Policy Evaluation vs. Control

- **Policy Evaluation (Ước lượng chính sách)**:
  - Cho trước một **chính sách cố định** π, nhiệm vụ là tính toán hoặc xấp xỉ hàm giá trị trạng thái $V^{\pi}(s)$ (hay hàm giá trị hành động $Q^{\pi}(s,a)$) tương ứng.
  - Mục đích: xác định xem policy đó "tốt" hay "xấu"—nói cách khác, **đánh giá khả năng thu được reward lâu dài** khi agent tuân theo policy này.
- **Control (Điều khiển / Tìm chính sách tối ưu)**:
  - Mục tiêu là **tìm một chính sách** π^* làm cho hàm giá trị tối ưu $V^*(s)$ (hay $Q^*(s,a)$) được tối đa.
  - Thay vì chỉ xác định giá trị, agent phải **cải thiện** policy sao cho càng nhiều reward càng tốt.
- **Điểm khác biệt then chốt**:
  - Với **policy evaluation**, policy đã được định nghĩa sẵn, chỉ việc ước lượng V^{π} hoặc Q^{π}.
  - Với **control**, policy là ẩn, cần lặp đi lặp lại giữa việc ước lượng giá trị và cải thiện policy (như Policy Iteration, Value Iteration).

---

3. Vai trò của Dynamic Programming trong Policy Evaluation

- **Dynamic Programming (DP)** yêu cầu:
  1. **Biết trước mô hình chuyển tiếp** $P(s', r \mid s, a)$ của MDP.
  2. **Không gian trạng thái/ hành động** đủ nhỏ để lưu trữ bảng giá trị.
- **Ứng dụng của DP**:
  - Khi policy π cố định, ta có hệ **Bellman Expectation Equation** cho $V^{\pi}(s)$:

$$V^{\pi}(s) = \sum_{a} \pi(a \mid s) \sum_{s',r} P(s', r \mid s, a)\,[r + \gamma\, V^{\pi}(s')].$$

  - Giải trực tiếp hệ phương trình đó (hệ tuyến tính) để tìm $V^{\pi}(s)$. Tuy nhiên, nếu số lượng trạng thái lớn, việc giải chính xác sẽ tốn kém về thời gian và bộ nhớ.
- **Hạn chế**:
  1. **Yêu cầu biết mô hình chuyển tiếp** đầy đủ (đối với nhiều bài toán thực tế, mô hình không rõ hoặc quá phức tạp).
  2. **Không gian trạng thái lớn** (độ phức tạp mảng/bảng tăng nhanh), dẫn đến việc tính ngược hoặc lưu trữ V không khả thi.
- **Giải pháp thay thế**:
  - **Iterative Policy Evaluation**: thay vì giải hệ tuyến tính, ta lặp lại cập nhật giá trị cho đến khi hội tụ. Mỗi lần lặp gọi là "sweep" qua toàn bộ trạng thái.

---

4. Iterative Policy Evaluation

4.1 Ý tưởng chính

- Ban đầu cho một ước lượng khởi tạo $V_0(s)$ (có thể gán 0 hoặc bất kỳ giá trị nào).
- Tại mỗi **sweep** k, cập nhật ước lượng $V_{k+1}(s)$ theo công thức Bellman Expectation dùng ước lượng cũ V_k.
- Lặp cho đến khi **max_delta** (độ thay đổi lớn nhất giữa V_{k+1} và V_k) nhỏ hơn một ngưỡng θ đặt trước. Khi đó, ta coi $V_k \approx V^\pi$.

4.2 Công thức cập nhật

- Với mỗi trạng thái s:

$$V_{k+1}(s) = \sum_a \pi(a \mid s) \sum_{s',r} P(s', r \mid s, a)\,[r + \gamma\, V_k(s')].$$

- Chú ý: khi cập nhật, để tránh "dùng giá trị mới giữa chừng" để ước lượng (sẽ dẫn đến lỗi), ta thường sử dụng **hai mảng**:
  - **V** giữ ước lượng cũ V_k.
  - **V'** (gọi là V_{new}) lưu kết quả $V_{k+1}(s)$.
  - Sau khi quét xong tất cả s, gán $V \leftarrow V'$ cho lần lặp kế tiếp.

4.3 Quy trình (Pseudo-code)

1. Khởi tạo:

$$V(s) \leftarrow 0 \quad \forall s.$$

2. Lặp (cho đến khi hội tụ):
   1. Gán $\Delta \leftarrow 0$.
   2. Tạo mảng mới V_{new}.
   3. Với mỗi s không phải terminal:
      1. $v_{\text{old}} \leftarrow V(s)$.
      2. Tính $v_{\text{new}} = \sum_a \pi(a \mid s) \sum_{s',r} P(s', r \mid s, a)[r + \gamma V(s')]$.
      3. Gán $V_{\text{new}}(s) \leftarrow v_{\text{new}}$.
      4. $\Delta \leftarrow \max(\Delta,\ |v_{\text{new}} - v_{\text{old}}|)$.
   4. Với trạng thái terminal, có thể giữ $V_{\text{new}}(s_{\text{terminal}}) = 0$ hoặc không thay đổi.
   5. Gán $V \leftarrow V_{\text{new}}$.
   6. Nếu $\Delta < \theta$, dừng vòng lặp.

4.4 Sự hội tụ

- Cho bất kỳ khởi tạo V_0, sau đủ nhiều lần quét (iterative sweeps), $V_k(s)$ sẽ **hội tụ** về $V^\pi(s)$ (đúng giá trị ước lượng) khi $k \to \infty$ và $\theta \to 0$.

- Thời gian hội tụ phụ thuộc:
  - **Kích thước không gian trạng thái**.
  - **Giá trị** γ (nếu γ gần 1, hội tụ chậm hơn).
  - **Cấu trúc mô hình** (trạng thái có nhiều liên kết chuyển tiếp, cập nhật nhiều lần để "lan" giá trị).

---

5. Ví dụ minh họa trên Grid World

5.1 Mô tả Grid World

- Lưới 4×4 (16 ô), trong đó:
  - **Trạng thái terminal** nằm ở **góc trên bên trái** và **góc dưới bên phải**. Khi agent bước vào các ô này, episode kết thúc.
  - Với mỗi bước di chuyển, agent nhận **reward = –1** (để khuyến khích agent tìm đường ngắn nhất đến terminal).
  - Discount factor $\gamma = 1$ (không chiết khấu), vì đây là episodic task có độ dài hữu hạn.
  - **Chính sách (Policy)**: agent chọn **ngẫu nhiên** (random policy)—mỗi bước, 4 hướng (up, down, left, right) đều có xác suất 0.25. Nếu hướng ra khỏi lưới, agent vẫn ở nguyên ô cũ nhưng vẫn bị trừ –1 reward.

5.2 Bảng trạng thái khởi đầu

- Khởi tạo ước lượng $V(s) = 0$ cho mọi trạng thái, kể cả non-terminal (terminal luôn giữ 0 hoặc không cần cập nhật).

5.3 Lần quét đầu tiên (Sweep 1)

- Với mỗi trạng thái phi-terminal (có 14 trạng thái), tính:

$$V_1(s) = \sum_{a \in \{\text{up,down,left,right}\}} \frac{1}{4}[-1 + V_0(s')],$$

  trong đó s' là ô kế tiếp nếu di chuyển theo a; nếu ra khỏi lưới, $s' = s$. Vì $V_0(s') = 0$ với mọi s', ta có:

$$V_1(s) = \sum_a \frac{1}{4}(-1 + 0) = -1.$$

- Kết quả: sau sweep 1, mọi ô phi-terminal có giá trị –1, ngoại trừ các ô ngay cạnh terminal (khi di chuyển về terminal, s' là terminal, nhưng $V_0(\text{terminal}) = 0$). Tuy nhiên, vì khởi đầu mô hình terminal không bị trừ thêm gì, kết quả chung vẫn là –1 với $\gamma = 1$.

5.4 Lần quét thứ hai (Sweep 2)

- Bây giờ $V_1(s')$ đã là –1 cho nhiều ô, nên:

$$V_2(s) = \sum_a \tfrac{1}{4}[-1 + V_1(s')] = \sum_a \tfrac{1}{4}(-1 + (-1)) = -2.$$

Tuy nhiên, lưu ý: nếu một action a đưa agent sang terminal, lúc đó reward = –1 nhưng $V_1(\text{terminal}) = 0$. Ví dụ, trạng thái bên cạnh góc terminal:

- Có 1/4 hướng "vào terminal" → $-1 + 0 = -1$.
- 3/4 hướng khác → $-1 + (-1) = -2$.
- Tổng: $V_2(s) = \tfrac{1}{4}(-1) + \tfrac{3}{4}(-2) = -\tfrac{1}{4} - \tfrac{6}{4} = -\tfrac{7}{4} = -1.75$.

- Với các ô xa terminal, chưa thể "tiếp cận" nên có $V_2(s) = -2$.

5.5 Các lần quét tiếp theo

- Mỗi lần quét, giá trị $V_k(s)$ tiếp tục được làm mới dựa trên "trung bình" reward –1 cộng với giá trị các ô kế tiếp từ lần quét trước.
- Giá trị hội tụ khi k đủ lớn, vì đường đi ngắn nhất từ mỗi ô đến terminal có **độ dài hữu hạn**. Cuối cùng, $V(s)$ sẽ phản ánh **độ ngắn** của đường đi đến terminal (với random policy, tức độ kỳ vọng số bước).

5.6 Dừng khi hội tụ

- Chọn ngưỡng $\theta = 0.001$.
- Mỗi sweep, tính $\Delta = \max_s \left| V_{\text{new}}(s) - V_{\text{old}}(s) \right|$.
- Khi $\Delta < \theta$, coi như hàm V đã hội tụ gần tới V^π.

---

6. Tóm tắt quá trình Iterative Policy Evaluation

1. **Khởi tạo**: Đặt $V(s) = 0$ hoặc một giá trị bất kỳ.
2. **Lặp cho tới khi hội tụ**:
   - Với mỗi state s (ngoại trừ các trạng thái terminal), cập nhật:

$$V_{k+1}(s) = \sum_a \pi(a \mid s) \sum_{s',r} P(s', r \mid s, a)[r + \gamma V_k(s')].$$

   - Tính $\Delta = \max_s |V_{k+1}(s) - V_k(s)|$.
   - Nếu $\Delta < \theta$, dừng; ngược lại, gán $V_k \leftarrow V_{k+1}$ và tiếp tục.
3. **Kết quả**: Khi dừng, $V(s)$ là ước lượng gần đúng của $V^\pi(s)$, tức là **giá trị kỳ vọng dài hạn** khi agent tuân theo policy π.

---

7. Ứng dụng và Hạn chế của Policy Evaluation với DP

- **Khi nào áp dụng được**:
  1. **Mô hình chuyển tiếp** P đã biết và có thể tra cứu.
  2. **Không gian trạng thái** đủ nhỏ để lặp qua và lưu trữ giá trị.
- **Hạn chế**:
  1. Nếu **không gian trạng thái quá lớn** (ví dụ: hàng triệu trạng thái), việc lặp qua tất cả state mỗi lần sweep là không khả thi.
  2. **Nếu mô hình** P **không biết** (model-free setting), không thể tính trực tiếp phép tổng trên $P(s', r \mid s, a)$.
  3. **Thời gian hội tụ** có thể rất chậm khi γ gần 1 hoặc khi trạng thái liên kết rộng (phải thông tin "lan" qua nhiều bước).
- **Giải pháp thay thế**:
  - Trong những trường hợp mô hình không biết và không gian trạng thái quá lớn, dùng các phương pháp **model-free** (ví dụ **TD Learning**, **Monte Carlo**, **Q-learning**) hoặc **approximate methods** (học hàm giá trị bằng hàm xấp xỉ như mạng neural).

---

8. Tổng kết (Summary)

1. **Policy Evaluation**: Cho policy π, tính hoặc xấp xỉ $V^{\pi}(s)$ (hoặc $Q^{\pi}(s, a)$). Không thay đổi policy.
2. **Control**: Cải thiện policy để đạt reward cao nhất → cần lặp giữa policy evaluation và policy improvement.
3. **Iterative Policy Evaluation**: Dùng **dynamic programming** để cập nhật dần giá trị, hội tụ về V^{π} nếu mô hình P biết và không gian trạng thái đủ nhỏ.
4. **Ví dụ Grid World**: Mỗi bước –1 reward, $\gamma = 1$, random policy, 4×4, hai terminal corners. Iterative sweeps cho đến khi $V(s)$ hội tụ, phản ánh "độ ngắn" đường đi trung bình đến terminal.
5. **Hạn chế của DP**: Yêu cầu biết mô hình, không gian trạng thái nhỏ; nếu không, cần chuyển sang phương pháp mô hình-free hoặc hàm xấp xỉ.