Reinforcement Learning #5

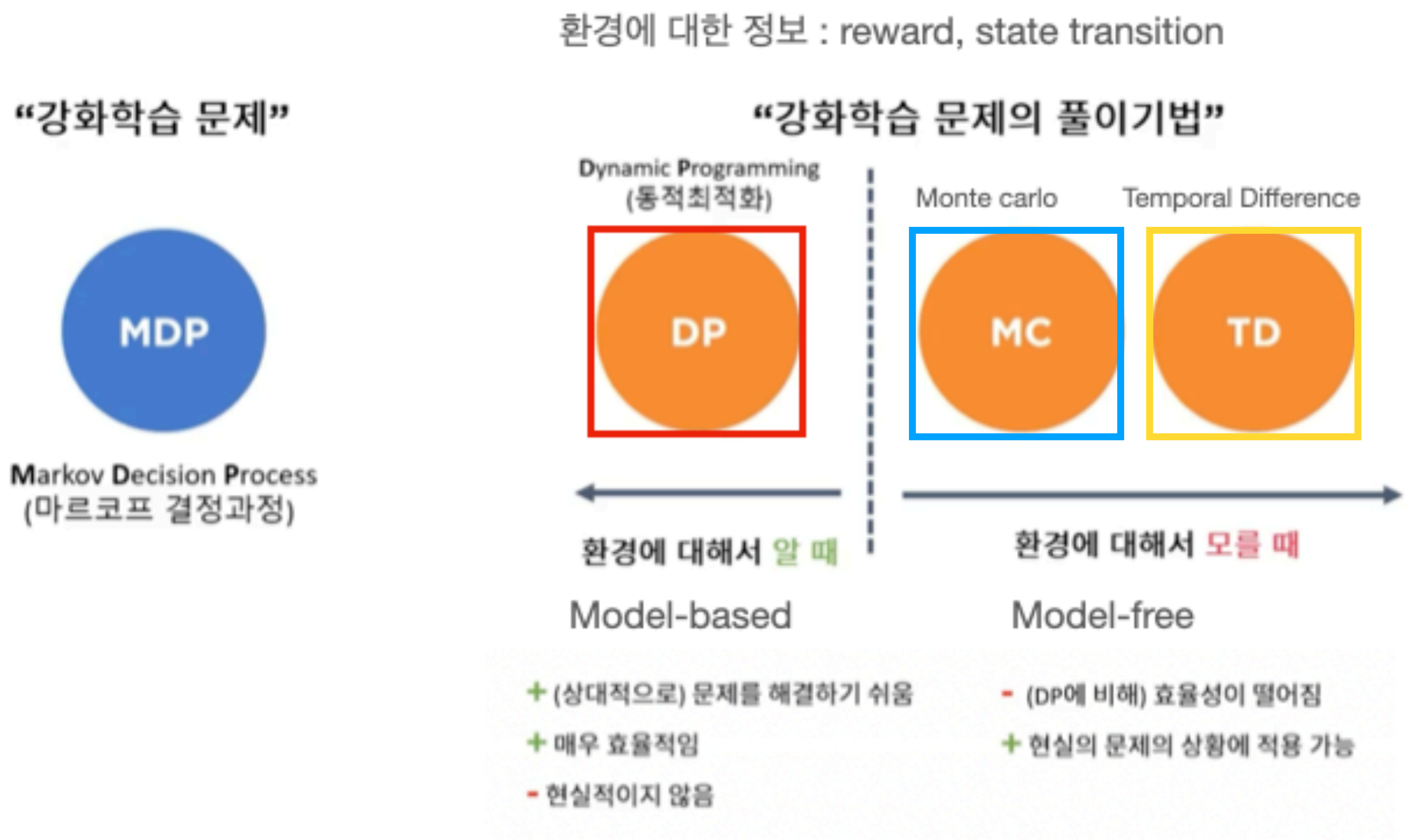
#Temporal Difference

정규헌

Reinforcement Learning

강화학습 문제

- 최적 정책를 찾는 것!
- **Model based : DP, Asynchronous DP**
  - **알고 있는 값**들을 활용해 특정 공식을 수렴할 때까지 반복해 최적해를 찾음
  - 알고있는 값 - 환경모델이 필요
- **Model free : MC, TD**
  - **MC** : 데이터를 활용해서 최적해를 찾음(불편추정량)
    - 각 state와 action 사이의 관계에 대한 정보는 전혀 활용되지 않음
  - **TD** : 알고있는 일부 경험과 함께 데이터를 활용
    - 각 state와 action 사이의 관계를 활용
    - 불편추정량이 아니기 때문에 오차가 발생할 수 있음



Temporal Difference

TD vs MC

TD 기법

- Episode가 종결되지 않아도 사용이 가능
- 편향이 존재
  - 편향으로 인해 시행횟수와 무관하게 오류가 생길 수 있음
  - 추정치의 분산이 낮아 적은 시행에도 좋은 추정치를 얻을 수 있음
- Markov Discision Process (MDP) 특성을 활용
  - MDP환경이 아니면 정확도가 떨어짐

MC 기법

- Episode가 종결되어야만 사용가능
- 편향 존재하지 않음 (불편추정량)
  - 분산이 높아 많은 시행이 필요
  - 시행횟수만 충분하다면 참 가치함수를 찾을 수 있음
  - 충분한 시행횟수를 얻기 위해 더 많은 시간이 필요
- Markov Discision Process (MDP) 특성 활용하지 않음
  - MDP 환경이 아니어도 정확한 추정이 가능

Temporal Difference

TD method

- TD 기법 자체가 MC기법과 유사
- MC는 state와 action 사이 관계의 정보를 활용하지 않음
  - state와 action 사이의 관계를 활용 - MDP
    - MDP : 현재 상태는 이전 상태에만 영향을 받는다
- **Temporal Difference method**
  - 바로 다음 step의 보상만을 활용 : TD(0)
  - V(s)를 TD target에 가까워지게 조정
  - V(s)와 TD target의 차이 : TD error

Incremental Monte Carlo

$$V(s) \leftarrow V(s) + \alpha\big(G_t - V(s)\big)$$

$$G_t \stackrel{\text{def}}{=} R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ \dots + \gamma^{T-1} R_T$$

Temporal Difference TD(0)

$$G_t \stackrel{\text{def}}{=} R_{t+1} + \gamma V(S_{t+1})$$

바로 다음 step의 Valuefunction만 더한다

TD target

$$R_{t+1} + \gamma V(S_{t+1})$$

TD error

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$

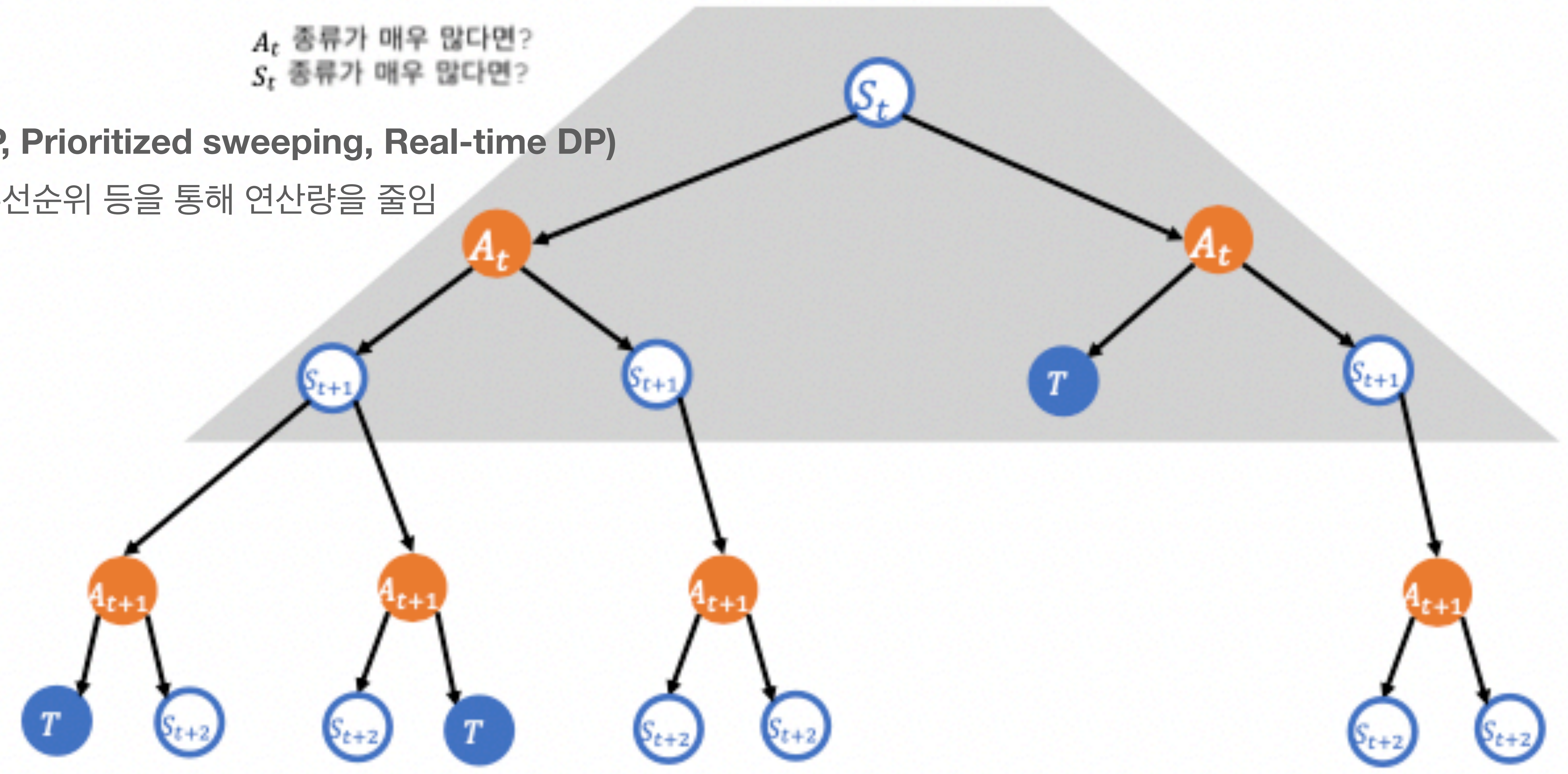
Temporal Difference

DP Full-width backup

모든 가능한 S,a에 대해서 동기적으로 업데이트 하는 과정을 거침

모든 가능한 S,a에 대해서 알고 있어야하고 그만큼 연산량이 매우 많아짐

DP 를 활용한 Policy evaluation : $V^{\pi}(S_t) \leftarrow \mathbb{E}_{\pi}[R_{t+1} + \gamma V^{\pi}(S_{t+1})]$



Asynchronous DP (In-place DP, Prioritized sweeping, Real-time DP)

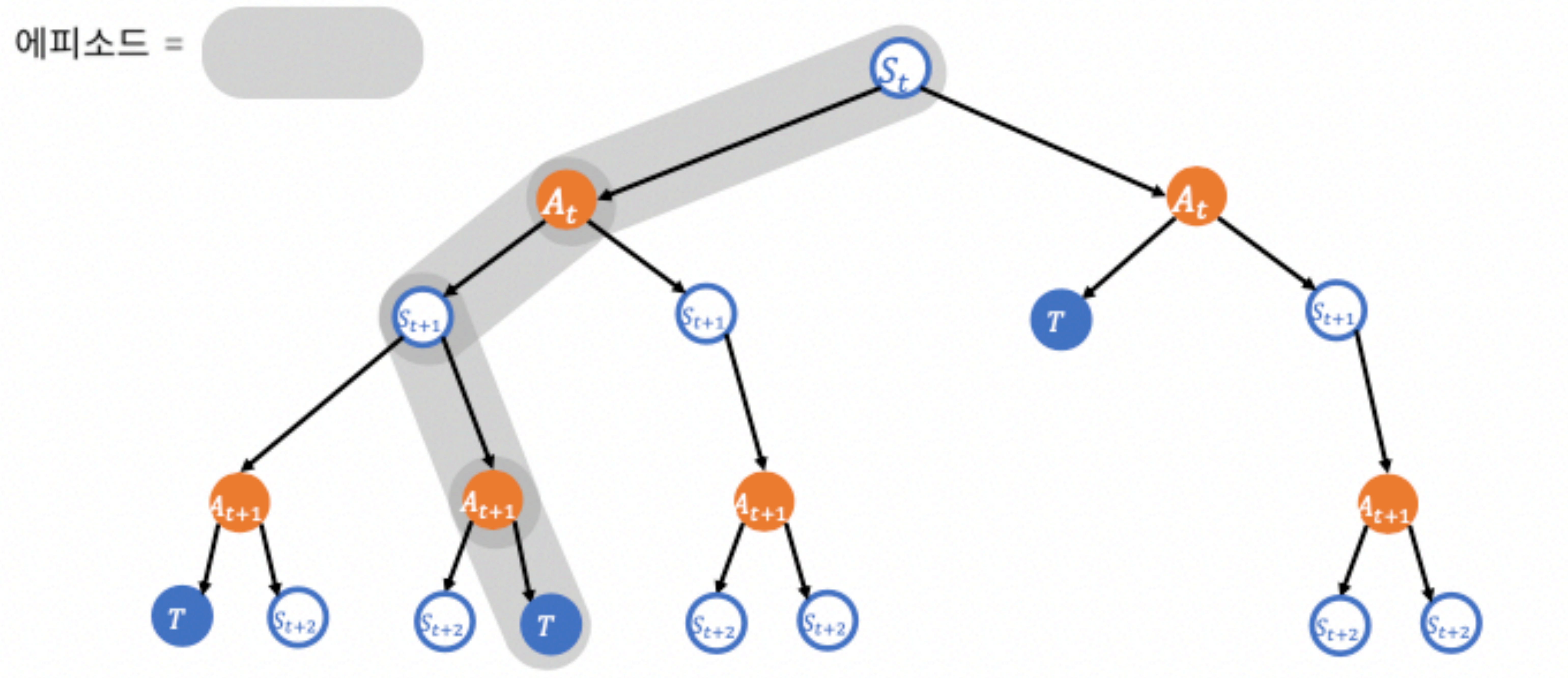
모두 연산하는 것이 아닌 순서, 우선순위 등을 통해 연산량을 줄임

Temporal Difference

MC Sample backup

에피소드 별로 탐색, 전체 state공간이 얼마나 크든 저장만 시키면 학습이 가능

MC Policy evaluation : $V(s) \leftarrow V(s) + \alpha\left(G_t - V(s)\right)$

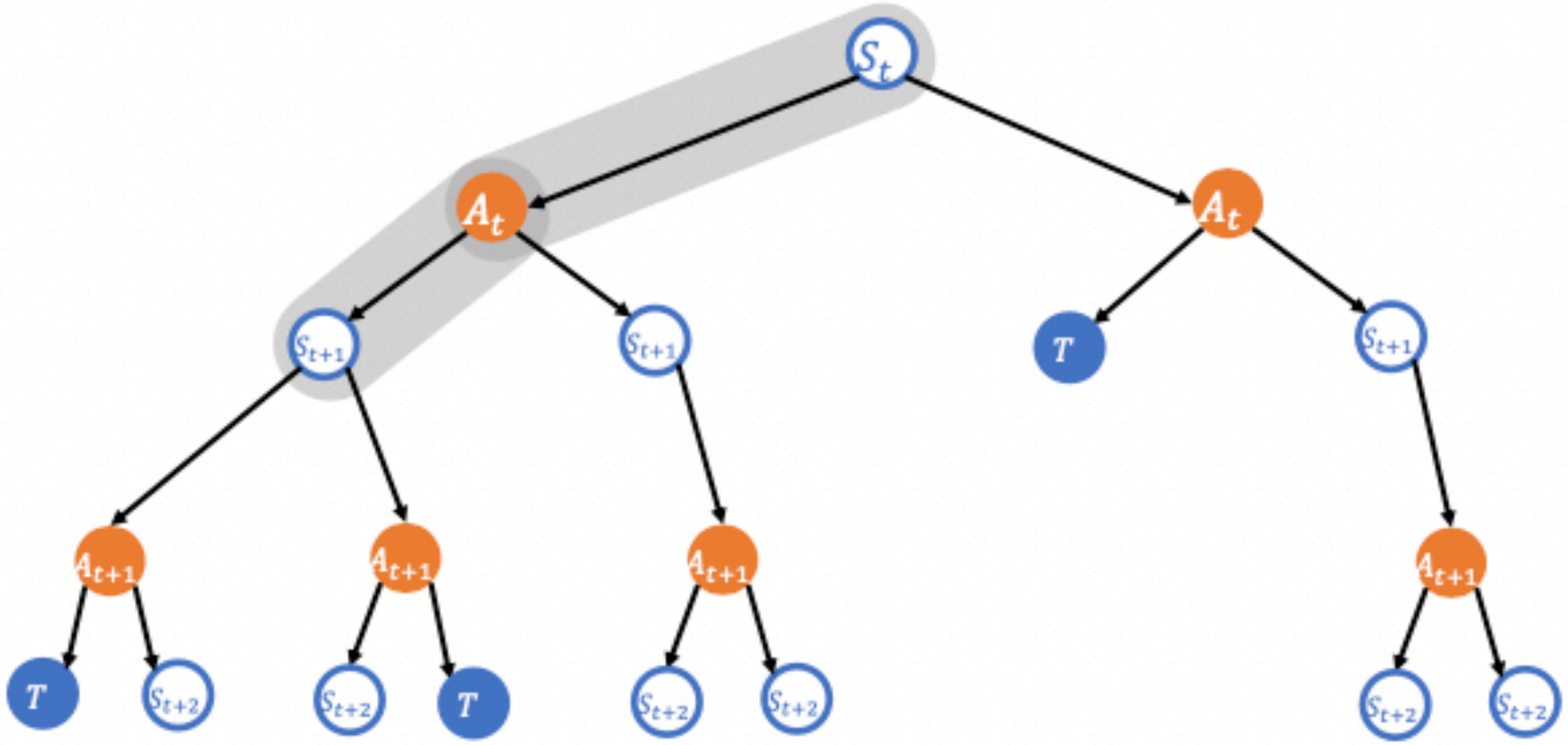


Temporal Difference

TD Sample backup

에피소드 별로 탐색, 전체 state공간이 얼마나 크든 저장만 시키면 학습이 가능 + 에피소드가 끝나지 않아도 가능(step조절이 가능)

TD Policy evaluation : $V(s) \leftarrow V(s) + \alpha\left(R_{t+1} + \gamma V(S_{t+1}) - V(s)\right)$



Temporal Difference

TD : n-step TD

$$V(s) \leftarrow V(s) + \alpha\left(G_t^{(n)} - V(s)\right)$$

Aka TD(0)

1-step TD : $G_t^{(1)} \stackrel{\text{def}}{=} R_{t+1} + \gamma V(S_{t+1})$: 1 스텝까지의 보상 (새로운 정보) + 가치함수 (알고 있는 정보)

2-step TD :$G_t^{(2)} \stackrel{\text{def}}{=} R_{t+1} + \gamma R_{t+2} + \gamma^2 V(S_{t+2})$: 2 스텝까지의 보상 (새로운 정보) + 가치함수 (알고 있는 정보)

3-step TD :$G_t^{(3)} \stackrel{\text{def}}{=} R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 V(S_{t+3})$

...

∞-step TD :$G_t^{(\infty)} \stackrel{\text{def}}{=} R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots + \gamma^{T-1} R_T$: 몬테카를로와 동일

$$G_t^{(n)} \stackrel{\text{def}}{=} \underbrace{\boxed{R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots + \gamma^{n-1} R_{t+n}}}_{\text{현재부터 n step까지 새로 알게된 정보}} + \underbrace{\gamma^n V(S_{t+n})}_{\text{원래 알고있던 정보}}$$

Temporal Difference

TD : n-step TD

$$V(s) \leftarrow V(s) + \alpha\left(G_t^{(n)} - V(s)\right)$$

G_t 의 추정치로서 다양한 n에 해당하는 $G_t^{(n)}$ 를 추산할 수 있다.

즉, 하나의 값을 추정하기 위해 여러가지 방식의 추정기법을 활용할 수 있다.

자연스러운 의문

서로 다른 추정치를 모두 사용하여 하나의 G_t 추정치를 만들 수는 없을까?

Temporal Difference

TD : n-step TD

산술평균으로 하면 안될까?　　적절한 scaling이 필요

$$G_t \stackrel{\text{def}}{=} \frac{1}{T}\left(G_t^{(1)} + G_t^{(2)} + G_t^{(3)} + G_t^{(4)} + G_t^{(5)} + \cdots + G_t^{(T)}\right)$$

안됨. MC에서도 말했듯이 불편 추정량이 아니기 때문

$G_t^{(1)}$: 1 step 정보만을 담고 있어 적은 분산을 가지지만, 편향이 매우 큼

$G_t^{(T)}$: 에피소드 전체 정보를 담고 있어 MC와 같음, 편향은 없지만 분산이 큼

$$G_t^{\lambda} \stackrel{\text{def}}{=} (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)}$$

$(0 \le \lambda \le 1)$

기하급수 $1, r, r^2, \ldots$

$$\sum_{n=0}^{N} \lambda^n = \frac{(1-\lambda^N)}{1-\lambda} \quad (\lambda \neq 1)$$

N 이 충분히 클 때, $\lambda^N \approx 0$

즉, $\sum_{n=0}^{N} \lambda^n \approx \frac{1}{1-\lambda}$

λ 를 바꿔 줌으로써, 분산-편향 관계를 조정가능.

- TD(0) ($\lambda = 0$ 일 때), $G_t^0 = G_t^{(1)}$ 이 됨. **1-step TD**와 동일
- TD(1) 매-방문 MC와 유사.

Exercise

Monte Carlo Policy evaluation

Vanilla MC PE

MC policy evaluation $V(s) \leftarrow \frac{S(s)}{N(s)}$

$S(s)$: 상태 s에 대한 G_t 들의 합
$N(s)$: 상태 s를 (처음) 방문한 횟수

General RL ‘**Agent-Environment Interaction**’ framework

반복 : 주어진 에피소드 수만큼

특정 에피소드 시작

반복 : 에피소드 내부

현재 state — 환경으로부터 현재 상태 관측

현재 action — agent의 정책함수에 따른 action 선택

다음 state, reward — 환경에 현재 action을 가함으로써 얻음

if 다음 상태 == 종결상태 :

Break

Policy Evaluation
Policy Improvement

선택적으로 매 episode마다 value function을 평가하거나 개선할 수 있음

```python
def run_episode(env, agent):
    env.reset()
    states = []
    actions = []
    rewards = []

    while True:
        state = env.observe()
        action = agent.get_action(state)
        next_state, reward, done, info = env.step(action)

        states.append(state)
        actions.append(action)
        rewards.append(reward)

        if done:
            break

    episode = (states, actions, rewards)
    agent.update(episode)
```

Exercise

1-step TD prediction

General RL ‘**Agent-Environment Interaction’** framework

반복 : 주어진 에피소드 수만큼

특정 에피소드 시작

반복 : 에피소드 내부

현재 state 환경으로부터 현재 상태 관측

현재 action agent의 정책함수에 따른 action 선택

다음 state, reward 환경에 현재 action을 가함으로써 얻음

if 다음 상태 == 종결상태 :

Break

Policy Evaluation
Policy Improvement
선택적으로 매 episode마다 value function을 평가하거나 개선할 수 있음

TD 1-step, TD(0)

$$V(s) \leftarrow V(s) + \alpha(G_t - V(s))$$

$$G_t = R_{t+1} + \gamma V(s_{t+1})$$

```
def run_episode(env, agent):
    env.reset()

    while True:
        state = env.observe()
        action = agent.get_action(state)
        next_state, reward, done, info = env.step(action)
        agent.sample_update(state=state,
                            action=action,
                            reward=reward,
                            next_state=next_state,
                            done=done)

        if done:
            break
```

state, action, reward 정보를 따로 저장하지 않아도 됨

MC와는 다르게 에피소드 진행중에도 update가 가능

Exercise

1-step TD prediction

$$G_t = R_{t+1} + \gamma V(s_{t+1})$$

$$V(s) \leftarrow V(s) + \alpha(G_t - V(s))$$

```
def sample_update(self, state, action, reward, next_state, done):
    # 1-step TD target
    td_target = reward + self.gamma * self.v[next_state] * (1 - done)
    self.v[state] += self.lr * (td_target - self.v[state])
```

done을 활용해 다음 상태가 terminal일 경우 reward가 0이 되게 update

TD 1-step, TD(0)

$$V(s) \leftarrow V(s) + \alpha(G_t - V(s))$$

$$G_t = R_{t+1} + \gamma V(s_{t+1})$$

```
def run_episode(env, agent):
    env.reset()

    while True:
        state = env.observe()
        action = agent.get_action(state)
        next_state, reward, done, info = env.step(action)
        agent.sample_update(state=state,
                            action=action,
                            reward=reward,
                            next_state=next_state,
                            done=done)

        if done:
            break
```

state, action, reward 정보를 따로 저장하지 않아도 됨

MC와는 다르게 에피소드 진행중에도 update가 가능

Exercise

n-step TD prediction

$$V(s) \leftarrow V(s) + \alpha(G_t - V(s))$$

$$G_t^n = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n})$$

```python
def update(self, episode):
    states, actions, rewards = episode
    ep_len = len(states) # 원래 episode의 길이를 알아야 update 정도를 계산할 수 있음

    states += [0] * (self.n_step + 1)  # append dummy states
    rewards += [0] * (self.n_step + 1)  # append dummy rewards
    dones = [0] * ep_len + [1] * (self.n_step + 1) # 원래 episode length + 추가된 length + 1 만큼

    kernel = np.array([self.gamma ** i for i in range(self.n_step)]) # i : 0 1 2 3 4 (n_step = 5)
    for i in range(ep_len):
        s = states[i] # 현재 상태
        ns = states[i + self.n_step] # 다음 상태
        done = dones[i]

        # compute n-step TD target
        # reward : 현재 시점부터 다음 state까지
        # kernel과 길이가 같으므로 positional wisely 계산 후 합
        g = np.sum(rewards[i:i + self.n_step] * kernel)
        g += (self.gamma ** self.n_step) * self.v[ns] * (1 - done)
        self.v[s] += self.lr * (g - self.v[s])
```

Done list를 만들어 마지막
n_step을 1로 변경

$$G_t^n = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n})$$

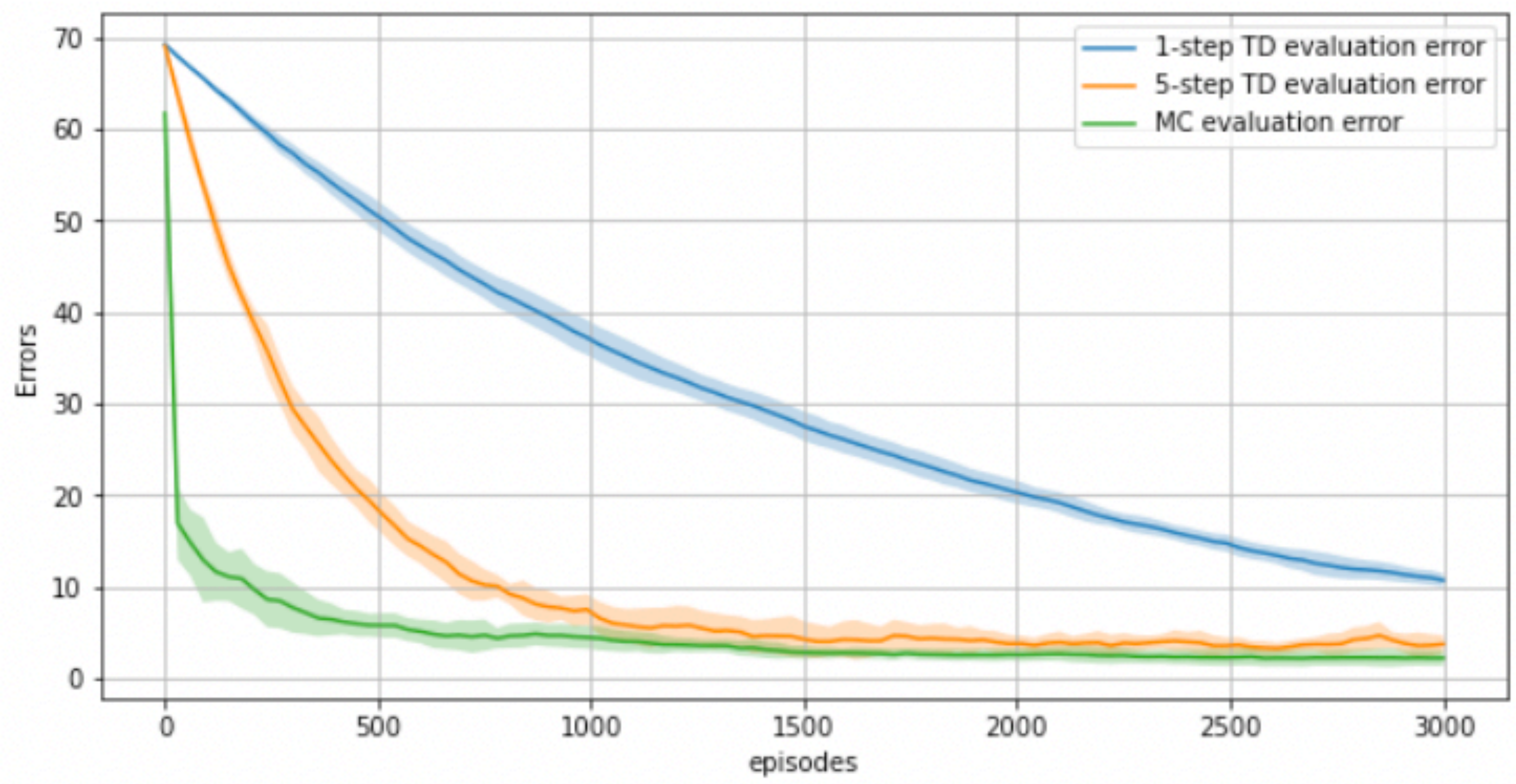
episode의 length가 random하기 때문

```python
# episodic한 형태로 구현해야함
# 기존 mc와 유사
def run_episode(env, agent):
    env.reset()
    states = []
    actions = []
    rewards = []

    while True:
        state = env.observe()
        action = agent.get_action(state)
        next_state, reward, done, info = env.step(action)

        states.append(state)
        actions.append(action)
        rewards.append(reward)

        if done:
            break

    episode = (states, actions, rewards)
    agent.update(episode)
```

✓ 0.2s

Exercise

1-step TD vs 5-step TD vs MC

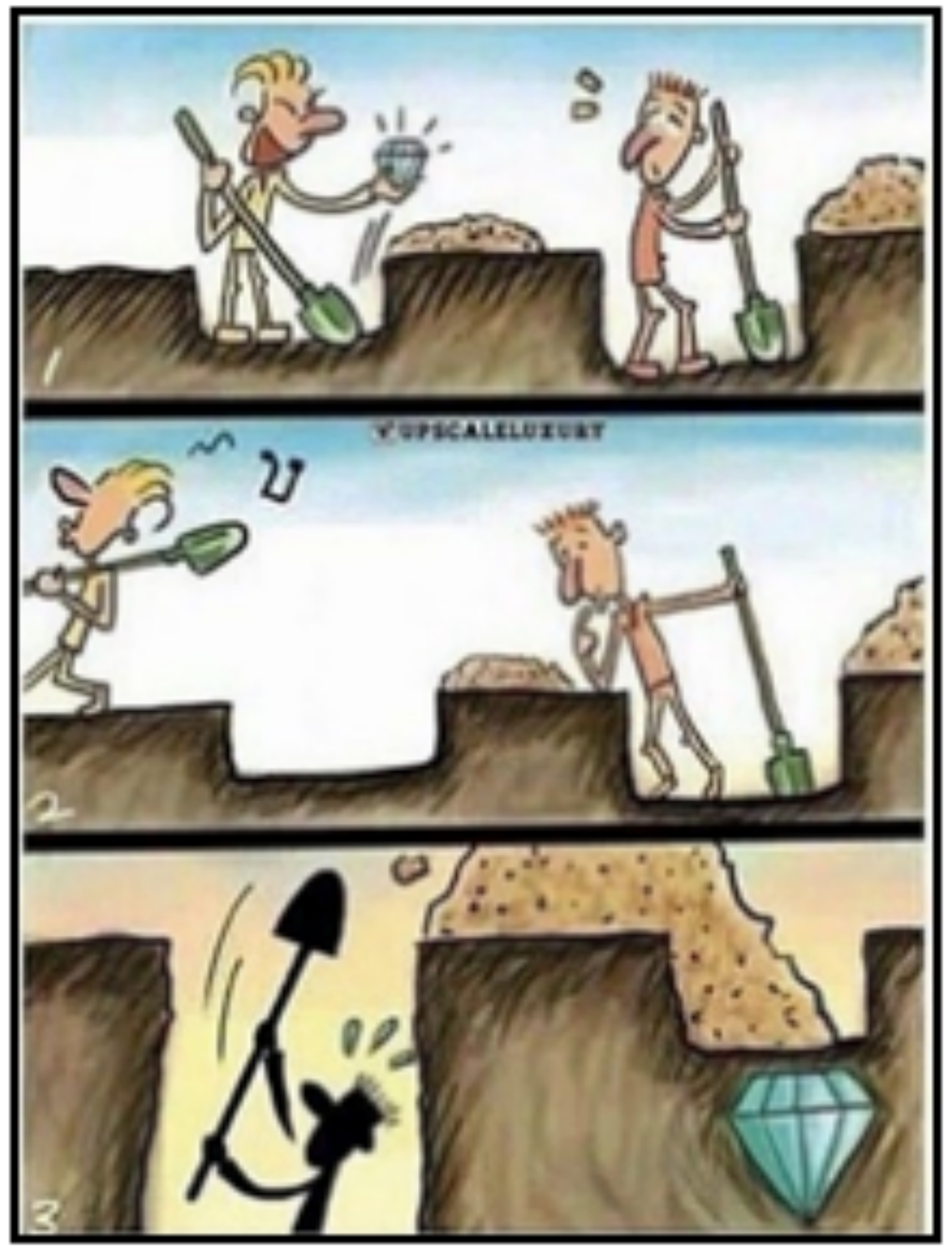


분산 : 1-step < 5-step < MC

error : 1-step > 5-step > MC

Finding Policy

TD든 MC든 구한건 결국 V



결국 V(s)를 구했으면 Q(s,a)를 구해야하고 그 값을 통해 policy π를 구해야함

- **Greedy action** : 점수가 가장 큰 쪽으로 움직임
    - 미래를 생각하지 않고 각 단계에서 가장 최선의 선택 (Exploitation)
    - Exploration이 충분하지 않아 최상의 결과가 나오기 힘듬
- **Epsilon-greedy** : 점수가 가장 큰 쪽으로 움직이되, **epsilon 확률값 0.2** 만큼은 random
    - 부족한 Exploration을 보충
- **Decaying epsilon-greedy** : epsilon 값을 0에 가깝게 점점 줄여나감
    - Exploration + Exploitation

Greedy policy

Epsilon greedy policy

(s, a) (처음) 때마다,

$$N(s,a) \leftarrow N(s,a) + 1$$

$$Q(s,a) \leftarrow Q(s,a) + \frac{1}{N(s,a)}\left(G_t - Q(s,a)\right)$$

GLIE 조건 : 모든 N(s, a)에 대해서 한 번씩 방문 할 수 있도록 epsilon을 학습진행에 따라 스케줄링

Greedy policy (탐욕적 정책) $\pi(a|s) = \begin{cases} 1, & if\ a = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, Q^{\pi}(s,a) \\ 0, & otherwise \end{cases}$

Learning rate 초기 값과 epsilon값을 어떻게 설정하느냐에 따라 성능에 영향을 크게 미침

ϵ-Greedy policy $\pi(a|s) = \begin{cases} \frac{\epsilon}{|\mathcal{A}|} + 1 - \epsilon, & if\ a = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, Q^{\pi}(s,a) \\ \epsilon/|\mathcal{A}|, & otherwise \end{cases}$ $|\mathcal{A}|$: 가능한 action 갯수

GLIE epsilon-greedy policy

$$\epsilon \leftarrow \frac{1}{k}$$

$$\pi \leftarrow \epsilon - greedy(Q)$$

감사합니다.