Chapter 6 TD법

몬테카를로법을 이용하면 환경 모델 없이도 정책을 평가할 수 있다. 그리고 평가와 개선을 번갈아 반복하면서 최적 정책을 얻을 수 있다. 하지만, 몬테카를로법은 에피소드의 끝에 도달한 후에야 가치 함수를 갱신할 수 있다. 에피소드가 끝나야만 수익이 확정되기 때문이다.

이번 장에서는 환경 모델을 사용하지 않을 뿐 아니라 행동을 한 번 수행할 때마다 가치 함수를 갱신하는 TD법을 설명한다. TD는 'Temporal difference'의 약자로, 시간차라는 뜻이다. 에피소드가 끝날 때까지 기다리지 않고 일정 시간마다 정책을 평가하고 개선한다.

6.1. TD법으로 정책 평가하기

TD법은 몬테카를로법과 동적 프로그래밍을 합친 기법이다.

6.1.1. TD법 도출

먼저 수익은 다음과 같이 수식으로 정리할 수 있다.

$$\begin{aligned} G_t &= R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \cdots \\ &= R_t + \gamma G_{t+1} \end{aligned}$$

시간 t부터 시작하며 보상이 R…식으로 주어진다면, 수익은 할인율을 적용한 보상들의 총합으로 표현된다. 이 수익을 적용하면 가치 함수는 다음과 같이 정의된다.

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t \mid S_t = s] \\ &= \mathbb{E}_\pi[R_t + \gamma G_{t+1} \mid S_t = s] \end{aligned}$$

이 식으로부터 몬테카를로법(MC법)과 DP법을 이용하여 기법을 도출할 수 있다.

MC법에서는 기댓값을 계산하는 대신 실제 수익의 샘플 데이터를 평균하여 기댓값을 근사한다. 지수 이동 평균을 이용하여 새로운 수익이 발생할 때마다 고정된 값 a로 갱신한다. 수식은 다음과 같다.

$$V'_\pi(S_t) = V_\pi(S_t) + \alpha\{G_t - V_\pi(S_t)\}$$

여기서 V_π는 현재의 가치 함수이고 V'_π는 갱신 후의 가치 함수이다. 그래서 식은 가치 함수 V_π를 G_t쪽으로 갱신하고 있다. 얼마나 갱신할지는 a로 조정한다.

다음은 DP법이다. DP법은 식의 계산으로 기댓값을 구한다.

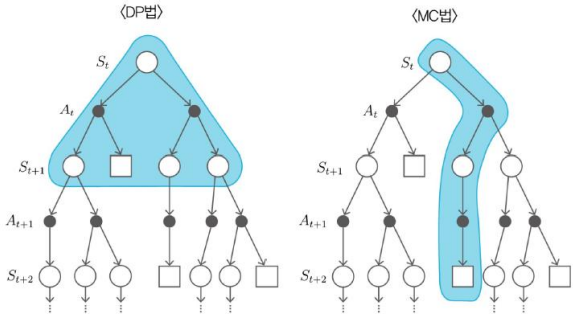
$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[R_t + \gamma G_{t+1} \mid S_t = s] \\ &= \sum_{a,s'} \pi(a \mid s)\, p(s' \mid s,a)\{r(s,a,s') + \gamma v_\pi(s')\} \end{aligned}$$

위 식은 벨만 방정식이다. DP법은 벨만 방정식을 기반으로 가치 함수를 순차적으로 갱신한다. 갱신식은 다음과 같다.

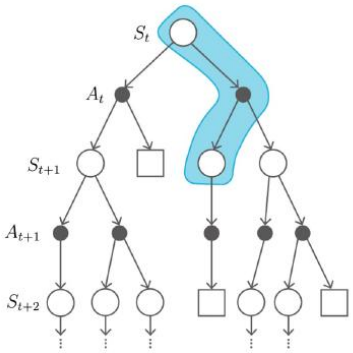
$$V'_\pi(s) = \sum_{a,s'} \pi(a \mid s)\, p(s' \mid s,a)\{r(s,a,s') + \gamma V_\pi(s')\}$$

위 식에서 중요한 점은 '현재 상태에서의 가치 함수'를 '다음 상태에서의 가치 함수'로 갱신한다는 것이다.

이때 모든 전이를 고려하고 있다는게 특징이다.



그림은 상태 S_t에서 시작하여 이후의 모든 전이를 표현한 모습이다. DP법은 다음 가치 함수의 추정치를 이용하여 현재 가치 함수의 추정치를 갱신한다. 이를 부트스트랩이라고 한다. 반면 MC법은 실제로 얻은 일부 경험만을 토대로 가치 함수를 갱신한다. 이 두방법을 융합한 것이 TD법이다.



TD법은 그림과 같이 다음 행동과 가치 함수만을 이용하여 현재 가치 함수를 갱신한다. DP처럼 부트스트랩을 통해 가치 함수를 순차적으로 갱신하고, MC처럼 환경에 대한 정보 없이 샘플링된 데이터 만으로 가치 함수를 갱신한다. 이 TD법을 수식에서 도출해볼 수 있다.

$$v_\pi(s) = \sum_{a,s'} \pi(a \mid s)\, p(s' \mid s,a)\{r(s,a,s') + \gamma v_\pi(s')\}$$
$$= \mathbb{E}_\pi[R_t + \gamma v_\pi(S_{t+1}) \mid S_t = s]$$

모든 후보에 대해 $r(s,a,s') + rv_\pi(s')$를, 즉 보상과 다음 가치 함수를 계산한다. 이를 기댓값 형태로 다시 쓰면 아래의 식이 된다. TD법에서는 식을 이용하여 가치 함수를 갱신하는데 $R_t + \gamma v_\pi(S_{t+1})$ 부분을 샘플 데이터에서 근사한다. 그래서 TD법 갱신식을 수식으로 표현하면 다음과 같다.

$$V'_\pi(S_t) = V_\pi(S_t) + \alpha\{R_t + \gamma V_\pi(S_{t+1}) - V_\pi(S_t)\}$$

V_π는 가치 함수의 추정치이고 $R_t + \gamma v_\pi(S_{t+1})$은 목적지(목표)이다. 이 목적지를 TD 목표라고 하며, TD법은 목표 방향으로 갱신한다.

6.1.2. MC법과 TD법 비교

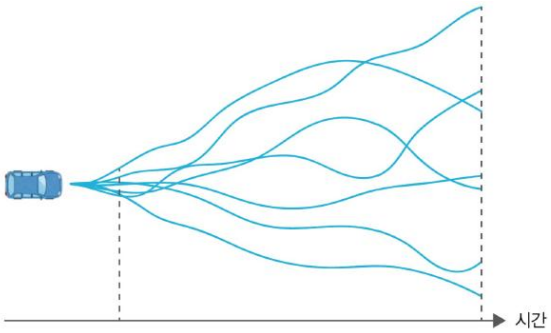
환경 모델을 모를때는 MC법과 TD법을 사용할 수 있다. 지속적인 과제에서는 TD법을 쓸 수 밖에 없다. 현실의 많은 문제에서는 TD법이 가치 함수 갱신이 더 빨라 더 빠르게 학습한다.

〈MC법〉 $V'_\pi(S_t) = V_\pi(S_t) + \alpha\{G_t - V_\pi(S_t)\}$

〈TD법〉 $V'_\pi(S_t) = V_\pi(S_t) + \alpha\{R_t + \gamma V_\pi(S_{t+1}) - V_\pi(S_t)\}$

MC법은 G_t를 목표로 하여 갱신한다. 여기서 G_t는 목표에 도달했을 때 얻을 수 있는 수익의 샘플 데이터이다. 반면 TD법의 목표는 한 단계 앞의 정보를 이용해 계산한다. 이 경우 시간이 한 단계씩 진행될 때마다 가치 함수를 갱신할 수 있기 때문에 효율적인 학습을 기대할 수 있다.

또한, MC법의 목표는 많은 시간을 쌓아서 얻은 결과이기 때문에 변동이 심해 분산이 크다. 반면 TD법은 한 단계 앞의 데이터를 기반으로 하여 변동이 작다.



달리는 자동차의 움직임을 묘사하고 있다. 운전자는 확률적으로 핸들을 오른쪽 또는 왼쪽으로 돌리거나 그대로 둘 수 있다. 선이 여러 개인 이유는 여러 가지 가능성을 그렸기 때문이다. 여기서 주목할 점은 시간이 지날수록 이동 경로의 변동이 커진다는 것이다. MC법의 목표는 오랜 시간이 쌓인 결과이기 때문에 분산이 크다. 반면 TD법은 겨우 한 단계 앞의 시간이기 때문에 분산이 작다.

TD법은 추정치로 추정치를 갱신하는 부트스트래핑이다. 이처럼 TD 목표는 추정치를 포함하기 때문에 정확한 값이 아니다. (편향

이 있다) 하지만 편향은 갱신이 반복될 때마다 점점 작아져 결국에는 0으로 수렴한다.

반면 MC법의 목표에는 추정치가 포함되지 않기 때문에 편향이 없다라고 할 수 있다. 또한 MC법의 목표는 많은 시간을 쌓아서 얻은 결과이기 때문에 값의 변동이 심한 편이다. (분산이 크다)

6.1.3. TD법 구현

무작위 정책에 따라 행동하는 에이전트이며, TD법으로 정책을 평가한다.

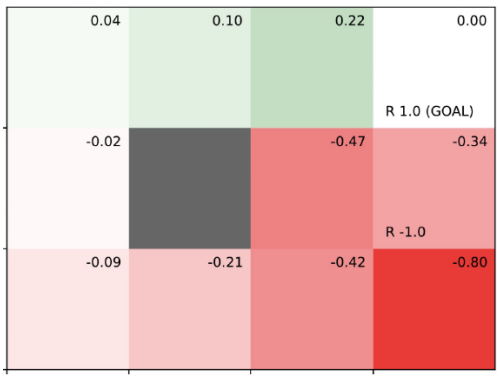
```python
class TdAgent:
    def __init__(self):
        self.gamma = 0.9
        self.alpha = 0.01
        self.action_size = 4

        random_actions = {0: 0.25, 1: 0.25, 2: 0.25, 3: 0.25}
        self.pi = defaultdict(lambda: random_actions)
        self.V = defaultdict(lambda: 0)

    def get_action(self, state):
        action_probs = self.pi[state]
        actions = list(action_probs.keys())
        probs = list(action_probs.values())
        return np.random.choice(actions, p=probs)

    def eval(self, state, reward, next_state, done):
        next_V = 0 if done else self.V[next_state]  # 목표 지점의 가치 함수는 0
        target = reward + self.gamma * next_V

        self.V[state] += (target - self.V[state]) * self.alpha
```

eval() 메서드는 상태 state에서 행동 action을 수행하고, 보상 reward를 받고, 다음 상태 next_state로 넘어갔을 때 호출된다. 또한 에피소드가 끝나는지 여부를 나타내는 플래그 done도 매개변수로 받는다.

에이전트에게 1000번의 에피소드를 실행하도록 한다.

```python
env = GridWorld()
agent = TdAgent()

episodes = 1000
for episode in range(episodes):
    state = env.reset()

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

        agent.eval(state, reward, next_state, done)  # 매번 호출
        if done:
            break
        state = next_state

env.render_v(agent.V)
```

MC법과의 가장 큰 차이는 에이전트의 eval() 메서드를 매번 호출한다는 점인데, TD법은 시간이 한 단계씩 진행될 때마다 갱신하기 때문이다. 반면 MC법에서는 목표에 도달해야만 eval() 메서드를 호출한다.



6.2. SARSA

앞 절에서는 TD법으로 정책을 평가했다. 정책 평가가 끝나면 다음 단계는 정책 제어이다. 이번에는 '온-정책'에 속하는 SARSA 기법을 사용할 수 있다.

6.2.1. 온-정책 SARSA

정책을 제어할 때는 상태 가치 함수가 아닌 행동 가치 함수 Q 함수가 대상이다. 개선 단계에서는 정책을 탐욕화해야하며, $V_\pi(s)$의 경우 환경 모델이 필요하다. 반면 $Q_\pi(s,a)$라면 다음 식처럼 계산할 수 있다.

$$\mu(s) = \underset{a}{\operatorname{argmax}}\, Q_\pi(s,a)$$

환경 모델이 필요하지 않다. 앞 절에서 도출한 상태 가치 함수 $V_\pi(s)$의 갱신식은 식으로 나타낼 수 있다.

$$V'_\pi(S_t) = V_\pi(S_t) + \alpha\{R_t + \gamma V_\pi(S_{t+1}) - V_\pi(S_t)\}$$

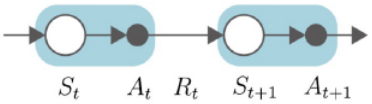
여기서 상태 가치 함수를 Q 함수로 바꾼다. $V_\pi(S_{t+1})$을 $Q_\pi(S_{t+1},A_{t+1})$로 대채하고, $V_\pi(S_t)$를 $Q_\pi(S_t,A_t)$로 대체하면 다음과 같다.

$$Q'_\pi(S_t,A_t) = Q_\pi(S_t,A_t) + \alpha\{R_t + \gamma Q_\pi(S_{t+1},A_{t+1}) - Q_\pi(S_t,A_t)\}$$

이 식이 Q 함수를 대상으로 한 TD법의 갱신식이다.

다음으로 온-정책 형태의 정책 제어 방식에 대해 설명하겠다. 온-정책에서 에이전트는 정책을 하나만 가지고 있다. 실제로 행동하는 정책과 평가 및 개선할 정책이 일치하는 것이다.

에이전트가 정책 π에 따라 행동한다고 하자.



Q 함수는 상태와 행동을 묶은 데이터를 하나의 단위로 삼는다. 그림에서는 시간 t에서의 상태와 행동 데이터를 (S_t,A_t), 한 단계 다음 시간의 데이터를 (S_{t+1},A_{t++1})로 묶어줬다. 그림과 같은 데이터 $(S_t,A_t,R_t,S_{t+1},A_{t+1})$을 얻었다면 위의 식에 대입하여 $Q_\pi(S_t,A_t)$를 즉시 갱신할 수 있다. 그리고 이 갱신이 끝나면 바로 개선 단계로 넘어갈 수 있다. 지금 예에서는 $Q_\pi(S_t,A_t)$가 갱신되기 때문에 상태 S_t에서의 정책이 바뀔 수 있다.

$$\pi'(a \mid S_t) = \begin{cases} \underset{a}{\operatorname{argmax}}\, Q_\pi(S_t,a) & (1-\varepsilon\text{의 확률}) \\ \text{무작위 행동} & (\varepsilon\text{의 확률}) \end{cases}$$

식과 같이 ε의 확률로 무작위 행동을 선택하고, 그 외에는 탐욕 행동을 선택한다. 탐욕 행동으로 정책을 개선하고, 무작위 행동으로 탐색을 하는 것이다.

이렇게 평가와 갱신을 번갈아 반복하면 최적에 가까운 정책을 얻을 수 있다. 이 알고리즘이 SARSA이다.

6.3. 오프-정책 SARSA

6.3.1. 오프-정책과 중요도 샘플링

오프-정책에서는 에이전트가 행동 정책과 대상 정책을 따로 가지고 있다. 행동 정책에서는 다양한 행동을 시도하며 샘플 데이터를 폭넓게 수집한다. 그리고 이 샘플 데이터를 이용해 대상 정책을 탐욕스럽게 갱신한다.

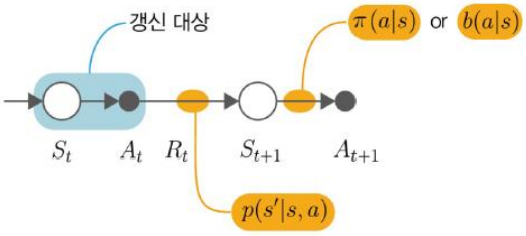
이때 주의할 점이 두 가지 있다.

- 행동 정책과 대상 정책의 확률 분포가 비슷할수록 결과가 안정적이다. 이 점을 고려하면 현재의 Q 함수에 대해 행동 정책은 ε-탐욕 정책으로 갱신하고, 대상 정책은 탐욕 정책으로 갱신한다.
- 두 정책이 서로 다르기 때문에 중요도 샘플링을 활용하여 가중치 p로 보정한다.

$Q_\pi(S_t, A_t)$를 갱신하는 경우를 생각해볼 수 있다. 이때 SARSA의 갱신식은 다음과 같다.

$$Q'_\pi(S_t, A_t) = Q_\pi(S_t, A_t) + \alpha\{R_t + \gamma Q_\pi(S_{t+1}, A_{t+1}) - Q_\pi(S_t, A_t)\}$$

이 갱신식에 대응하는 백업 다이어그램이다.



그림에서 보듯 상태와 행동을 묶은 (S_t, A_t)가 갱신 대상이다. 이때 다음 상태 S_{t+1}은 환경의 상태 전이 확률에 따라 샘플링 된다. 그리고 상태 S_{t+1}에서 선택되는 행동은 대상 정책 π또는 행동 정책 b에 따라 샘플링 된다. 이렇게 얻은 샘플 데이터를 식에 대입하여 $Q_\pi(S_t, A_t)$를 갱신한다. 이때 행동이 정책 π에 따라 선택됨을 명시하면 SARSA의 갱신식을 다음처럼 작성할 수 있다.

$$\text{샘플링: } A_{t+1} \sim \pi$$
$$Q'_\pi(S_t, A_t) = Q_\pi(S_t, A_t) + \alpha\{R_t + \gamma Q_\pi(S_{t+1}, A_{t+1}) - Q_\pi(S_t, A_t)\}$$

위 식은 $Q_\pi(S_t, A_t)$를 $R_t + \gamma Q_\pi(S_{t+1}, A_{t+1})$ 방향으로 갱신함을 나타낸다. 즉 $R_t + \gamma Q_\pi(S_{t+1}, A_{t+1})$이 TD의 목표이다.

다음으로 행동 A_{t+1}이 정책 b에 따라 샘플링된 경우를 생각해볼 수 있다. 이 경우 가중치 ρ로 TD 목표를 보정한다. (중요도 샘플링) 가중치 ρ는 '정책이 π일 때 TD 목표를 얻을 확률'과 '정책이 b일 때 TD 목표를 얻을 확률'의 비율을 말한다. 수식으로는 다음과 같다.

$$\rho = \frac{\pi(A_{t+1} \mid S_{t+1})}{b(A_{t+1} \mid S_{t+1})}$$

따라서 오프-정책 SARSA의 갱신식은 다음과 같다.

$$\text{샘플링: } A_{t+1} \sim b$$
$$Q'_\pi(S_t, A_t) = Q_\pi(S_t, A_t) + \alpha\{\rho(R_t + \gamma Q_\pi(S_{t+1}, A_{t+1})) - Q_\pi(S_t, A_t)\}$$

이와 같이 행동은 정책 b에 따라 샘플링되고 가중치 ρ로 TD 목표가 보정된다.

6.4. Q 러닝

오프-정책 방식에서는 에이전트가 행동 정책과 대상 정책을 따로 가지고 있었다. 두 정책이 역할을 분담하여 행동 정책으로는 '탐색'을, 대상 정책으로는 '활용'을 수행하는 것이다. 이렇게 하면 최적의 정책을 할 수 있다. 하지만 오프-정책 SARSA에서는 중요도 샘플링을 이용해야 한다.

하지만, 중요도 샘플링은 결과가 불안정하기 쉽다는 문제를 안고 있다. 특히 두 정책의 확률 분포가 다를수록 중요도 샘플링에서 사용하는 가중치 ρ도 변동성이 커진다. 따라서 SARSA의 갱신식에 등장하는 목표도 변동되기 때문에 Q 함수의 갱신 역시 불안해진다.

이 문제를 해결하는 것이 Q 러닝이다. Q러닝은 TD법, 오프-정책, 중요도 샘플링을 사용하지 않는 것이 특징이다.

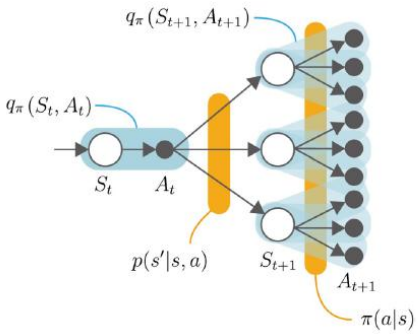
Q 러닝을 도출하기 위해 벨만 방정식과 SARSA의 관계부터 확인할 수 있다. 그런 다음 벨만 최적 방정식에서 Q 러닝을 도출할 수 있다.

6.4.1. 벨만 방정식과 SARSA

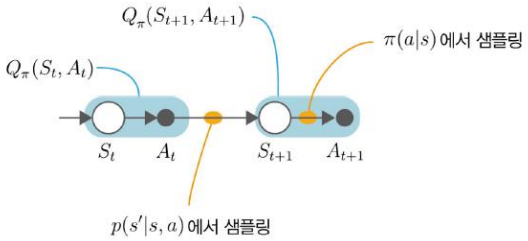
정책 π에서의 Q 함수를 $q_\pi(s,a)$라고 했을 때 벨만 방정식은 다음 식으로 표현된다.

$$q_\pi(s,a) = \sum_{s'} p(s' \mid s,a)\left\{r(s,a,s') + \gamma \sum_{a'} \pi(a' \mid s') q_\pi(s',a')\right\}$$

이 식에서 중요한 점은 환경의 상태 전이 확률에 따른 다음 단계의 '모든'상태 전이를 고려한다는 점과 에이전트의 정책 π에 따른 '모든'행동을 고려한다는 것이다. 벨만 방정식의 백업 다이어그램이다.



그림처럼 벨만 방정식은 다음 상태와 다음 행동의 모든 후보를 고려한다. 따라서 SARSA는 벨만 방정식의 '샘플링 버전'으로 볼 수 있다. 이는 모든 전이가 아니라 샘플링된 데이터를 사용한다는 것이다.



그림처럼 SARSA에서 다음 상태 S_{t+1}은 $p(s'|s,a)$로부터 샘플링 한다. 그리고 다음 행동 A_{t+1}은 정책 $\pi(a|S)$로부터 샘플링 한다.

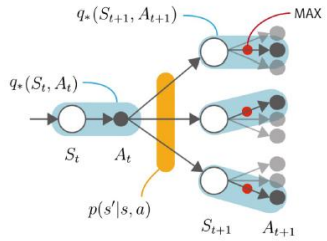
6.4.2. 벨만 최적 방정식과 Q 러닝

가치 반복법은 최적 정책을 얻기 위한 평가와 개선의 두 과정을 하나로 묶은 기법이다. 가치 반복법의 중요한 점은 벨만 최적 방정식에 기반하여 단 하나의 갱신식을 반복함으로써 최적 정책을 얻을 수 있다는 것이다. 이번 절에서는 벨만 최적 방정식에 의한 갱신인 동시에 이를 '샘플링 버전'으로 만든 방법을 알아보겠다.

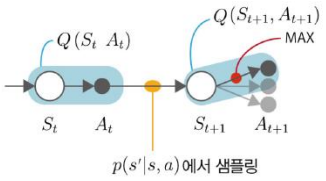
먼저 Q 함수의 벨만 최적 방정식은 다음과 같다.

$$q_*(s,a) = \sum_{s'} p(s' \mid s,a)\left\{r(s,a,s') + \gamma \max_{a'} q_*(s',a')\right\}$$

벨만 최적 방정식을 백업 다이어그램으로 표현하면 다음과 같다.



샘플링 버전으로 다시 작성해보면



그림에 기반한 방법이 Q 러닝이다. Q 러닝에서 추정치 $Q(S_t, A_t)$의 목표는 $R_t + \gamma max_a Q(S_{t+1}, a)$가 된다. 이 목표 방향으로 Q 함수를 갱신한다. 수식으로는 다음과 같다.

$$Q'(S_t,A_t) = Q(S_t,A_t) + \alpha\left\{R_t + \gamma \max_a Q(S_{t+1},a) - Q(S_t,A_t)\right\}$$

식에 따라 Q 함수를 반복해서 갱신하면 최적 정책의 Q 함수에 가까워진다. 그림에서 중요한 점은 Q 함수가 가장 큰 행동으로 A_{t+1}을 선택한다는 것이다. 특별한 정책에 따라 샘플링하지 않고 max 연산자로 선택한다. 따라서 중요도 샘플링을 이용한 보정이 필요 없다.

Q 러닝은 오프-정책 기법으로 대상 정책과 행동 정책을 따로 가지며 행동 정책으로는 탐색을 수행한다. 행동 정책은 현재 추정치인 Q 함수를 ε-탐욕화한 정책이다. 행동 정책이 결정되면 그에 따라 행동을 선택하여 샘플 데이터를 수집한다. 그리고 에이전트가 행동할 때마다 위의 식으로 Q 함수를 갱신한다.