 1% giá trị mới**

Lợi ích

- **Giảm nguy cơ dao động hoặc phân kỳ**
- Cập nhật **ổn định hơn**, giúp mô hình **hội tụ dần dần**
- Dễ kiểm soát quá trình học hơn