강화 학습(Reinforcement Learning)

환경(시뮬레이터)에서 **에이전트**가 상태를 관측하고 행동을 하고 행동에 대한 (+/−) 보상을 받는다. 에이전트는 보상의 장기간 기대치를 최대로 하는 행동을 학습한다.

상태: 자신의 상황에 대한 관찰(시간에 따라 달라짐)

$$S_t = s \;\; s \in S$$

행동: 에이전트가 상태 S_t에서 가능한 행동

$$A_t = a \;\; a \in A$$

정책: 에이전트가 행동을 결정하기 위해 사용하는 알고리즘

확률적 정책: 무작위성이 포함된 정책

훈련시키는 것

1. 정책 파라미터(policy parameter)

   정책 공간이 너무 크면 정책 탐색이 무식할 수 있어 조심하자

2. 유전 알고리즘(genetic algorithm)

   1. 1세대 정책 m개를 랜덤하게 생성해서 시도
   2. 성능이 낮은 정책 n개를 버리고(m 〉 n)

      m−n 개를 살려 각각 자식 정책 p개를 생산

      이 자식 정책은 부모를 복사하는 것에 약간의 무작위성을 더한 것
   3. 살아남은 정책(m−n 개)와 그 자식((m−n)xp 개)는

      2세대((m−n)x(p+1) 개)를 구성
   4. 좋은 정책을 찾을 때까지 여러 세대에 걸쳐 반복

3. 정책 그레디언트(policy gradient. PG)

   정책 파라미터에 대한 보상의 그레디언트를 평가해서

   높은 보상의 방향을 따르는 그레디언트로 파라미터를 수정하는 최적화 기법

1. 신경망 정책

: 관측을 입력을 받고 행동에 대한 확률을 추정

가장 높은 점수의 행동을 그냥 선택하지 않고 신경망이 만든 확률을 기반으로 랜덤하게 행동을 선택 → 탐험(exploring)과 활용(exploiting) 사이에 균형

탐험을 위해 과거의 행동과 관측은 무시해도 괜찮다. 각 관측이 환경에 대한 완전한 상태를 담고 있기 때문이다.

BUT 만약 어떤 상태가 숨겨져 있다면 과거의 행동과 관측도 고려해야 한다.

2. 활용 평가: 신용 할당 문제

에이전트가 보상을 받았을 때 어떤 행동 때문인지 알기 어렵다.

보상을 받기 직전의 행동 때문인지 아니면 그 이전의 어떤 행동 때문인지

대가(할인된 보상의 합)로 행동을 평가한다

⇒ 할인된 보상: 단계마다 할인 계수(discount facor) γ를 적용한 보상

⇒ 행동의 대가(return): 할인된 보상의 합

할인 계수(γ)가 0에 가까우면 미래의 보상은 현재의 보상만큼 중요하지 않음

할인 계수(γ)가 1에 가까우면 미래의 보상은 현재의 보상만큼 중요

3. 행동 이익(action advantage)

: 가능한 다른 행동과 비교해서 각 행동이 얼마나 좋은지 혹은 나쁜지 추정

많은 에피소드를 실행하고 모든 행동의 대가를 표준화 $\frac{x-mean}{std}$해야 한다.

결과가 음수인 행동은 나쁘고 양수인 행동은 좋다고 가정할 수 있다.

마르코프 결정 과정

마르코프 프로세스(Markov Process, MP)

↓ + 보상

마르코프 보상 프로세스(Markov Reward Process, MRP)

↓ + 행동

마르코프 결정 과정(Markov Decision Process, MDP)

↓ + 학습 개념

강화학습 문제 표현

마르코프 연쇄(Markov Chain)

: 시간에 따른 상태 변화에 대한 메모리가 없는 확률 과정

정해진 개수의 상태를 가지고 있으며, 각 스텝마다 한 상태에서 다른 상태로 랜덤하게 전이

싱테 s에서 상태 s'로의 전이 확률은 고정되어 있고, (시스템에 메모리가 없으므로) 과거 상태에는 상관없이 (s, s') 쌍에만 의존

종료 상태(terminal state)

1. 마르코프 프로세스(MP)

: 마르코프 특성을 지니는 이산 시간에 대한 확률적 상태 변화 과정

마르코프 특성(Markov Property): 다음 상태는 현재 상태에 의존

⇒ 미래 상태가 과거 상태와는 독립적으로 현재 상태로만 결정

$P(S_{t+1}|S_t) = P(S_{t+1}|S_1, \cdots, S_t)$, P: 확률

전이: 상태 변화(확률로 표현)

상태 전이 확률(state transition probability)

: 어떤 상태 $S_t = s$에 있을 때 그다음 상태 $S_{t+1} = s'$가 될 확률

$P(S_{t+1} = s'|S_t = s) = P_{ss'}$, S: 상태의 집합, $P_{ss'}$: 확률

$P(A|B)$: 조건부 확률(B가 발생했을 때 A가 발생할 확률)

상태 전이 행렬(state transition probability matrix)

: 상태 전이 확률을 행렬로 표현(각 행의 요소를 모두 더하면 1)

2. 마르코프 보상 프로세스(MRP)

: MP에서 각 상태마다 (+/−)보상이 추가된 확률 모델

보상을 받을 시점: 현재 or 미래
'특정 상태에 빨리 도달해서 즉시 받을 것인지' vs '나중에 도달해서 받을 것인지'
미래 가치는 t부터 충분히 시간이 지나고 이자가 붙어야 현재 가치와 같다.
⇒ 현재 가치에 할인율을 적용해야 현재 시점에서 본 미래 가치가 나온다.
할인율 = 0 : 당장의 보상 → 근시안적 행동
할인율 = 1 : 미래 보상에 대한 할인이 적다 → 원시안적 행동

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k+1}^{\infty} \gamma^k r_{t+k+1}$$

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = r_{t+1} + \gamma(r_{t+2} + \gamma r_{t+3} + \cdots) = r_{t+1} + \gamma G_{t+1}$$

가치: 현재 상태 s일 때 앞으로 발생할 것으로 기대되는(E) 모든 보상의 합
가치 함수: 현재 시점에서 미래에 기대되는 모든 보상을 표현하는 미래 가치
목표: 미래 가치의 최대화

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

3. 마르코프 결정 과정(MDP)

: MRP에서 행동이 추가된 확률 모델
마르코프 연쇄와 비슷하지만 각 스텝에서 에이전트는 여러 가능한 행동 중 하나를 선택할 수 있고 전이 확률은 선택된 행동에 따라 달라진다. 어떤 상태 전이는 (+/−) 보상을 반환한다.

목표: 전체적 보상을 최대화하는 정책 또는 각 상태에서의 행동 찾기

정책(policy, π): 각 상태마다 행동 분포(행동이 선택될 확률)를 표현하는 함수

$$\pi(a|s) = P(A_t = a | S_t = s)$$

MDP가 주어진 정책을 따를 때 상태 s에서 상태 s'로 전이할 확률

$$P^{\pi}_{ss'\pi} = \sum_{a \in A} \pi(a|s) P^{a}_{ss'}$$

상태 s에서 얻을 수 있는 보상

$$R^{\pi}_{s} = \sum_{a \in A} \pi(a|s) R^{a}_{s}$$

1. 상태-가치 함수(state-value function)

: 정책에 따라 행동을 결정하고 다음 상태로 이동할 때

상태 s에서 얻을 수 있는 보상의 기댓값

$$\begin{aligned} v_{\pi}(s) &= E_{\pi}[G_t | S_t = s] \\ &= E_{\pi}[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_{t=s}] \\ &= E_{\pi}[R_{t+1} + \gamma G_{t+1} | S_{t=s}] \\ &= E_{\pi}[R_{t+1} + \gamma v_{\pi}(S_{t+1}) | S_t = s] \\ &= E_{\pi}[\text{즉각적 보상} + \text{할인률} \times \text{미래 보상} | S_t = s] \end{aligned}$$

2. 행동-가치 함수(action-value function)

: 상태 s에서 행동 a를 선택했을 때 얻을 수 있는 보상의 기댓값

$$\begin{aligned} q_{\pi}(s,a) &= E_{\pi}[G_t | S_t = s, A_t = a] \\ &= E_{\pi}[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a] \\ &= E_{\pi}[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a] \\ &= E_{\pi}[\text{즉각적 보상} + \text{할인률} \times \text{미래 보상} | S_t = s, A_t = a] \end{aligned}$$

벨만 방정식(Bellman equation)

: 상태-가치 함수와 행동-가치 함수의 관계

1. 벨만 기대 방정식(Bellman expectation equation)

: 정책을 고려한 다음 상태로의 전이를 표현한 가치 함수

MDP의 상태-가치 함수에 대한 도출 과정

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

$$
\begin{aligned}
v_\pi(s) &= E_\pi[G_t|S_t=s] \\
&= E_\pi[R_{t+1}+\gamma R_{t+2}+\gamma^2 R_{t+3}+\cdots|S_t=s] \\
&= E_\pi[R_{t+1}+\gamma(R_{t+2}+\gamma R_{t+3}+\cdots)|S_t=s] \\
&= E_\pi[R_{t+1}+\gamma G_{t+1}|S_t=s] \\
&= E_\pi[R_{t+1}+\gamma v_\pi(S_{t+1})|S_t=s] \quad \cdots (1)
\end{aligned}
$$

현재 시점의 가치를 현재의 보상과 다음 시점의 가치로 표현할 수 있다.(재귀적)

상태-가치 함수

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

기댓값 $E_\pi[\ ]$**: 상태 s에서 정책에 따라 행동했을 때**

얻을 수 있는 각 행동($\pi(a|s)$**)과 그 행동이 발생할 확률(**$p(s',r|s,a)$**)를 곱한 것**

R_{t+1}**: 현재 행동을 선택했을 때 즉각적으로 얻어지는 보상**

γG_{t+1}**: 할인율 x 더 미래에 일어날 보상 = 미래에 일어날 모든 일의 평균**

$$
\begin{aligned}
v_\pi &= E_\pi[R_{t+1}+\gamma G_{t+1}|S_t=s] \\
&= \sum_{a\in A}\pi(a|s)\sum_{s',r}P(s',r|s,a)[r+\gamma E_\pi[G_{t+1}|S_{t+1}=s']]
\end{aligned}
$$

이고 $\gamma E_\pi[G_{t+1}|S_{t+1}=s']=\gamma v_\pi(s')$ **이므로**

$$
\begin{aligned}
v_\pi &= E_\pi[R_{t+1}+\gamma G_{t+1}|S_t=s] \\
&= \sum_{a\in A}\pi(a|s)\sum_{s',r}P(s',r|s,a)[r+\gamma E_\pi[G_{t+1}|S_{t+1}=s']] \\
&= \sum_{a\in A}\pi(a|s)\sum_{s',r}P(s',r|s,a)[r+\gamma v_\pi(s')] \quad \cdots (1) \\
&= \sum_{a\in A}\pi(a|s)q_\pi(s,a) \quad \cdots (2)
\end{aligned}
$$

식 (1)과 식 (2): 상태-가치 함수의 벨만 방정식

행동에 대한 가치

1. 상태 s에서 행동 α를 했을 때의 보상

2. 그다음 상태의 가치 함수

: 할인율과 상태 s'로 전이될 확률을 적용해야 한다

$$
q_\pi(s,a)=R_s^a+\gamma\sum_{s'\in S}P_{ss'}^a v_\pi(s') \quad \cdots (3)
$$

식 (3)을 식 (2)에 대입

$$
v_\pi=\sum_{a\in A}\pi(a|s)q_\pi(s,a)=\sum_{a\in A}\pi(a|s)(R_s^a+\gamma\sum_{s'\in S}P_{ss'}^a v_\pi(s'))
$$

행동-가치 함수

$$\begin{aligned} q_\pi(s,a) &= E_\pi[G_t | S_t = s, A_t = a] \\ &= E_\pi[R_{t+1} + \gamma E_\pi[G_{t+1} | S_{t+1} = s'] | S_t = s, A_t = a] \\ &= E_\pi[R_{t+1} + \gamma E[G_{t+1} | S_{t+1} = s', A_{t+1} = a'] | S_t = s, A_t = a] \end{aligned}$$

에 $R_s^a = E_\pi[R_{t+1} | S_t = s, A_t = a]$와

$P_{ss'}^a = P(s' | s, a) = P[S_{t+1} = s' | S_t = s, A_t = a]$를 대입하면

$$\begin{aligned} q_\pi(s,a) &= E_\pi[R_{t+1} + \gamma E[G_{t+1} | S_{t+1} = s', A_{t+1} = a'] | S_t = s, A_t = a] \\ &= R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_\pi(s') \end{aligned}$$

이고, 앞서 구한 상태-가치 함수 $v_\pi = \sum_{a \in A} \pi(a|s) q_\pi(s,a)$를 대입하면

$$q_\pi(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_\pi(s') = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \sum_{a' \in A} \pi(a' | s') \, q_\pi(s', a')$$

이고, 정리하면

$$\begin{aligned} q_\pi(s,a) &= E_\pi[G_t | S_t = s, A_t = a] \\ &= E_\pi[R_{t+1} + \gamma E_\pi[G_{t+1} | S_{t+1} = s'] | S_t = s, A_t = a] \\ &= E_\pi[R_{t+1} + \gamma E[G_{t+1} | S_{t+1} = s', A_{t+1} = a'] | S_t = s, A_t = a] \\ &= E_\pi[R_{t+1} + \gamma v_\pi(s') | S_t = s, A_t = a] \\ &= R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \sum_{a' \in A} \pi(a' | s') \, q_\pi(s', a') \end{aligned}$$

현재와 바로 다음 상태, 그리고 행동 간 관계가 드러나도록 정리한

상태-가치 함수(state-value function)

: 뒤따를 행동-가치 함수의 정책 기반 가중 평균

$$v_\pi = \sum_{a \in A} \pi(a|s) (R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_\pi(s'))$$

행동-가치 함수(action-value function) = Q-가치(Q-Value)

: 다음 상태-가치 함수에 대한 보상과 상태 전이확률에 대한 결합확률의 가중 평균

최적의 상태-행동 가치(sate-action value)를 추정할 수 있는 알고리즘

$$q_\pi(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \sum_{a' \in A} \pi(a' | s') \, q_\pi(s', a')$$

2. 벨만 최적 방정식(Bellman optimality equation)

: 최적화된 정책(최대 보상을 얻는 정책)을 따르는 벨만 방정식

최적의 가치 함수(optimal value function): 최대의 보상을 갖는 가치 함수
상태-가치 함수: 어떤 상태가 더 많은 보상을 받을 수 있는지
행동-가치 함수: 어떤 상태에서 어떤 행동을 해야 더 많은 보상을 받을 수 있는지

∴ 모든 상태에 대해 상태-가치 함수를 계산할 수 있다면
모든 상태에 대한 최적의 행동을 선택할 수 있다

최적의 상태-가치 함수

: 주어진 모든 정책에 대한 상태-가치 함수의 최대값 $V_*(s)$

에이전트가 상태 s에 도달한 후 최적으로 행동한다고 가정하고 평균적으로 기대할 수 있는 할인된 미래 보상의 합

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

또는

재귀적인 식으로 에이전트가 최적으로 행동할 때 현재 상태의 최적 가치
= 한 최적 행동으로 평균적으로 받게 될 보상
+ 이 행동으로 인해 가능한 모든 상태의 최적 가치의 기대치

$$V_*(s) = \max_{a} \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_*(s')] \quad \text{모든} s\text{에 대해}$$

$T(s,a,s')$: a를 선택했을 때 s에서 s'로 전이될 확률
$R(s,a,s')$: a를 선택해서 s에서 s'롤 전이했을 때 받을 수 있는 보상

방법
1. 모든 상태 가치를 0으로 초기화
2. **가치 반복(value iteration) 알고리즘**으로 반복 업데이트

$$V_{k+1}(s) \leftarrow \max_{a} \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_k(s')] \quad \text{모든} s\text{에 대해}$$

최적의 행동-가치 함수 = 최적의 Q-가치

: 주어진 모든 정책에 대한 행동-가치 함수의 최대값
상태-행동 (s, a)쌍에 대한 최적의 Q-가치 $Q_*(s,a)$

에이전트가 상태 s에 도달해서 행동 a를 선택하고 행동의 결과를 얻기 전
평균적으로 기대할 수 있는 할인된 미래 보상의 합

가정: 에이전트가 이 행동 이후 최적의 행동을 할 것

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

방법

1. Q-가치를 추정을 모두 0으로 초기화
2. **Q-가치 반복(Q-value iteration) 알고리즘**으로 업데이트

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')] \quad \text{모든 } (s,a) \text{에 대해}$$

최적화된 정책

: 행동-가치 함수를 최대로 하는 행동만 선택하는 정책 $\pi_*(s)$

상태 s에 도달했을 때 최적의(가장 높은) Q-가치를 가진 행동

행동-가치 함수(Q-함수)에 대한 최적의 가치 함수를 구할 수 있다면
주어진 상태에서 Q 가치가 가장 큰 행동을 선택할 수 있다
⇒ 최적화된 정책 찾을 수 있다

$$\pi_*(a|s) = \begin{cases} 1 & a = \underset{a}{argmax}\, q_*(s,a) \text{일 때} \\ 0 & \text{그 외} \end{cases} \quad \text{또는} \quad \pi_*(s) = \underset{a}{argmax}\, Q_*(s,a)$$

강화학습 과정

1. 처음 에이전트가 접하는 상태 s나 행동 a는 임의의 값으로 설정
2. 환경과 상호 작용하면서 얻은 보상과 상태에 대한 정보를 이용해서
   어떤 상태에서 어떤 행동을 선택하는 것이 최대 보상을 얻을 수 있는지 판단
3. 상태-가치 함수와 행동-가치 함수를 최적의 행동을 판단하는 수단으로 사용
   벨만 기대 방정식을 이용하여 업데이트하면서
   높은 보상을 얻을 수 있는 상태와 행동을 학습
4. 2~3의 과정에서 최대 보상을 갖는 행동을 선택하도록 최적화된 정책을 찾는다

∴ 벨만 기대 방정식의 의미: 미래의 가치 함수 값들을 이용하여 초기화된 임의의 값들을 업데이트하면서 최적의 가치 함수로 다가가는 것

가치함수를 계산하는 방법

1. 다이나믹 프로그래밍(dynamic programming)

다이나믹: 연속적으로 발생하는 문제 풀기 + 프로그래밍: 수학적 문제
다이나믹 프로그래밍: 연속적으로 발생되는 문제를 수학적으로 최적화하여 풀기

가정: MDP의 모든 상황에 대한 것을 이미 알고 있다
→ 계획(planning)이 가능해진다
어떤 행동을 선택했을 때 어떤 상태가 되는지 이미 아는 것과 같다

예측 과정(prediction): MDP와 정책을 입력하여 가치 함수를 찾아내기
컨트롤 과정(control): MDP를 입력으로 하여 기존 가치 함수를 더욱 최적화하기

단점: 대부분 RL 문제는 상태와 전이가 많다

① 정책 이터레이션(policy iteration)

평가(evaluate): 현재 정책을 이용하여 가치 함수 찾기
발전(improve): 평가에서 구한 가치 값과 행동에 대한 가치 값을 비교하여
더 좋은 정책을 찾아가는 과정
이 두 가지 과정을 반복 수행하면 정책과 가치는 특정 값으로 수렴(최적화)

– 정책 평가(policy evaluation)
: 모든 상태에 대해 그 다음 상태가 될 수 있는 행동에 대한 보상의 합을 저장
주변 상태의 가치 함수와 바로 다음 상태에서 얻어지는 보상만 고려해서
현재 상태의 다음 가치 함수 계산
실제 가치 함수값은 아니지만 무한히 반복하면 실제 가치 함수값으로 수렴

– 정책 발전(policy improvement)
ex) 욕심쟁이 정책 발전(greedy policy improvement)
: 에이전트가 할 수 있는 행동들의 행동–가치 함수를 비교하고
가장 큰 함수 값을 가진 행동 선택
이전 가치 함수보다 업데이트된 정책으로 받을 가치 함수는
무조건 크거나 같고 장기적으로 최적화된 정책에 수렴

$$\pi'(s) = \underset{a \in A}{argmax}\, q_{\pi}(s, a)$$

② 가치 이터레이션(value iteration)

: 최적의 정책을 가정하고 벨만 최적 방정식을 이용하여 순차적으로 행동 결정

정책 발전이 필요하지 않다. 벨만 최적 방정식으로 한 번의 정책 평가 과정을 거치면 최적의 가치 함수와 최적의 정책이 구해지면서 MDP 문제를 풀 수 있기 때문.

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

2. 몬테카를로(Monte Carlo method)

: 전체 상태 중 일부 구간만 방문하여 근사적으로 가치 추정

초기 상태에서 시작하여 중간 상태들을 경유해서 최종 상태까지 간 후 최종 보상을 측정하고 방문했던 상태들의 가치를 업데이트

단점: 최종 상태까지 도달해야만 가치를 업데이트할 수 있다

몬테카를로 트리 탐색

모든 노드를 대상으로 탐색하는 대신 게임 시뮬레이션을 이용하여
가장 가능성이 높아 보이는 방향으로 행동을 결정하는 탐색 방법
무작위 방법 중 가장 승률이 높은 값을 기반으로 시도

1. 선택

: 루트 R에서 시작해서 현재 펼쳐진 트리 중 가장 승산 있는 자식 노드 L을 선택

$$\frac{w_i}{n_i} + C\sqrt{\frac{\log t}{n_t}}$$

$$\begin{pmatrix} w_i : i\text{번 움직인 후의 승리 횟수} \\ n_i : i\text{번 움직인 후의 시뮬레이션 횟수} \\ C\text{: 탐험 파라미터로 } \sqrt{2}\text{를 처음 초기값으로 많이 사용} \\ \text{하지만 } C\text{값은 실험으로 조정해야 한다} \\ t : \text{시뮬레이션의 전체 횟수} \\ t\text{는 모든 } n_i\text{의 합이므로 } t\text{는 부모 노드의 } n_i\text{값} \end{pmatrix}$$

2. 확장

: 노드 L에서 종료되지 않으면 하나 또는 그 이상의 자식 노드를 생성하고
그 중 하나의 노드 C를 선택

3. 시뮬레이션
: 노드 C에서 랜덤으로 자식 노드를 선택하여 게임을 반복 진행

4. 역전파
: 시뮬레이션의 결과로 노드 C, L, R까지 경로에 있는 노드들의 정보 갱신

5. 1~4번 과정(선택→확장→시뮬레이션→역전파) 반복

최선의 선택을 위해 트리에서 랜덤 시뮬레이션으로 최적의 선택 결정
이때 여러 랜덤 시뮬레이션에 대해 각 움직임을 측정하여 효율적인 경우의 수 예측

3. 시간 차 학습(tempoeral difference learning, TD)

: 최종 상태 도달 전 방문한 상태의 가치를 즉시 업데이트

가치 반복 알고리즘과 비슷하지만 에이전트가 MDP에 대해 일부 정보만 알고 있을 때 다룰 수 있도록 변형한 것
도입 배경
: 보통 마르코프 결정 과정은
① 초기 에이전트는 전이 확률 $T(s,a,s')$에 대해 알지 못하고
보상 $R(s,a,s')$이 얼마나 되지는 알지 못한다.
② 보상에 대해 알기 위해서는 적어도 한번은 각 상태와 전이를 경험해야 한다.
③ 전이 확률에 대해 신뢰할 만한 추정을 얻으려면 여러 번 경험해야 한다.

가정: 에이전트가 초기에 가능한 상태와 행동만 알고 다른 것은 모른다

탐험 정책을 사용해 MDP를 탐험한다.
탐험을 진행하면서 TD 학습 알고리즘이 실제로 관측된 전이와 보상에 근거하여
상태 가치의 추정값을 업데이트 한다.

$$V_{k+1}(s) \leftarrow (1-\alpha)\, V_k(s) + \alpha(r + \gamma\, V_k(s'))$$

또는 다음과 같이 쓸 수 있다

$$V_{k+1}(s) \leftarrow V_k(s) + \alpha\, \delta_k(s,r,s') \quad \delta_k(s,r,s') = r + \gamma\, V_k(s') - V_k(s)$$

α: 학습률

$r+\gamma V_k(s')$: TD 타겟

$\delta_k(s,r,s')=r+\gamma V_k(s')-V_k(s)$: TD 오차

첫 번째 형태를 더 간단히 쓰는 방법: $a \underset{\alpha}{\leftarrow} b$ 표기법($a_{k+1} \leftarrow (1-\alpha)a_k+\alpha b$)

$$V(s) \underset{\alpha}{\leftarrow} r+\gamma V(s')$$

각 상태 s에서 이 알고리즘은 에이전트가 이 상태를 떠났을 때
얻을 수 있는 당장의 보상과
(최적으로 행동한다고 가정하여) 나중에 기대할 수 있는 보상을
더한 이동 평균을 저장한다.

4. 함수적 접근 학습(functinal approximation learning)

: 연속적 상태를 학습하고자 상태와 관련된 특성 벡터 도입
특성의 가중치를 업데이트하여 가치의 근사치 찾을 수 있다

도입 배경
: 보통 마르코프 결정 과정은 상태가 아주 많거나 상태가 연속적인 값일 때 가치 함수를 테이블 형태로 학습하기 어렵다

심층 강화 학습(Deep Reinforcement Learning)

마르코프 결정 과정(MDP)와 함께
1. 정책 그레디언트(policy gradient. PG)
2. Q-러닝(Q-Learning) & 탐험 정책(exploration policy)
3. 심층 Q-네트워크(deep Q-network. DQN) a.k.a 딥 Q-러닝
4. 더블 DQN(double DQN)
5. 듀얼링 DQN(dueling DQN)

1. 정책 그래디언트(PG 알고리즘)

① Reinforce 알고리즘

1. 신경망 정책이 여러 번에 걸쳐 게임을 플레이하고 매 스탭마다 선택된 행동이 더 높은 가능성을 가지도록 만드는 그레디언트를 계산한다. 하지만 아직 이 그레디

언트는 적용하지 않는다
2. 에피소드를 몇 번 실행한 다음, 각 행동의 이익을 계산
3. 한 행동이 양수면 좋은 것이므로 미래에 선택될 가능성이 높도록 앞서 계산한 그레디언트를 적용. 반대로 한 행동이 음수면 나쁜 것이므로 미래에 이 행동이 덜 선택되도록 반대의 그레디언트를 적용. 이는 각 그레디언트 벡터와 그에 상응하는 행동 이익을 곱하면 된다.
4. 모든 결과 그레디언트 벡터를 평균 내어 경사 하강법 스텝을 수행

정책 그레디언트 알고리즘은 CartPole 보다 더 크고 복잡한 문제에는 잘 적용되지 못한다. 샘플 효율성이 매우 좋지 못해 아주 긴 시간 동안 게임을 플레이해야 정책을 많이 개선할 수 있기 때문이다.

② 엑터-크리틱 알고리즘(actor-critic)

2. Q-러닝

Q-함수: 에이전트가 상태 s에서 행동 a을 했을때 보상의 기댓값을 예측하는 함수
Q-러닝: 모델 없이 학습하는 강화 학습 기법 중 하나로 Q-함수를 사용하여
마르코프 결정 과정에서 최적화된 정책을 학습하는 강화 학습 기법
여러 실험(episode)을 반복하여 최적의 정책을 학습

전이 확률과 보상을 초기에 알지 못한 상황에서 Q-가치 반복 알고리즘을 적용

$$Q(s,a) \underset{\alpha}{\leftarrow} r + \gamma \max_{a'} Q(s',a')$$

에이전트가 플레이하는 것을 보고 점진적으로 Q-가치 추정을 업데이트
정확한(또는 충분히 근접한) Q-가치 추정을 얻게 되면
최적의 정책은 가장 높은 Q-가치를 가지는 행동을 선택(탐욕적 정책)
각 (s, a)쌍마다 a를 선택해 s를 떠났을 때 에이전트가 받을 수 있는 보상 r과 기대할 수 있는 할인된 미래 보상의 합을 더한 이동 평균을 저장

미래 보상의 합 추정: 타깃 정책이 이후로 최적으로 행동한다고 가정하고
다음 상태 s'에 대한 Q-가치의 최대값을 선택

학습 방법

1. 초기화: Q-테이블(Q-table)에 있는 모든 Q-가치를 '0'으로 초기화

   Q-table: 모든 상태와 행동에 대한 기록을 담고 있어,

   각 상태마다 최적의 행동을 취할 수 있는 가이드 제공

2. 행동 a를 선택하고 실행
3. 보상 r과 다음 상태 s'를 관찰
4. 상태 s'에서 가장 높은 Q 가치를 갖는 행동인 a'를 선택
5. 다음 공식을 이용하여 상태에 대한 Q-가치를 업데이트

$$Q_t(s_t, a_t) \leftarrow Q_{t-1}(s_t, a_t) + \alpha[R_{t+1} + \gamma \max_{a'} Q_t(s', a') - Q_{t-1}(s_t, a_t)]$$

R_{t+1}: 현재 상태 s에서 어떤 행동 a를 취했을 때 얻는 즉각적 보상

$\max_{a'} Q_t(s', a')$: (미래에 보상이 가장 클 행동을 했다고 가정한) 다음 단계의 가치

$R_{t+1} + \gamma \max_{a'} Q_t(s', a')$: 목표값(target value)

$Q_{t-1}(s_t, a_t)$: 실제 관측값

$(R_{t+1} + \gamma \max_{a'} Q_t(s', a')) - Q_{t-1}(s_t, a_t) = \text{목표값} - \text{실제관측값}$

⇒ 목표값과 실제 관측값의 차이만큼 업데이트 진행

6. 종료 상태에 도달할 때까지 2~5 반복

매 실험(에피소드)에서 각 상태마다 행동을 취하는데, 이때 행동은 랜덤한 선택을 한다. 이유는 가보지 않은 곳을 탐험하면서 새로운 좋은 경로를 찾기 위해.

단점

1. 에이전트가 선택할 수 있는 상태와 행동이 많으면 적용하기 어렵다. 많은 상태와 행동을 가진 대규모 (또는 중간 규모)의 MDP에 대한 Q-table 구축에 한계 있기 때문이다.
2. 데이터 간 상관관계로 학습이 어렵다.

해결 방법

1. 근사 Q-러닝: 어떤 (s, a) 쌍의 Q-가치를 근사하는 함수 $Q_\theta(s, a)$를 (파라미터 벡터 θ로 주어진) 적절한 개수의 파라미터를 사용하여 찾는 것

2. 심층 Q-네트워크(deep Q-network): Q-가치를 추정을 위해 사용한 DNN
3. 심층 Q-러닝(deep Q-learning): 근사 Q-러닝을 위해 DQN 사용

탐험 정책

활용(exploitation): 현재까지 경험 중 현 상태에서
가장 최대의 보상을 받을 수 있는 행동을 하는 것
탐험(exploration): 다양한 경험을 쌓기 위한 새로운 시도

활용과 탐험 사이의 균형이 필요한 이유
: 풍부한 경험이 있어야 더 좋은 선택을 할 수 있지만, 경험을 쌓기 위한 새로운 시도들은 시간과 자원이 낭비되기 때문이다. 균형을 찾아 최단 시간 주어진 환경의 모든 상태 관찰하고, 이를 기반으로 보상을 최대화할 수 있는 행동을 수행한다.

완전한 랜덤 정책은 결국 모든 상태와 전이를 여러 번 경험한다고 보장하지만, 이렇게 하면 극단적으로 오래 걸릴 수 있으므로

① ε-그리디 정책(ε-greedy policy. 욕심쟁이greedy 방법)

: 각 스텝에서 ε 확률로 랜덤하게 탐험하거나 1-ε 확률로 그 순간 가장 최선인 것으로(가장 높은 Q-가치를 선택하여) 활용한다
0~1 사이 난수를 추출해서 특정 임계치(threshold) 보다 낮으면 랜덤하게 행동한다. 임계치는 실험을 반복하면서(학습을 진행하면서) 점점 낮은 값을 갖는다.

장점: (완전한 랜덤 정책에 비해) Q-가치 추정이 점점 향상되기 때문에 환경에 관심 있는 부분을 살피는 데 점점 더 많은 시간을 사용한다. 그럼에도 MDP에서 아려지지 않은 지역을 방문하는데 일정 시간을 사용한다.
ε의 값은 높게 시작해서 점점 감소하는 것이 일반적

② 탐험 함수(exploration function)

: 팀험의 가능성에 의존하는 대신 이전에 많이 시도하지 않았던 행동을 시도하도록 탐험 정책을 강조하는 방법
Q-가치 추정에 보너스 추가

$$Q(s,a) \underset{\alpha}{\leftarrow} r + \gamma \max_{a'} f(Q(s',a'), N(s',a'))$$

$N(s', a')$: s'에서 a'를 선택한 횟수를 카운트
$f(Q, N) = Q + \kappa/(1+N)$: 탐험 함수(κ는 에이전트가 알려지지 않은 곳에 얼마나 흥미를 느끼는지 나타내는 하이퍼 파라미터)

3. 심층 Q-네트워크(deep Q-network. DQN) a.k.a 딥 Q-러닝

: 신경망을 이용하여 Q-함수를 학습하는 강화 학습 기법

목표: 신경망층을 깊게 하여 훈련하여 Q 가치의 정확도를 높이자

DQN은 덜 직접적으로 정책을 최적화한다
에이전트가 각 상태에 대한 할인된 미래의 대가를 추정하도록 학습
각 상태에 있는 각 행동에 대한 할인된 미래의 대가를 추정하도록 학습
이 지식을 사용하여 어떻게 행동할지 결정
이를 위해 마르코프 결정 과정(Markov Decision Process) 사용

1. 근사 Q-러닝
   : 어떤 (s, a) 쌍의 Q-가치를 근사하는 함수 $Q_\theta(s, a)$를 (파라미터 벡터 θ로 주어진) 적절한 개수의 파라미터를 사용하여 찾는 것
2. 심층 Q-네트워크(deep Q-network)
   : Q-가치를 추정하기 위해 사용한 DNN
3. 심층 Q-러닝(deep Q-learning)
   : 근사 Q-러닝을 위해 DQN을 사용하는 것

학습 방법
: 주어진 (s, a)에 대해 DQN이 계산한 근사 Q-가치
1. 벨만 방정식으로 구한 근사 Q-가치는 상태 s에서 행동 a를 했을 때 관측된 보상 r과 그 이후에 최적으로 행동해서 얻은 할인된 가치를 더한 값에 가능한 가깝게 되어야 한다.
2. 다음 상태 s'와 가능한 모든 행동 a'에 대해 DQN을 실행하여 미래의 할인된 가치를 계산하여 가능한 모든 행동에 대한 미래의 근사 Q-가치를 추정한다.
3. (최적으로 행동할 것으로 가정하기 때문에) 가장 높은 Q-가치를 고르고 할인을 적용하여 할인된 미래 보상을 추정한다.

상태–행동 쌍 (s, a)에 대한 타깃 Q–가치
: 보상 r과 미래의 할인된 가치 추정의 합

$$Q_{target}(s,a) = r + \gamma \max_{a'} Q_{\theta}(s',a')$$

타깃 Q–가치로 경사 하강법을 사용해 훈련 단계 수행

추정된 Q–가치 $Q(s,a)$와 타깃 Q–가치 $Q_{target}(s,a)$의 제곱 오차를 최소화

$(Q(s,a) - Q_{target}(s,a))^2$ 또는 $\frac{(Q(s,a) - Q_{target}(s,a))^2}{2}$

또는 알고리즘이 큰 오차에 민감하지 않도록 후버 손실 사용

강화 학습 주의사항
1. 최악의 망각(catastrophic forgetting)
: 환경의 한 부분에서 학습한 것이 앞서 학습한 것을 망가뜨릴 수 있다.

2. 훈련이 불안정하고 파라미터 값과 랜덤 시드의 선택에 매우 민감하다.

3. 손실 그래프를 그리지 않는 이유
: 손실은 강화 학습 모델 성능을 재는 데 좋지 않다. 손실이 내려가더라도 에이전트가 엉망으로 행동할 수 있기 때문이다.(ex. 에이전트가 환경의 작은 지역에 갇힐 때 DQN이 이 영역에서 과대적합) 반대로 손실이 올라가더라도 에이전트가 더 잘 수행할 수 있다.(ex. DQN이 Q–가치를 과소평가하여 예측이 올바르게 증가하기 시작하면 에이전트가 더 많은 보상을 받을 것이다. 하지만 DQN이 타겟을 더 크게 설정하기 때문에 손실을 증가)

4. 더블 DQN(double DQN)

: 타깃 네트워크가 Q–가치를 과대평가하기 쉽다는 관측을 기반하여 이를 개선하기 위해 다음 상태에서 최선의 행동을 선택할 때 타깃 모델 대신 온라인 모델을 사용하도록 제안. 타깃 모델은 최선의 행동에 대한 Q–가치를 추정할 때 사용.

고정 Q–가치 타깃: 두 개의 DQN을 사용하기

도입 배경

: 기본 심층 Q-러닝 알고리즘에서 모델은 예측을 만들고 타깃을 설정하는데 모두 사용한다. 그런데 이런 피드백 순환 과정은 네트워크를 불안정하게 만들 수 있다. (발산, 진동, 동결 등의 문제)

1. 온라인 모델(online model) = Q-네트워크, Q-함수
   : 각 스텝에서 학습하고 에이전트를 움직이는 데 사용

2. 타깃 모델(target model) = 타깃 Q-네트워크, 타깃 Q-함수
   : 타깃을 정의하기 위해서만 사용하는 온라인 모델의 단순한 복사본(온라인 모델 자체와 가중치를 타깃 모델로 복사 but 학습하면서 파라미터는 달라진다)

타깃 모델(타깃 Q-네트워크, 타깃 Q-함수)

: 다음 상태의 Q-가치를 계산할 때 온라인 모델 대신 타깃 모델을 사용

Q-러닝에서 Q-함수가 학습하면서 Q-가치가 계속 바뀌는 문제가 있는데, 이를 해결하기 위해 DQN에서 타깃 Q-네트워크를 사용한다.

DQN에서는 수렴을 원활하게 시키기 위해 타깃 Q-네트워크를 게속 업데이트하는 것이 아니라 주기적으로 한 번씩 업데이트한다.

훈련을 수행할 때의 손실 함수로는 MSE를 사용한다. 네트워크 2개가 분리되어 있으므로 각 네트워크에서 사용되는 파라미터 θ의 표기가 다른 것을 확인할 수 있다

$$L_i(\theta_i) = E_{(s,a,r,s') \sim U(D)}[(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i))^2]$$

리플레이 메모리(replay memory)

= 리플레이 버퍼(replay memor), 재생 메모리, 재생 버퍼

: 에이전트가 수집한 데이터를 저장해 두는 저장소(deque로 구현)

Q-러닝에서는 데이터 간 상관관계로 학습 속도가 느려지는 문제가 있었다.
딥 Q-러닝에서는 리플레이 메모리를 사용하여
에이전트 상태가 변경되어도 즉시 훈련시키지 않고 일정 수의 데이터가 수

집되는 동안 기다린다. 나중에 일정 수의 데이터가 리플레이 메모리(버퍼)에 쌓이면 랜덤하게 데이터를 추출하여 미니 배치를 활용하여 학습한다. 이때 하나의 데이터에는 상태, 행동, 보상, 다음 상태가 저장된다.

$$e_t = (s_t, a_t, r_t, s_{t+1})$$

중요도 샘플링(importance sampling. IS)

= 우선 순위 기반 경험 재생(prioritized experience relay. PER)

: 중요한 경험을 더 자주 샘플링

어떤 경험이 학습 진행을 더 빠르게 만들면 '중요한' 것으로 간주

방법

: TD 오차 $\delta = r + \gamma V(s') - V(s)$의 크기 계산

큰 TD 오차는 전이 (s, a, s')가 매우 놀랍다를 의미(배울 가치 있다)

1. 경험이 재생 버퍼에 기록될 때 적어도 한 번은 샘플링되기 위해 매우 높은 우선 순위 값으로 설정된다.
2. 샘플링된 후 (그리고 샘플링될 때마다) TD 오차 δ를 계산하여 경험 우선 순위를 $p = |\delta|$로 설정한다.
   ※ 경험의 샘플링 확률이 0이 되지 않도록 작은 상수를 더한다
3. 우선 순위 p의 경험을 샘플링할 확률 P는 p^{ζ}에 비례한다.
   ζ: 우선 순위 샘플링을 얼마나 탐욕적으로 할 것인지 제어하는 하이퍼 파라미터
   ζ=0: 균등하게 샘플링
   ζ=1: 완전 중요한 샘플링

※ 샘플이 중요한 경험에 평향되어 있으므로 훈련하는 동안 중요도에 따라 경험의 가중치를 낮춰서 이 편향을 보상해주어야 한다.

중요한 경험이 더 자주 샘플링되기를 원하지만 이는 훈련 과정에서 이 샘플에 낮은 가중치를 주어야 한다.

각 경험의 훈련 가중치 $w = (n P)^{-\beta}$

n: 재생 버퍼에 있는 경험의 개수

β: 중요도 샘플링 편향을 얼마나 보상할지 조정하는 하이퍼파라미터

(0은 보상이 전혀 없고 1은 그 반대)

ex. 학습 초기에는 β=0.4를 사용하고 학습 마지막에는 β=1까지 선형적 증가

5. 듀얼링 DQN(dueling DQN)

(s, a)의 Q-가치가 $Q(s,a) = V(s) + A(s,a)$처럼 표현될 수 있으므로

$V(s)$: 상태 s의 가치

$A(s,a)$: 상태 s에서 가능한 모든 행동과 비교하여 행동 a를 선택했을 때 **이득**

상태의 가치 = 이 상태에서 최선의 행동 a^*의 Q-가치

최적의 정책은 최선의 행동을 선택한다고 가정하기 때문

$$V(s) = Q(s,a^*)\text{이므로 } A(s,a^*) = 0$$

듀얼링 DQN 모델은 상태의 가치와 가능한 각 행동의 이익을 모두 추정하고

최선의 행동은 이익이 0이기 때문에($A(s,a^*) = 0$)

모델이 예측한 모든 이익에서 모든 최대 이익을 뺀다.

참고

1. 모델-기반 vs. 모델-프리

   모델-기반 강화학습(model-based RL)

   : MDP의 전이 확률과 보상에 대한 모델을 알고 있는 경우(가치 반복 알고리즘)

   모델-프리 강화학습(model-free RL)

   : 반대의 경우 (시간차 학습, Q-러닝)

2. 온 폴리시 vs. 오프 폴리시

   온 폴리시(on-policy)

   : 훈련된 정책을 반드시 실행에 사용(정책 그레디언트 알고리즘)

   오프 폴리시(off-policy)

   : 훈련된 정책을 반드시 실행에 사용하지는 않는다 (Q-러닝 알고리즘)

OpenAI GYM

: 다양한 종류의 시뮬레이션 환경 제공 툴킷

gym.envs.registry.all() 함수: 환경의 전체 목록

gym.make() 함수: 환경 생성

.reset() 메서드: 환경 초기화(환경을 만들고 초기화 필수)하고
첫 번째 관찰 변수(상태를 관찰하고 그 정보를 저장)를 반환
관측은 환경에 따라 다르다

.render() 메서드: 환경 출력(반환된 렌더링된 이미지를 mode에 따라 출력)
렌더링된 이미지를 넘파이 배열로 받으려면 mode="rgb_array"로 지정

.step() 메서드: 주어진 행동을 실행하고 4가지 값을 반환
obs: 새로운 관측값
reward: 보상
done: 에피소드의 종료 여부(True면 끝난 것)
에피소드가 끝나면 환경을 다시 사용하기 전에 꼭 초기화해야 한다
info: 디른 환경에서 이 딕셔너리에 디버깅이나 훈련에 유용한 추가적 정보

.close() 메서드: 한 환경을 다 사용했을 때 이 메서드를 호출하여 자원을 반납