2장. 강화학습 기초1 : MDP와 벨만 방정식

Tags

keyword : 벨만 방정식, 순차적 행동 결정 문제, MDP, 가치함수

MDP

- 강화학습 : 순차적으로 행동을 계속 결정해야 하는 문제를 푸는 것.
- MDP : 순차적으로 결정해야 하는 문제를 수학적으로 표현
- MDP의 구성요소 : **상태, 행동, 보상 함수, 상태 변환 확률(State Transition Probability), 할인율(Discount Factor)**
- 사람이라면 풀고자하는 문제에 대해 파악하고자 한다. ⇒ 에이전트에게 사용자가 문제를 정의해주어야 한다. 문제를 잘못 정의하면 에이전트가 학습을 못 할 수도 있으므로 가장 중요한 단계이다. 에이전트를 구현하는 사람은 학습하기에 많지도 않고 적지도 않은 적절한 정보를 에이전트가 알 수 있도록 문제를 정의해야 한다.

상태

- S : 에이전트가 관찰 가능한 상태의 집합
- S_t : 시간 t일 때 상태
- 시간 t에 에이전트가 있을 상태는 전체 상태 중에 하나가 될 것이다. 시간 t까지 에이전트가 움직이는 것을 임의 실험이라고 본다면 시간 t에 에이전트가 있을 상태는 확률 변수가 된다. 임의의 실험에서 나오는 변수는 자신이 나타날 확률값을 가지고 있다.

행동

- A : 상태 S_t에서 할 수 있는 가능한 행동의 집합.

- A_t : 어떤 t라는 시간에 집합 A에서 선택한 행동.
- 상태 변환 확률: 에이전트가 특정 행동을 했을 때 어디로 이동할지 결정하는 것.

보상함수

- $r(s,a) = E(R_{t+1}|S_t = s, A_t = a)$: 시간 t에서 상태가 $S_t = s$ 이고, 행동이 $A_t = a$일 때 에이전트가 받을 보상.
- 보상은 에이전트가 알고 있는 것이 아니라 환경이 알려주는 것이기 때문에 보상을 받는 것은 t+1 시점이고, 어떤 정확한 값이 아니라 나오게 될 숫자에 대한 예상이므로 기댓값 형태로 나타내진다.

상태 변환 확률

- 상태의 변화에는 확률적인 요인이 들어가고 이를 수치적으로 표현한 것이 상태 변환 확률이다.
- $P^a_{SS} = P[S_{t+1} = s|S_t = s, A_t = a]$
- 보상과 마찬가지로 에이전트가 알지 못하는 값으로서 에이전트가 아닌 환경의 일부이다. 상태 변환 확률은 환경의 모델이라고도 부른다.

할인율

- 에이전트는 항상 현재에 판단을 내리기 때문에 현재에 가까운 보상일수록 더 큰 가치를 가진다.
- 이자 : 나중에 받을 보상을 현재의 보상과 같게 만드는 개념. 이자는 나중에 받을 보상에 추가적인 보상을 더해 현재의 보상과 같게 한다.
- 즉, 같은 보상이라면 나중에 받을수록 가치가 줄어든다.
- 할인율(discount factor) : $\gamma \in$ [0,1]
    - 할인: 미래의 가치를 현재의 가치로 환산
    - 만약 시간 t로부터 k가 지난 후에 보상을 R_{t+K}라 한다면 현재 그 보상의 가치는 $\gamma^{k-1}R_{t+k}$가 된다.

정책

- 모든 상태에서 에이전트가 할 행동
- 상태가 입력으로 들어오면 행동을 출력으로 내보내는 일종의 함수
  - 단 하나의 행동만을 나타낼 수도 있고, 확률적으로 나타낼 수도 있다.
  - 최적 정책 : 각 상태에서 단 하나의 행동만을 선택.
- 에이전트가 학습하고 있을 때는 정책이 하나의 행동만을 선택하기 보다는 확률적으로 여러 개의 행동을 선택할 수 있어야 한다.
- $\pi(a|s) = P(A_t = a|S_t = s)$: 시간 t에 s에 에이전트가 있을 때 a를 할 확률
- 정책만 가지고 있으면 에이전트는 모든 상태에서 자신이 해야 할 행동을 알 수 있다. 하지만 강화학습을 통해서 얻고자 하는 것은 '최적 정책'이다. 최적 정책을 얻기 위해서는 현재의 정책보다 더 좋은 정책을 학습해야 한다.

정리

- MDP를 통해 순차적 행동 결정 문제를 정의
- 에이전트는 현재 상태에서 앞으로 받을 보상을 고려하여 행동을 결정
- 환경은 에이전트에게 실제 보상과 다음 상태를 알려줌
- 이 과저을 반복하면서 앞으로 받을 보상을 학습함
- 이 때 앞으로 받을 것이라 예상하는 보상을 **가치함수(value function)**이라 함.
- 이 과정에서 에이전트느 실제로 받은 보상을 토대로 가치함수와 정책을 바꿔나가고 이러한 학습 과정을 충분히 반복하면 가장 많은 보상을 받게 하는 정책을 학습할 수 있음.

가치함수

가치함수

- 에이전트는 어떻게 최적 정책을 찾는가? 현재 상태에서 앞으로 받을 보상들을 고려해서 선택해야 좋은 선택을 할 수 있다.
- **가치 함수** : 앞으로 받을 보상에 대한 개념을 다룸.
- 현재 시각 t로부터 에이전트가 행동을 하면서 받을 보상들을 합하면 $R_{t+1} + R_{t+2} + ...$ 로 적을 수 있다. 보상은 행동을 했을 때가 아닌 그 다음 타임스텝에 받기 때문에 시간 t에 행동을 해서 받는 보상은 R_{t+1}이다. 이때 쓴 보상 R은 확률변수이다.
- 하지만 위에처럼 보상을 단순히 더하기만 하면 아래의 세 가지 문제가 생긴다.
  1. 에이전트 입장에서 지금 받은 보상이나 미래에 받는 보상이나 똑같이 취급한다. 할인하지 않는다면 에이전트가 보게 되는 보상의 합은 단순한 덧셈이다.
  2. 100이라는 보상을 1번 받는 것과 20이라는 보상을 5번 받는 것을 구분할 방법이 없다.
  3. 시간이 무한대라고 생각한다면 시간마다 0.1을 받는 경우와 1을 받는 경우 모두 결국 합이 무한대가 된다. 즉, 수치적으로 두 경우를 구분할 수 없다.
- 이러한 문제 때문에 에이전트는 단순한 보상의 합으로는 시간 t에 있었던 상태가 어떤 가치를 가지는지 판단하기 어렵다.
  - 할인율 : 미래의 보상을 현재의 보상으로 변환하기 위해 미래에 받은 보상에 곱해주는 0과 1 사이의 값.
  - 반환값(return) : 할인율을 적용한 보상들의 합. 에이전트가 실제로 환경을 탐험하며 받은 보상의 합.

    $$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ...$$
  - 이 책에서는 에이전트와 환경이 유한한 시간동안 상호작용 하는 경우만 다룬다. 이렇게 유한한 에이전트와 환경의 상호작용을 **에피소드**라고 부른다. 에피소드에서는 에피소드를 끝낼 수 있는 마지막 상태가 있고, 이 **마지막 상태**가 되면 그 때 반환값을 계산할 수 있다.
- MDP로 정의되는 세계에서 에이전트와 환경의 상호작용은 **불확실성을 내포**하고 있어, 특정 상태의 반환값은 에피소드마다 다를 수 있다. 즉, 이렇게 얻은 반환값은 각 상태의 가치를 대표하기 어렵다. 따라서 에이저트는 **반환값에 대한 기댓값**으로 특정 상태의 가치를 판단한다.
  - 가치함수는 다음 처럼 표시할 수 있다. 각 타임스텝마다 받는 보상이 모두 확률적이고 반환값이 그 보상들의 합이므로 반환값은 확률 변수이다. 하지만 가치함수는 확률변수가 아니라 특정 양을 나타내는 값이므로 소문자로 표현한다.

$v(s) = E[G_t|S_t = s]$

    - 이렇게 가치를 고려하는 이유는 만약 현재 에이전트가 **갈 수 있는 상태들의 가치를 안다면** 그 중에서 가장 **가치가 제일 높은 상태를 선택**할 수 있기 때문이다.
    - 에이전트는 이러한 가치함수를 통해 어느 상태가 좋을지 판단한다.
    - 위에서 정의한 식에 반환값의 수식을 대입하면 다음과 같다.

    $v(s) = E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ...|S_t = s]$

    $v(s) = E[R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + ...)|S_t = s]$

    $v(s) = E[R_{t+1} + \gamma(G_{t+1})|S_t = s]$

    $v(s) = E[R_{t+1} + \gamma v(S_{t+1})|S_t = s]$

- 여기까지는 가치함수를 고려할 때 정책을 고려하지 않았지만, 에이전트가 앞으로 받을 보상에 대해 생각할 때 정책을 고려하지 않으면 안 된다. 에이전트는 무조건 행동을 해야하고 각 상태에서 행동을 하는 것이 에이전트의 정책이기 때문이다.
    - 현재 상태에서 에이전트가 다음에 어떤 상태로 갈지 결정하는 것 : 정책에 따라 선택할 행동, 상태 변환 확률
    - 보상은 어떤 상태에서 어떤 행동을 하는지에 따라 환경에서 에이전트에게 주어지기 때문에 MDP로 정의되는 문제에서 가치함수는 항상 정책에 의존하게 된다.
    - 따라서 가치함수에 아래 첨자로 정책을 쓰면 더 명확한 수식이 된다. 기댓값을 계산할 때도 정책에 따라 계산해야 하기 때문에 기댓값 기호 아래 첨자로 정책을 써준다. 이는 **벨만 기대 방정식(Bellman Expectation Equation)**이라 한다. 벨만 기대 방정식은 **현재 상태의 가치함수 $v_\pi(s)$와 다음 상태의 가치함수 $v_\pi(S_{t+1})$사이의 관계를 말해주는 방정식**이다.

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

- **상태 가치함수(state value-function)** : 상태가 입력으로 들어오면 그 상태에서 앞으로 받을 보상의 합을 출력하는 함수, 가치함수를 통해 어떤 상태에 있는 것이 얼마나 좋은지 알 수 있다.
-

큐함수

- 에이전트는 가치함수를 통해 다음에 어떤 상태로 가야 할지 판단할 수 있다. **어떤 상태로 가면 좋을지 판단한 후**에 **그 상태로 가기 위한 행동을 따져볼 것**이다.

- 하지만, 에이전트가 선택한 행동에 따라 즉각적으로 받는 보상이 달라진다. 또한 에이전트가 왼쪽으로 가는 행동을 했더라도 상태 변환 확률에 따라 왼쪽으로 가지 않을 수 있다.
- 상태 가치함수가 각 상태에 대해 가치를 알려주는 것처럼 각 행동에 대해 가치를 알려주는 함수가 있으면 유용할 듯함.

- **큐함수(Q function)** , 행동 가치함수 : 어떤 상태에서 어떤 행동이 얼마나 좋은지 알려주는 함수
  - 상태, 행동이라는 두 가지 변수를 가진다. $q_\pi(s,a)$로 나타낸다.
- 가치함수과 큐함수 사이의 관계
  1. 각 행동을 했을 때 앞으로 받을 보상인 큐함수 $q_\pi(s,a)$를 $\pi(a|s)$에 곱한다.
  2. 모든 행동에 대해 1의 식을 더하면 가치함수가 된다.

     $v_\pi(s)$ = $\sum_{a\in A}\pi(a|s)q_\pi(s,a)$
- 강화학습에서 에이전트가 행동을 선택하는 기준으로 가치함수보다는 보통 큐함수를 사용한다.
- 큐함수의 벨만 기대 방정식 형태

  $q_\pi(s,a) = E_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1})|S_t = s, A_t = a]$

벨만 방정식

벨만 기대 방정식

- 가치함수 : 에이전트가 그 상태로 갈 경우에 앞으로 받을 보상의 합에 대한 기댓값. 가치함수는 에이전트의 정책에 영향을 받고, 따라서 정책을 반영하여 식으로 나타내면 $v_\pi(s) = E_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s]$ 이다. 이를 **벨만 기대 방정식**이라 한다. 식에 기댓값 개념이 들어가며, 이 벨만 방정식은 **현재 상태의 가치함수와 다음 상태의 가치함수 사이의 관계**를 식으로 나타낸 것이다.
- 벨만 방정식이 왜 중요할까?
  - 반환값으로 나타내는 가치함수

    $v_\pi(s) = E_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3}...|S_t = s]$

- 위의 식으로 부터 기댓값을 알아내려면 앞으로 받을 모든 보상에 대해 고려해야 한다.
    - 정의상 가능하지만 상태가 많아질수록 상당히 비효율적이다.
    - 많은 컴퓨터 계산에서 방정식을 풀 때 식 하나로 풀어내는 방법보다 식 자체로는 풀리지 않지만 **계속 계산을 하면서 푸는 방법**을 사용한다.
- 벨만 방정식을 이용해서 가치함수를 계산하면, 한 번에 모든 것을 계산하는 것이 아니라 값을 변수에 저장하고 루프를 도는 계산을 통해 참 값을 알아나가는 것과 같다.

- 방식
    - 벨만 기대 방정식 : $v_\pi(s) = E_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s]$
    - 만약 가치함수가 참 값이라면 등호가 성립하지만, 그렇지 않다면 수식의 왼쪽과 오른쪽이 다르다.
    - 이 경우 원래 가치함수의 값을 $E_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s]$로 대체한다.
        - 여기서 기댓값은 어떻게 계산?
        - 기댓값에는 어떠한 행동을 할 확률(정책 $\pi(a|s)$)과 그 행동을 했을 때 어떤 상태로 가게 되는 확률(상태 변환 확률 $P^a_{ss'}$)이 포함되어 있다.
        - 따라서 정책과 상태 변환 확률을 포함해서 계산하면 된다. 식은 다음과 같다.

          $v_\pi(s) = \sum_{a \in A} \pi(a|s)(r_{(s,a)} + \gamma \sum_{s' \in S} P^a_{ss'} v_\pi(s'))$
        - 벨만 기대 방정식을 이용해 현재의 가치함수를 계속 업데이트하다보면 참값을 구할 수 있다.
            - 참값: **현재의 정책을 따라갔을 경우** 에이전트가 얻을 실제 보상의 값에 대한 참 기댓값 (최대로 받을 보상 X)

벨만 최적 방정식

- 참 가치함수와 최적 가치함수(optimal value function)
    - 참 가치함수: 어떤 정책을 따라서 움직였을 경우에 받게 되는 보상에 대하 참값
    - 최적의 가치함수: 수많은 정책 중에서 가장 높은 보상을 얻게 되는 정책을 따랐을 때의 가치함수
- $v_{k+1}(s) \leftarrow \sum_{a in A} \pi(a|s)(r_{(s,a)} + \gamma v_k(s'))$

- $v_{k+1}(s)$: 현재 정책에 따라 k+1 번째 계산한 가치함수를 뜻하고, 그 중에서 상태 s의 가치함수를 의미한다.
- k+1 번째의 가치함수는 k번째 가치함수 중에서 주변 상태들 s'을 이용하여 구한다.
- **이 계산은 모든 상태에 대해 동시에 진행한다.**
- 상태집합에 속한 모든 상태에 대해 가능한 행동들을 고려하고, 주변 상태에 저장되어 있는 가치함수를 통해 현재의 가치함수를 업데이트한다.

- 그렇다면 최적 정책은 어떻게 찾는가?
  - 단순히 현 정책에 대한 가치함수를 찾는 것이 아니라, 더 좋은 정책으로 현재의 정책을 업데이트해나가야 한다.
  - 정책을 따라갔을 때 받을 보상들의 합인 가치함수를 통해 판단할 수 있다. 다시말해 가치함수는 정책이 얼마나 좋은지를 알려줄 수 있다.
  - 모든 정책 중 가장 큰 가치함수를 주는 정책이 최적 정책이다.
  - **최적 가치함수:** 최적 정책을 따라갔을 때 받을 보상의 합
  - 최적의 가치함수

    $$v_*(s) = max_\pi[v_\pi(s)]$$

  - 최적의 큐함수

    $$q_*(s,a) = max_\pi[q_\pi(s,a)]$$

- 최적 정책
  - 가장 높은 가치함수/큐함수를 찾았다면, 각 상태 s에서의 최적의 큐함수 중 가장 큰 큐함수를 가진 행동을 하는 것이다. 즉, 선택 상황에서 판단 기준은 큐함수이며, 최적 정책은 언제나 이 큐함수 중에서 가장 높은 행동을 하는 것이다.
  - 최적 정책

    $$\pi_*(s,a) = \begin{cases} 1 & \text{if } \ a = argmax_{a \in A} q_*(s,a) \\ 0 & \qquad\qquad \text{otherwise} \end{cases}$$

  - 어떻게 최적 가치함수와 큐함수를 구하는지는 다음 장에서 학습(순차적 행동 결정 문제)
- 최적 가치 함수끼리의 관계

- 큐함수 중 최대를 선택하는 최적 가치함수, 선택의 기준이 되는 큐함수가 최적의 큐함수가 아니라면 아무리 에이전트가 큐함수 중 최대를 선택하더라도 최적의 가치함수가 되지 않는다.

  $v_*(s) = max_a[q_*(s,a)|S_t = s, A_t = a]$

- 위의 식에서 큐함수를 가치함수로 바꾸면 **벨만 최적 방정식(Bellman Optimality Equation)**이 된다. 이는 **최적의 가치함수에 대한 것**이다.

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

- 큐함수에 대해서의 벨만 최적 방정식을 구할 수 있다.

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

- 다이내믹 프로그래밍(Dynamic programming)
  - 벨만 기대 방정식과 벨만 최적 방정식을 이용하여 MDP로 정의되는 문제를 계산으로 푸는 방법