4. Temporal Difference(TD)

경희대학교

기계공학과 RCI 연구실
박보형

2025-2 이동로봇

목차

- Target Policy, Behavior Policy
- On-Policy, Off-Policy
- Temporal Difference(TD)

Target Policy, Behavior Policy

Target Policy

- Agent가 학습하고 싶은, 최종적으로 도달하고자 하는 Policy

$$\pi(a \mid s) = P(A_t = a \mid S_t = s, \text{target policy})$$

Behavior Policy

- Agent가 실제로 환경에서 데이터를 수집할 때 따르는 Policy
  - 지금 당장 어떤 Action을 취할지 결정하는 Policy

$$\mu(a \mid s) = P(A_t = a \mid S_t = s, \text{behavior policy})$$

On-Policy, Off-Policy

On-Policy

- Behavior-Policy와 Target-Policy가 동일
- 실제로 ε-greedy Policy로 Action을 선택하고, update도 같은 Policy를 기준으로 함

Off-Policy

- Behavior-Policy와 Target-Policy가 다름
- Behavior-Policy는 ε-greedy Policy : Explorational Action 선택
- Target-Policy는 greedy Policy : 항상 Optimal Action을 취한다는 가정으로 update

Temporal Difference(TD)

TD의 필요성

- Bellman Equation으로 Return이 없어도 One-Step으로 계산할 수 있게 되었다.
  - 그런데 Transition Probability를 몰라서 Value-Function을 직접 계산할 수 없다.

- Value-Function을 모른다면 MC를 이용해서 실제로 받는 Return의 평균으로 추정하자.

- BE와 MC를 합친 것이 TD
  - Value-Function은 모르지만 MC처럼 현재 추정치를 이용해서 BE처럼 한 단계만 보고 업데이트 해보자.

Temporal Difference(TD)

TD(0)

- TD error

$$\delta_t = r_{t+1} + \gamma Q(s', a') - Q(s, a)$$

  - Prediction Q(s, a)와 New Target의 차이
  - 지금 내 Q값 예측이 얼마나 틀렸는지 알려주는 오차 신호

- Bootstrapping

$$r_{t+1} + \gamma Q(s', a')$$

  - Return을 기다리지 않고 이미 가지고 있는 추정 값을 이용해서 새로운 추정 값을 업데이트하는 것
  - 진짜 정답이 아니라 내가 생각하는 임시 정답(bootstrapping Target)

- TD(0)

$$Q(s, a) \leftarrow Q(s, a) + \alpha \Big[r_{t+1} + \gamma Q(s', a') - Q(s, a) \Big]$$

  - Bootstrapping Target으로 한 Step의 TD error를 계산하고 학습률 α만큼 업데이트

Temporal Difference(TD)

SARSA(On-Policy TD)

- State – Action – Reward – State' – Action'
  - 현재 State를 받아서 현재 Policy가 선택한 Action을 취하고, 다음 State와 Reward를 관찰한 뒤, 다시 "같은 Policy"로 다음 행동 A'를 선택하고 이의 Q값을 이용해 아래 식으로 업데이트

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left[r_{t+1} + \gamma Q(s',a') - Q(s,a)\right]$$

Initialize $Q(S,A)$, all $S \in \mathcal{S}$, $A \in \mathcal{A}(S)$, arbitrarily and $Q(\text{terminal},\cdot) = 0$
Repeat (for each episode):
  Initialize S
  Choose A from S using policy derived from Q (e.g., ϵ-greedy)
  Repeat (for each step of episode):
    Take action A; observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ϵ-greedy)
    $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma Q(S',A') - Q(S,A)\right]$
    $S \leftarrow S'$; $A \leftarrow A'$
  Until S is terminal

Sarsa

behavior policy (on-policy)
$\ldots, S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}, \ldots$

Temporal Difference(TD)

Q-Learning(Off-Policy TD)

- State – Action – Reward – State' – max Action'
  - 현재 State를 받아서 현재 Behavior Policy로 Action을 선택하고, 다음 State와 Reward 관찰
  - Behavior Policy와는 무관하게 greedy한 Target Policy를 이용해서 Q값 업데이트

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right]$$

Initialize $Q(S, A)$, all $S \in \mathcal{S}$, $A \in \mathcal{A}(S)$, arbitrarily and $Q(\text{terminal}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q (e.g., ϵ-greedy)
    Take action A; observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\left[R + \gamma \max_a Q(S', a) - Q(S, A)\right]$
    $S \leftarrow S'$
  Until S is terminal

Q-learning

not behavior policy (off-policy)

$\ldots, S_t, A_t, R_{t+1}, S_{t+1}, \ldots$

Temporal Difference(TD)

TD(λ)

n-step TD

$$G_t^{(n)} = r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n Q(s_{t+n}, a_{t+n})$$

- MC는 Episode 끝까지 봐야 하고, TD는 바로 다음 Step만 보고 Target Update
- MC와 Target의 중간단계로 n step까지만 보고 싶을 때 사용

TD(λ)

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

- n-step TD에서 Reward의 합은 n step을 고려하는데, Q는 마지막 step으로만 결정됨
- Q도 n step으로 고려하자 : G(1)부터 G(n)까지의 가중합을 사용
- 이때 λ는 조절 가능한 파라미터
  - λ = 0 : TD(0)
  - λ = 1 : MC

REINFORCE(Policy Gradient)

- 지금까지는 Value Function을 예측해서 Action의 선택 기준으로 개선하는 방식으로 Policy를 개선
  - 가장 큰 문제점은 Action Space가 연속적일 때 Value 기반의 방법을 사용할 수 없음
    - Continuous State, Action이 필요한 환경에서 Q-Table을 작성 불가능
    - Q-Table을 Neural Network로 대체한 DQN으로 Continuous State로는 확장 가능

- Policy Gradient 방식에서는 Behavior Policy를 확률적으로 모델링
  - Continuous Action을 출력할 수 있음
  - 가장 기본적인 방법이 REINFORCE

감사합니다