2. Temporal-Difference Learning là gì?

- **Temporal-Difference (TD) Learning** kết hợp ưu điểm của hai phương pháp chính trong Reinforcement Learning:
  - Từ **Monte Carlo methods**: không cần mô hình môi trường (model-free) và tính toán trực tiếp trên trải nghiệm.
  - Từ **Dynamic Programming**: cập nhật liên tục (bootstrap) dựa trên giá trị ước tính ở bước kế tiếp.
- TD Learning cho phép **ước lượng hàm giá trị** $V(s)$ (hoặc $Q(s, a)$) ngay trong quá trình tương tác, mà không phải đợi toàn bộ episode kết thúc.

---

3. Temporal-Difference Error (Lỗi sai theo thời gian)

3.1. TD Target

- Tại mỗi bước t, khi agent đang ở trạng thái S_t, quan sát được reward R_{t+1} và chuyển sang trạng thái kế tiếp S_{t+1}, ta xây dựng một **mục tiêu (TD target)**:

$$\text{TD target}_t = R_{t+1} \; + \; \gamma \, V(S_{t+1}),$$

  trong đó
  - $\gamma \in [0, 1]$ là hệ số chiết khấu,
  - $V(S_{t+1})$ là giá trị ước tính hiện tại của trạng thái kế tiếp.

3.2. TD Error

- **Temporal-Difference Error** (ký hiệu δ_t) chính là độ chênh giữa **TD target** và **giá trị ước tính hiện tại** $V(S_t)$:

$$\delta_t = [R_{t+1} + \gamma \, V(S_{t+1})] \; - \; V(S_t).$$

- δ_t đo lường "bao nhiêu" giá trị $V(S_t)$ cần phải điều chỉnh để khớp với phần thưởng quan sát và giá trị của bước sau.

---

4. Thuật toán TD(0)

TD(0) là phiên bản đơn giản nhất của TD Learning, cập nhật **ngay lập tức** sau mỗi bước:

1. **Khởi tạo**:
   - Với mỗi trạng thái s, gán giá trị ban đầu $V(s)$ (thường là 0 hoặc ngẫu nhiên nhỏ).

2. **Với mỗi bước t trong quá trình tương tác**:
   - Quan sát (S_t, R_{t+1}, S_{t+1}).
   - Tính **TD error**:
   $$\delta_t = R_{t+1} + \gamma\, V(S_{t+1}) - V(S_t).$$
   - Cập nhật giá trị ước tính của S_t theo learning rate $\alpha \in (0, 1]$:
   $$V(S_t) \leftarrow V(S_t) + \alpha\, \delta_t.$$
   - Tiếp tục sang bước $t + 1$ cho đến khi kết thúc quá trình.
3. **Thuật toán lặp đi lặp lại** cho đến khi V hội tụ hoặc đạt giới hạn episode.

---

5. Policy Improvement với TD Learning

- Mặc dù slide chỉ đề cập sơ lược, nhưng TD(0) có thể mở rộng để **học hàm giá trị hành động** $Q(s, a)$, rồi dùng cho **policy improvement**.
- **Q-Learning** là một ví dụ off-policy TD, trong đó:
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a') - Q(S_t, A_t)].$$
- Agent sau đó **chọn hành động greedy** (hoặc epsilon-greedy) dựa trên Q.

---

6. Ví dụ minh họa đơn giản (Grid World)

Trong slide có **ví dụ Grid World** rất gọn:

1. **Môi trường**:
   - Một dãy ô thẳng (line world) với vị trí A (bắt đầu) và B (mục tiêu).
   - Agent có thể di chuyển **trái** hoặc **phải**.
   - Chỉ khi đến ô B thì nhận **reward = +1**; các bước đi khác reward = 0.
2. **Quá trình tương tác** (một episode mẫu):
   - Agent khởi đầu ở A, chọn đi phải → tới ô kế (S_{t+1}).
   - Quan sát reward $R = 0$ (chưa phải B) và giá trị V của ô mới.
   - Tính δ và cập nhật $V(A)$.
   - Tiếp tục đến khi vào B, nhận $R = +1$, tính δ ở bước cuối, cập nhật V cho trạng thái trước đó.
3. **Cách update cụ thể** (ví dụ minh họa):
   - Giả sử ban đầu $V(A) = 0,\ V(\text{ô kế}) = 0$.

- Bước 1: từ A → ô kế, $R_1 = 0$.

  $$\delta_0 = 0 + \gamma \cdot V(\text{ô kế}) - V(A) = 0 + 0 - 0 = 0$$

  ⇒ $V(A)$ không đổi.
- ...
- Cuối cùng khi đến B, giả sử từ ô trước B chuyển sang B, $R = 1$:

  $$\delta = 1 + \gamma \cdot V(B) - V(\text{ô trước}) = 1 + 0 - V(\text{ô trước}).$$

  Với $\alpha > 0$, update $V(\text{ô trước})$ dần lên gần 1.
- Qua nhiều episode, **giá trị ước tính** $V(A)$ và các ô giữa sẽ **tiệm cận** giá trị thực (kỳ vọng số bước để đến B).

---

7. Kết quả và Hội tụ

- Nhờ cập nhật bootstrap (dựa trên giá trị bước kế), TD(0) thường **hội tụ nhanh hơn** Monte Carlo, vì không phải đợi đến cuối episode mới update.
- So với Dynamic Programming, TD không cần biết mô hình môi trường.
- Kết hợp cả hai, TD Learning là một phương pháp **powerful** trong RL hiện đại.