

Reinforcement Learning Basic

Week3. Bellman Equation & Tabular Learning

박준영

Hanyang University
Department of Computer Science

지난 시간

- 강화학습 문제는 MDP를 정의해서 푼다.
- 강화학습 알고리즘의 분류
- Cross Entropy Method

1 Bellman Equation

2 Dynamic Programming

3 Tabular Learning

Review: Returns, Value Function

강화학습의 목표를 반환값(Returns)을 통해 수학적으로 표현할 수 있었다.

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

위 반환값을 근사하는 함수를 가치함수라 하고, 다음 두 종류의 가치함수가 존재한다.

$$\begin{aligned}v_{\pi}(s) &= \mathbb{E}_{\pi} [G_t | S_t = s] \\q_{\pi}(s, a) &= \mathbb{E}_{\pi} [G_t | S_t = s, A_t = a]\end{aligned}$$

Review: Returns, Value Function

episodic 문제에서 $T = 5$ 일 때 각 반환값은 다음과 같다. ($t = 1, 2, 3, 4, 5$)

$$G_1 = R_2 + \gamma R_3 + \gamma^2 R_4 + \gamma^3 R_5 + \gamma^4 R_6$$

$$G_2 = R_3 + \gamma R_4 + \gamma^2 R_5 + \gamma^3 R_6$$

$$G_3 = R_4 + \gamma R_5 + \gamma^2 R_6$$

$$G_4 = R_5 + \gamma R_6$$

$$G_5 = R_6$$

Review: Returns, Value Function

Q) $T = 5$ 인데 왜 R_6 가 존재하는 이유?

A) t 에서의 행동에 대한 보상은 $t + 1$ 일 때 받기 때문.

```
state = env.reset()

while True:
    action = generate_action(state)

    next_state, reward, done, _ = env.step(action)
```

벨만 방정식(Bellman Equation)

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots$$

$$G_t = R_{t+1} + \gamma (R_{t+2} + \gamma R_{t+3} + \cdots)$$

$$G_t = R_{t+1} + \gamma G_{t+1}$$

∴ 반환값은 그 다음 timestep의 반환값으로 표현할 수 있다.

벨만 방정식(Bellman Equation)

이를 가치함수의 정의에 대입해보자.

$$v_{\pi}(s) = \mathbb{E}_{\pi} [G_t | S_t = s]$$

$$v_{\pi}(s) = \mathbb{E}_{\pi} [R_{t+1} + \gamma R_{t+2} + \cdots | S_t = s]$$

$$v_{\pi}(s) = \mathbb{E}_{\pi} [R_{t+1} + \gamma (R_{t+2} + \gamma R_{t+3} + \cdots) | S_t = s]$$

$$v_{\pi}(s) = \mathbb{E}_{\pi} [R_{t+1} + \gamma G_{t+1} | S_t = s]$$

벨만 방정식(Bellman Equation)

가치함수의 정의를 다시 살펴보면...

$$v_{\pi}(s) = \mathbb{E}_{\pi} [G_t | S_t = s]$$

따라서 가치함수는 다음과 같이 근사할 수 있다.

$$v_{\pi}(s) = \mathbb{E}_{\pi} [R_{t+1} + \gamma v_{\pi}(S_{t+1}) | S_t = s]$$

$$v_{\pi}(s) = \sum_{a \in \mathbf{A}} \pi(a|s) \left[R_{t+1} + \gamma \sum_{s' \in \mathbf{S}} P_{s \rightarrow s'}^a v_{\pi}(s') \right]$$

위 방정식을 **벨만 방정식**이라고 한다.

Example

현재 상태 $S_t = s$ 에 대하여 다음과 같은 상황을 가정해보자.

- $\forall a \in \mathbf{A}, \pi(a|s) = \frac{1}{4}, \gamma = 0.9$
- 상하좌우로 이동하면 해당 위치로 상태가 전이될 확률은 1

- $P_{s \rightarrow s_{\text{up}}}^{\text{up}} = 1$

- $P_{s \rightarrow s_{\text{down}}}^{\text{down}} = 1$

- $P_{s \rightarrow s_{\text{left}}}^{\text{left}} = 1$

- $P_{s \rightarrow s_{\text{right}}}^{\text{right}} = 1$

- 각 상태에 대한 가치 함수의 값은 다음과 같다.

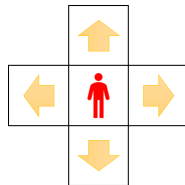
- $v_{\pi}(s_{\text{up}}) = 0$

- $v_{\pi}(s_{\text{down}}) = 0.5$

- $v_{\pi}(s_{\text{left}}) = 0$

- $v_{\pi}(s_{\text{right}}) = 1$

- 아래로 이동할 때는 1의 보상을, 나머지는 0의 보상을 준다.



Example

이제 현재 상태 $S_t = s$ 에 대한 가치함수의 값을 계산해보자.

- $A_t = \text{up일 때: } \pi(\text{up}|s) [R_{t+1} + \gamma v_\pi(s_{\text{up}})] = \frac{1}{4} \times (0 + 0.9 \times 0) = 0$
- $A_t = \text{down일 때: } \pi(\text{down}|s) [R_{t+1} + \gamma v_\pi(s_{\text{down}})] = \frac{1}{4} \times (1 + 0.9 \times 0.5) = 0.3625$
- $A_t = \text{left일 때: } \pi(\text{left}|s) [R_{t+1} + \gamma v_\pi(s_{\text{left}})] = \frac{1}{4} \times (0 + 0.9 \times 0) = 0$
- $A_t = \text{right일 때: } \pi(\text{right}|s) [R_{t+1} + \gamma v_\pi(s_{\text{right}})] = \frac{1}{4} \times (0 + 0.9 \times 1) = 0.225$

$$\therefore v_\pi(s) = 0 + 0.3625 + 0 + 0.225 = 0.5875$$

최적 정책과 최적 가치 함수

Definition

두 정책 π, π' 에 대하여 $\forall s \in \mathbf{S}, v_{\pi}(s) \geq v_{\pi'}(s)$ 를 만족하면 정책 π 가 정책 π' 보다 좋으며, $\pi \geq \pi'$ 이라 표기한다.

특히 $\forall \pi', \pi \geq \pi'$ 을 만족하면 정책 π 를 최적 정책(Optimal Policy, π_*)라고 한다.

$v_{\pi}(s) \geq v_{\pi'}(s)$ 는 어떤 상황 s 에서 정책 π 를 따랐을 때 정책 π' 을 따랐을 때보다 더 높은 보상을 얻을 수 있음을 의미한다.

최적 정책과 최적 가치 함수

Definition

최적 정책이 갖는 상태 가치 함수를 최적 상태 가치 함수(Optimal State-Value Function)이라 한다.

$$v_*(s) = \max_{\pi} v_{\pi}(s)$$

최적 정책과 최적 가치 함수

Definition

최적 정책이 갖는 행동 가치 함수를 최적 행동 가치 함수(Optimal Action-Value Function)이라 한다.

$$q_*(s, a) = \max_{\pi} q_{\pi}(s, a)$$

최적 행동 가치 함수는 다음과 같이 표현할 수도 있다.

$$q_*(s, a) = \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a]$$

벨만 최적 방정식(Bellman Optimal Equation)

v_* 도 결국 정책 π_* 에 대한 가치 함수이므로 벨만 방정식을 만족해야한다.

$$v_*(s) = \max_{a \in \mathbf{A}} q_{\pi_*}(s, a)$$

$$v_*(s) = \max_a \mathbb{E}_{\pi_*} [R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a]$$

$$v_*(s) = \max_a \mathbb{E} [R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a]$$

위 방정식을 벨만 최적 방정식이라 한다.

벨만 최적 방정식(Bellman Optimal Equation)

q_* 를 구하기 위한 벨만 최적 방정식은 다음과 같다.

$$q_*(s, a) = \mathbb{E}[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a) | S_t = s, A_t = a]$$

동적 프로그래밍(Dynamic Programming)

지금까지 가치 함수를 통해 최적 정책을 구하는 방법에 대해 알아보았다.
→ 그렇다면 가치 함수(v_π)는 어떻게 구할 수 있을까?

Dynamic Programming

: 큰 문제를 작은 문제로 나누어 푸는 방식

- 겹치는 연산을 활용해 총 연산량을 줄일 수 있다.
- MDP가 완벽하게 주어졌을 때 최적 정책을 계산할 수 있다.

정책 반복(Policy Iteration)

- 1 정책 평가(Policy Evaluation)
- 2 정책 향상(Policy Improvement)
- 3 1번으로 돌아간다.

$$\pi_0 \rightarrow v_{\pi_0} \rightarrow \pi_1 \rightarrow v_{\pi_1} \rightarrow \pi_2 \rightarrow \cdots \rightarrow \pi_* \rightarrow v_*$$

정책 평가(Policy Evaluation)

벨만 방정식을 이용해 다음의 갱신 규칙을 사용한다.

$$v_{k+1}(s) = \mathbb{E}_{\pi} [R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s]$$

이때 $k \rightarrow \infty$ 이면 v_k 는 v_{π} 에 수렴한다.

정책 향상(Policy Improvement)

새 정책이 좋은지 나쁜지를 어떻게 알 수 있을까?

→ 행동 가치 함수를 사용한다. ($q_{\pi}(s, a) \geq v_{\pi}(s)$ 이면 새로운 정책이 좋다는 것을 의미.)

따라서 다음과 같이 정책을 변경한다.

$$\pi'(s) = \operatorname{argmax}_a q_{\pi}(s, a)$$

정책 반복 알고리즘

$\forall s \in \mathbf{S}$, 임의로 $v(s) \in \mathbb{R}$ 와 $\pi(s) \in \mathbf{A}$ 설정.

▷ 1. 초기화

루프:

$$\nabla \leftarrow 0$$

$\forall s \in \mathbf{S}$ 에 대한 루프:

$$v \leftarrow v(s)$$

$$v(s) \leftarrow \sum_{s'} P_{s \rightarrow s'}^{\pi(s)} [R_{t+1} + \gamma v(s')]$$

$\nabla < \theta$ 를 만족할 때까지 반복

▷ 2. 정책 평가

정책 반복 알고리즘

안정적 정책 \leftarrow true

모든 $s \in \mathbf{S}$ 에 대해:

 이전 행동 $\leftarrow \pi(s)$

$\pi(s) \leftarrow \operatorname{argmax}_a \sum_{s'} P_{s \rightarrow s'}^a [R_{t+1} + \gamma v(s')]$

 이전 행동 $\neq \pi(s)$ 이면, 안정적 정책 \leftarrow false

안정적 정책이 false이면 2번 과정으로 되돌아감.

return $\pi \approx \pi'$

▷ 3. 정책 향상

가치 반복(Value Iteration)

- 정책 반복은 정책 평가를 매 주기마다 해야하는 단점이 존재한다.
- 정책 평가는 계산량이 매우 크다.
(심지어 무한히 계산해야만 수렴을 보장할 수 있다.)
→ 중간에 정책 평가를 끊어보자.

가치 반복(Value Iteration)

- 현재 가치함수가 최적이라 가정한다.
- 틀린 가정이지만 언젠가는 최적 정책을 구할 수 있기 때문에 문제가 되지 않는다.

$$v_{k+1}(s) = \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s, A_t = a]$$

동적 프로그래밍의 한계

■ 차원의 저주

- 상태의 개수가 늘어나면 계산할 양이 기하급수적으로 증가.
- 대부분의 환경은 상태의 개수가 엄청나게 많음.
(ex) 바둑: 약 10^{171} , 로봇제어: ∞

■ 환경에 대한 완벽한 정보 필요

- 많은 경우 환경에 대한 정보를 정확하게 알 수 없다.
(ex) 블랙잭에서 보유한 카드의 숫자의 총합이 14이고 패스할때 승리할 확률은?

→ 환경과의 상호작용을 통해 경험을 바탕으로 학습하는 방법론 = 강화학습

표를 이용한 강화학습(Tabular Learning)

- 강화학습은 함수를 찾아나가는 과정이다.
(ex) $v(s)$, $q(s, a)$, $\pi(a|s)$
- 표를 이용해 함수를 표현할 수 있다.

$X = x$	1	2	3	4
$P(X = x)$	1/5	2/5	1/5	1/5

시간차(Temporal Difference) 학습

얻은 평균을 이용해 가치함수를 만드는 것을 생각해보자.

$$\begin{aligned}v_{n+1}(S_t) &= \frac{1}{n} \sum_{i=1}^n G_t^i \\&= \frac{1}{n} \left[G_t^n + \sum_{i=1}^{n-1} G_t^i \right] \\&= \frac{1}{n} \left[G_t^n + (n-1) \frac{1}{n-1} \sum_{i=1}^{n-1} G_t^i \right]\end{aligned}$$

(1)

시간차(Temporal Difference) 학습

$$\begin{aligned} &= \frac{1}{n} [G_t^n + (n-1)v_n(S_t)] \\ &= \frac{1}{n} [G_t^n + nv_n(S_t) - v_n(S_t)] \\ &= v_n(S_t) + \frac{1}{n} [G_t^n - v_n(S_t)] \end{aligned}$$

따라서 $v(S_t) \leftarrow v(S_t) + \alpha [G_t - v_n(S_t)]$ 의 가치 함수 갱신 식을 만들 수 있다.

시간차(Temporal Difference) 학습

- 환경 모델 없이 경험만으로 학습이 가능하다.
- 추정값을 바탕으로 추정값을 갱신한다. (부트스트랩)

$$v(S_t) \leftarrow v(S_t) + \alpha [R_{t+1} + \gamma v(S_{t+1}) - v(S_t)]$$

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


Q-Learning

다음의 갱신 규칙을 이용해 행동 가치 함수를 갱신한다.

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

- 자신이 따르는 정책에 상관없이 최적 행동 가치 함수를 직접적으로 근사한다.
- 충분히 갱신을 반복하면 q_* 에 수렴한다.

Q-Learning



```
def value_update(self, s, a, r, next_s):  
    best_v, _ = self.best_value_and_action(next_s)  
  
    new_v = r + GAMMA * best_v  
    old_v = self.values[(s, a)]  
    self.values[(s, a)] = old_v * (1-ALPHA) + new_v * ALPHA
```