강화학습

학습목표

- 강화학습의 원리를 이해한다.
- 동적 계획법의 개념을 이해한다.
- 몬테카를로 방법의 개념을 이해한다.
- 시간차 학습의 다양한 유형을 이해한다.
- 심층 Q-망을 이해하고 익힌다.

1. 개요

- 2016년 3월 전 세계가 주목한 세기의 대결이 펼쳐졌다. 구글 딥마인드가 개발한인공지능 바둑 프로그램 알파고와 바둑 세계 챔피언 이세돌의 대결이었다. 바둑은 인간이 만든 게임 중 가장 복잡하다. 왜냐하면 바둑의 경우의 수는 약 10^{171}으로, 우주 전체의 원자 수보다 많기 때문이다. 따라서 사람들은 이세돌이 승리할 것으로 생각했지만 알파고가 이세돌을 4:1로 제압하였다. 컴퓨터가 인간을 이길 수 없는 마지막 보루라고 생각한 바둑마저도 컴퓨터에 챔피언의 자리를 내어 준 것이다.
- 알파고를 계기로 알파고의 핵심 기술인 강화학습에 대한 관심이 폭발적으로 증가하여 이에 대한 연구가 활발히 진행되었다. 이를 가능하게 한 데에는 딥마인드, OpenAI와 같은 연구기관들이 공개한 오픈소스의 역할이 컸다.

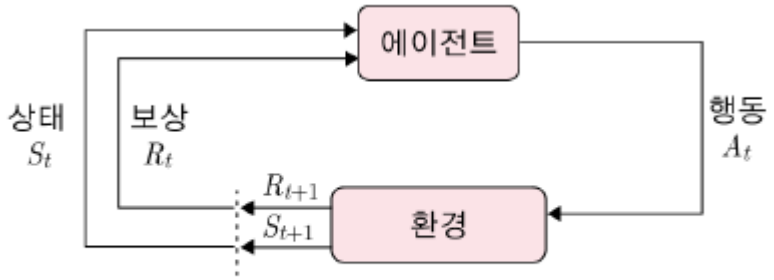


그림 7-1 에이전트-환경 상호작용 구조

1. 개요

- 강화학습은 기계학습의 한 분야로서, 주어진 환경에서 에이전트가 현재의 상태를 파악하여 선택 가능한 행동들 중 누적 보상을 최대화하는 행동을 선택하는 정책을 찾는 방법이다. 이러한 문제는 매우 포괄적이기 때문에 게임이론, 제어이론, 정보이론, 모의실험 기반 최적화, 다중 에이전트 시스템, 통계학, Operations Research, 유전 알고리즘 등의 분야에서도 연구한다.
- 고전적 동적 계획법과 강화학습의 주된 차이점을 보면, 강화학습은 MDP의 정확한 수학적 모형에 대한 지식을 가정하지 않고, 정확한 방법이 실행불가능한 대규모 MDP를 대상으로 한다는 것이다. 즉, 강화학습은 시스템에 매우 많은 수의 상태가 있고 복잡한 확률적 구조를 가질 때 사용된다. 반면, 동적 계획법은 문제가 비교적 적은 수의 상태를 가지며 확률적 구조가 상대적으로 단순할 때 사용된다.
- 강화학습은 지도학습 및 비지도학습과 함께 세 가지 기계학습 패러다임 중 하나로 간주된다. 강화학습은 정확한 입력과 출력의 쌍이 필요하지 않으며 차선의 행동을 명시적으로 수정하지 않는다는 점에서 지도학습과 다르다. 대신 강화학습은 탐색과 이용 사이의 균형을 찾는 데 초점을 맞춘다. 탐색과 이용의 절충 문제는 다중 슬롯머신 문제와 유한 MDP를 통해 철저히 연구되었다.

1. 개요

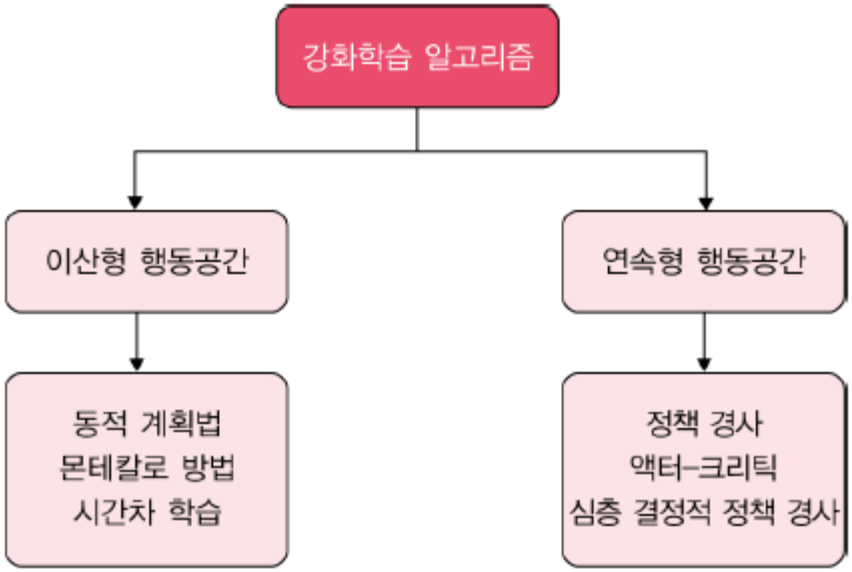


그림 7-2 강화학습 알고리즘의 종류

- [그림 7-2]에서처럼 강화학습 알고리즘은 행동공간action space의 형태에 따라 크게 두 종류로 나누어진다. 아타리와 보드게임 같은 게임에서 문제는 이산형 행동공간을 다룬다. 그리고 모의실험과 로봇 제어 관련 문제는 연속형 행동공간을 다룬다. 강화학습 알고리즘 중 Q-학습 알고리즘이 가장 대표적인데, 이 알고리즘은 시간차 학습과 관련 있다. 이 장에서는 강화학습의 기본개념과 여러 가지 알고리즘을 설명한 후 심층 Q-망 학습까지 다루는데, 심층 Q-망 학습에 초점을 맞춘다고 생각하면 된다.

2. 강화학습의 원리

1. 강화학습의 요소

A. 상태 행동 보상

- 강화학습은 시간에 따라 순차적으로 작업을 수행한다. 시간을 나타내는 변수 t 는 연속값일 수도 있지만, 일반적으로 이산값을 사용한다. 예를 들어, 바둑 또는 장기에서 한 수를 둘 때마다 다음 단계로 넘어가기 때문에 시간 t 는 이산값을 취한다.
- 시간 t 는 0에서 시작하여 편의상 1씩 증가한다고 보고 t 는 0,1,2,3, … 으로 표기한다.
- 각 시점 t 에 에이전트는 환경의 상태 $S_t \in \boldsymbol{S}$ 를 관측하며 이를 바탕으로 행동 $A_t \in \boldsymbol{A}(S_t)$ 를 선택한다.
- $\boldsymbol{S}$: 모든 가능한 상태의 집합 (상태공간) $\boldsymbol{A}$: 모든 가능한 행동의 집합 (행동공간)
- $\boldsymbol{A}(S_t)$: S_t 에서의 모든 가능한 상태의 집합
- 행동의 결과로서 1시점이 지난 후에 보상 $R_{t+1} \in \boldsymbol{R}$ 을 받으며 새로운 상태 S_{t+1} 을 찾는다.
- 보상은 에이전트가 선택한 행동에 대한 환경의 반응이다. 이러한 방식을 반복하다 보면 종료 상태에 도달할 수 있다. 종료시점을 T 로 표기하자. 앞의 [그림 7-1]은 에이전트-환경의 상호작용을 보여 준다.

2. 강화학습의 원리

- 종료 상태는 중간 상태의 영향을 받아 성립되었으므로 종료 시점의 보상을 이전 상태에 나눠 주어야 한다. 이 문제를 신용 할당 문제 라고 한다. 강화학습에서는 보상 책정을 뒤로 미루는 경우가 많으므로 신용 할당 문제를 잘 다루어야 한다.

B. 에이전트와 환경

- 강화학습은 어떤 환경에서 정의된 에이전트가 현재의 상태를 인식하여 선택 가능한 행동들 중 보상을 최대화하는 행동 또는 행동 순서를 선택하는 방법이다. 강화학습에서는 [그림7-1]에서처럼 행동을 결정하고 상태 전이와 보상을 결정하는 주체가 필요하다. 그 주체가 에이전트와 환경인데, 에이전트는 행동을 결정하며 환경은 상태 전이와 보상 결정을 담당한다. MDP는 문제를 정의할 때 주어지는 정보이다. 환경은 MDP에 따라 상태와 보상을 결정하고, 에이전트는 정책에 따라 행동을 결정한다. 강화학습은 주어진 MDP에서 최적의 행동을 결정하는 정책을 찾는다.

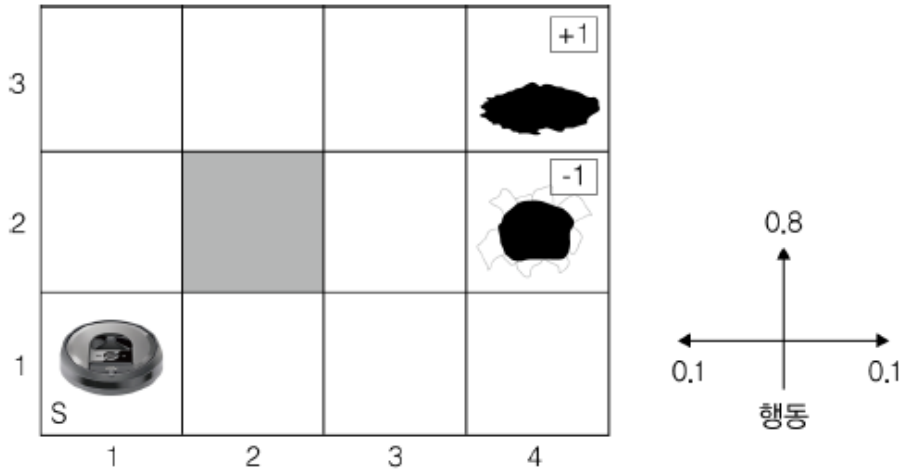


그림 7-3 3×4 격자보드 예제의 강화학습 환경

2. 강화학습의 원리

- 에이전트는 환경의 상태를 어느 정도 감지할 수 있어야 하며 상태에 영향을 미치는 행동을 취할 수 있어야 한다. 또한 에이전트는 환경의 상태와 관련된 하나 또는 여러 개의 목표를 가지고 있어야 하며, 자신의 경험을 통해 학습할 수 있어야 한다. 환경은 보통 에이전트를 제외한 나머지로 간주되는데, 물리적으로 정의하기 힘든 경우가 많다. 에이전트와 환경은 물리적으로 명확한 대상체가 있기도 하지만, 대부분 추상적인 대상체이다.
- [그림 7-3]은강화학습을 설명하는 간단한 3 ×4 격자보드 예제이다. 로봇청소기는 현재 위치에서 동서북 세 방향으로 이동할 수 있으며, 에이전트는 로봇청소기의 모든 전이에 대하여 보상 –0.001을 받는다. 그리고 로봇청소기가 먼지를 발견하면 보상 +1을 받고 구멍에 빠지면 보상 –1을 받는다. 이 예제에서 에이전트는 로봇이며 환경은 3 ×4 격자보드이다. 즉, 강화학습은 에이전트와 환경이 행동 . 상태 . 보상이라는 정보를 통해 상호작용하면서 목표를 달성하는 학습 과정이다.
- 두가지 종류의 환경을 생각해 보자.

  - 결정적 환경: 결정적 환경이란 상태전이 모형과 보상 모형 모두 결정적 함수라는 것을 의미한다. 만약 에이전트가 특정 상태에서 특정 행동을 반복하면 에이전트는 항상 동일한 다음 상태로 가게 되고 동일한 보상을 받는다.
  - 확률적 환경: 확률적 환경에서는 행동 결과에 대해 불확실성이 있다. 에이전트가 특정 상태에서 동일한 행동을 반복할 때 다음 상태와 보상이 항상 같지 않을 수 있다. 예를 들어, 로봇 작동의 불완전성 또는 환경의 다른 요인(예: 미끄러운 바닥)으로 인해 전진하려는 로봇은 때로는 전진하게 되지만, 때로는 의도치 않게 왼쪽이나 오른쪽으로 이동하기도 한다.

2. 강화학습의 원리

- 결정적 환경은 해결하기가 더 쉽다. 왜냐하면 환경 MDP가 주어진다면 에이전트는 불확실성 없이 자신의 행동을 계획할 수 있기 때문이다. [그림 7-4]는 결정적 환경과 확률적 환경의 예를 보여 준다.

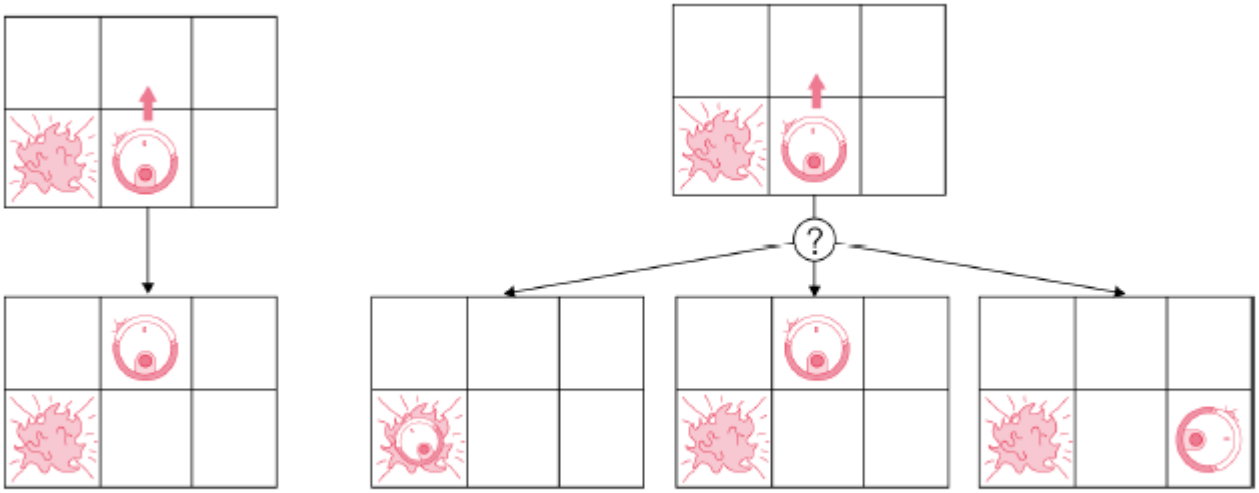


그림 7-4 결정적 격자세계(왼쪽)와 확률적 격자세계(오른쪽)

C. 정책

- MDP의 목표는 에이전트를 위해 최적의 정책을 찾는 것이다. 정책은 에이전트가 현재 상태에 근거하여 행동하는 방식이다. 수학적으로 표현한다면, 정책은 에이전트가 상태 s 에서 취할 행동 a를 선택하는 함수 π_t로 나타낼 수 있다. 일반적으로 정책 π_t 는 두 가지 방식으로 정의되고, 정책은 에이전트의 행동을 완전히 결정한다.

  - 결정적deterministic 정책: $a = \pi_t(s)$
  - 확률적deterministic 정책: $\pi_t(a \mid s) = P(A_t = a \mid S_t = s)$ (7.2.1)

- [그림 7-3]은 모든 상태에서 동, 서, 남, 북을 같은 확률로 선택하는 정책을 사용하였다고 가정했을 때, 이 정책은 다음과 같이 표현할 수 있다. 종료 상태를 뺀 나머지 14가지 상태 모두 동일 확률로 동, 서, 남, 북을 선택한다.

$$\text{정책: } P(\text{동} \mid i) = P(\text{서} \mid i) = P(\text{남} \mid i) = P(\text{북} \mid i) = \frac{1}{4},\ i = 1, 2, \cdots, 14$$

- 강화학습의 목표는 '누적' 보상을 최대화하는 정책을 찾는 것이다

2. 강화학습의 원리

D. 반환값과 가치함수

- 보상 R_t는 각 시점 [t □□ 에이전트가 받게 되는 값이다. 현재 시점 |t| □□□ 에이전트가 행동을 하면서 받게 될 보상들을 합하면 식 (7.2.2)와 같다.

$$R_{t+1} + R_{t+2} + R_{t+3} + R_{t+4} + \cdots \qquad (7.2.2)$$

- 보상은 행동을 했을 때가 아닌 그다음 시점에서 받는다는 것을 기억하자. 따라서 시점 t 에 행동을 하여 받게 되는 보상은 R_{t+1}이다. 여기서 대문자로 표기한 R 은 확률변수이다.
- 에이전트가 시점마다 보상을 받을 수도 있고 게임이 끝날 때 한 번에 받을 수도 있다. 그런데 에이전트 입장에서는 지금 받는 보상이나 미래에 받는 보상이나 똑같이 취급한다면 문제가 생긴다. 왜냐하면 200이라는 보상을 1번 받는 것과 50이라는 보상을 4번 받는 것을 구분할 방법이 없기 때문이다. 그리고 시간이 무한대라고 하면 보상을 시점마다 0.2씩 받아도 합이 무한대가 되고 1씩 받아도 무한대가 되기 때문이다. 따라서 할인율을 이용하여 미래에 받게 될 보상의 현재 가치를 더하는 것이 필요하다. 이것을 반환값이라고 하며, 식 (7.2.3)으로 정의한다. 반환값은 에이전트가 실제로 환경을 탐색하며 받은 보상의 합이다. 그리고 반환값은 확률변수라는 것을 기억하자. 에이전트의 목표는 반환값을 최대화하는 정책을 찾는 것이다.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \qquad (7.2.3)$$

2. 강화학습의 원리

- 여기서 $\gamma \in [0,1]$ 은 할인율이다. 만약 할인율이 0이면 미래에 받게 될 보상이 모두 0이 되어 고려하지 않게 된다. 따라서 바로 다음의 보상만 고려하는 근시안적인 행동을 하게 될 것이다. 반대로 할인율이 1에 가까우면 미래 보상에 대한 할인이 작아지기 때문에 미래 보상을 더 많이 고려하는 원시안적인 행동을 하게 될 것이다. γ 는 현재의 보상이 미래에 얻게 되는 보상보다 얼마나 더 중요한지를 나타내는 값이다. 즉, γ는 지연 보상과 즉각보상의 상대 가치를 결정하는 값이다. 가치함수는 반환값의 기댓값으로, 두 종류가 있다. 그것은 다음과 같이 정의되는 상태 가치함수와 행동 가치함수이다.

  - 상태 가치함수: $v(s) = E[G_t \mid S_t = s]$
  - 행동 가치함수: $q(s,a) = E[G_t \mid S_t = s, A_t = a]$ (7.2.4)

2. 탐색과 이용

- 지식과 편견이 부족한 어린아이의 행동은 대체로 거침이 없다. 위험한지 모른 채 높은 곳에 올라가 넘어지기도 하고 아무것이나 입에 넣어보기도 한다. 그 결과 종종 다치기도 하고 부모한테 혼나기도 하면서 세상 또는 환경에 대한 경험을 쌓아가며 올바른 행동 방식을 찾아간다. 이렇게 성장한 아이가 성인이 되면 위험하지 않고 안전하게 효율적인 방식으로 행동하게 된다. 본인이 이해할 수 있는 세상 또는 환경 속에서 가장 바람직한 행동을 선택하여 행동한다. 그러나 새로운 시도를 통하여 배우는 빈도는 점점 더 줄어든다. 즉, 사람은 나이가 들면 기존의 경험을 이용하며 보수적으로 생각하고 살아가게 된다.
- 사람과 비슷하게 에이전트는 환경 내에서 매 순간 어떤 결정을 내리고 행동을 한다. 초기단계에서 행동 대부분은 경험을 쌓기 위한 무작위에 가까운 행동이다. 그리고 이러한 행동의 결과에 대해 환경은 에이전트에 보상을 준다. 에이전트는 이를 경험에 저장한다. 에이전트는 점점 경험을 축적하며 이에 따라 무작위 행동을 하는 빈도를 낮추며 보수적으로 행동하게 된다. 정리하면 강화학습은 무작위 행동빈도를 낮추면서 환경이 주는 보상에 대한 경험을 쌓아 환경의 다양한 상태에 대응할 수 있는 최적의 정책을 찾아가는 학습방법이다.

2. 강화학습의 원리

- 강화학습에서 에이전트는 보상을 얻기 위해 이미 알고 있는 행동을 이용exploitation해야 하지만, 이와 함께 앞으로 더 바람직한 행동을 선택하기 위해 탐색도 반드시 필요하다.
- 그러나 행동 선택에서 탐색과 이용이 함께 이루어지는 것은 불가능하므로, 탐색과 이용 사이의 충돌이 발생한다. 따라서 에이전트에 있어서 탐색과 이용의 균형을 이루는 것은 매우 중요하다. 탐색을 하는 것이 바람직한지 아니면 이용하는 것이 더 바람직한지는 여러 가지 요인에 근거하여 평가한다. 탐색과 이용의 관계를 설명하기 위해 다음의 예제를 생각해보자.

  • 식당 선택

  이용: 가장 좋아하는 식당으로 간다.

  활용: 새로운 식당을 시도한다.

  • 온라인 배너 광고

  이용: 가장 성공적인 광고를 보여 준다.

  활용: 다른 광고를 보여 준다.

  • 석유 시추

  이용: 가장 잘 알려진 위치에서 시추한다.

  활용: 새로운 위치에서 시추한다.

  • 게임 플레이

  이용: 가장 좋았다고 생각하는 동작을 플레이한다.

  활용: 실험적인 동작을 플레이한다.

3. MDP

- 강화학습은 순차적으로 계속 행동을 결정해야 하는 문제를 푸는 것이다. MDP는 이러한 의사결정 과정을 모형화하는 수학적인 틀이다. 동적 계획법은 MDP를 계산으로 푸는 방법이고, 강화학습은 MDP를 학습으로 푸는 방법이라고 생각할 수 있다.

A. 마르코프 과정

- 마르코프 과정은 미래의 상태 S_{t+1}은 현재 상태 S_t 에 영향을 받지만, 과거 상태 $S_{t-1}, S_{t-2},$ 에는 영향을 받지 않는 마르코프 성질을 가진 상태 S_t 에 대한 확률 과정이다. 마르코프 성질을 수학적으로 표현하면 식 (7.2.5)와 같다.

$$P(S_{t+1} = s_{t+1} \mid S_t = s_t) = P(S_{t+1} = s_{t+1} \mid S_0 = s_0, \cdots, S_t = s_t) \qquad (7.2.5)$$

- 마르코프 과정은 튜플$< \boldsymbol{S}, \boldsymbol{P} >$로 표기하겠다. 여기서 $\boldsymbol{S}$ 는 상태집합이고 $\boldsymbol{P}$ 는 상태전이확률행렬이다. 상태전이확률은 다음과 같이 정의된다.

$$P_{ss'} = P(S_{t+1} = s' \mid S_t = s), \quad \forall s, s' \in \boldsymbol{S} \qquad (7.2.6)$$

2. 강화학습의 원리

- 상태전이확률행렬 [P는 상태 s 에서 다음 상태 Is '으로 전이하는 확률을 의미한다. 상태집합 S 가 유한집합일 때, 즉 $S=\{s_1, s_2, \cdots, s_n\}$ 口 때 P는 다음과 같이 표현할 수 있다.

$$P=\begin{bmatrix} P_{11} \cdots P_{1n} \\ \ddots \\ P_{n1} \cdots P_{nn} \end{bmatrix} \qquad (7.2.7)$$

- 이때 행렬의 각 열의 원소들을 합하면 1이 된다. 격자세계와 바둑은 마르코프 성질을 만족한다. 바둑의 경우를 생각해 보자. 만약 우리가 어떤 시점의 흑돌과 백돌의 배치 그리고 서로 얼마나 상대편의 돌을 땄는지 등의 정보를 알고 있다면, 다음 수를 두기 위해 바둑이 지금까지 어떻게 진행되었는지에 대한 모든 과거 정보가 필요한 것은 아니다. 왜냐하면 바둑은 마르코프 성질을 가지기 때문이다. 그러나 실제 문제에서 마르코프 성질이 성립하지 않을 때가 많다.

B. 마르코프 보상 과정

- 마르코프 보상 과정은 마르코프 과정에 보상 개념을 추가한 것이다. MRP는 튜플 $< S, P, R, \gamma >$ 로 표기하겠다. 여기서 S 는 상태집합이고 P 는 상태전이확률행렬이며 R은 다음과 같이 정의되는 보상함수이다.

$$R_s = E(R_{t+1} \mid S_t = s) \qquad (7.2.8)$$

- $\gamma \in [0,1]$ 은 할인율이다.

C. 마르코프 결정 과정

- 마르코프 결정 과정은 MRP에 행동 개념을 추가한 것이다. 즉, MDP는 튜플 $< S, A, P, R, \gamma >$ 로 표현된다. 여기서 S 는 상태집합이고 A 는 행동집합이며 P 는 원소가 다음과 같이 정의되는 상태전이확률행렬이다.

$$P_{ss'}^{a} = P(S_{t+1} = s' \mid S_t = s, A_t = a) \qquad (7.2.9)$$

- 그리고 R 은 다음과 같이 정의되는 보상함수이다.

$$R_s^a = E(R_{t+1} \mid S_t = s, A_t = a) \qquad (7.2.10)$$

- $\gamma \in [0,1]$ 은 할인율이다.
- MDP는 행동 결정 문제를 수학적으로 정의한 것이다. MDP는 상태, 행동, 보상, 상태전이 확률, 할인율 및 정책으로 구성되어 있다. MDP에서 관측값은 바로 환경의 상태와 같다는 의미에서 시스템은 MDP에서 완전히 관측할 수 있다. MDP의 관련 경로는 다음과 같이 유한경로와 무한경로로 표현된다.

유한경로 $z : (S_t, R_t) \xrightarrow{A_t} (S_{t+1}, R_{t+1}) \xrightarrow{A_{t+1}} (S_{t+2}, R_{t+2}) \xrightarrow{A_{t+2}} \cdots \xrightarrow{A_{T-1}} (S_T, R_T)$

무한경로 $z : (S_t, R_t) \xrightarrow{A_t} (S_{t+1}, R_{t+1}) \xrightarrow{A_{t+1}} (S_{t+2}, R_{t+2}) \xrightarrow{A_{t+2}} (S_{t+3}, R_{t+3}) \cdots$

2. 강화학습의 원리

- 특정 조건이 만족되면 환경은 이 경로를 종료하는데, 이를 에피소드라고 한다.
- 즉, 에피소드는 종료 상태가 되면 만들어진다. 상태, 행동, 보상으로 MDP의 한 에피소드episode를 만든다. 에피소드는 에이전트와 환경의 상호작용 결과이다.

3. 정책과 가치함수

- 강화학습의 핵심은 좋은 정책을 찾는 것이다. 사실 정책은 에이전트가 행동을 결정하기위해 사용하는 알고리즘이다. 좋은 정책이 있으면 누적 보상을 최대로 하는 최적 행동을 매순간 선택할 수 있다. 강화학습에서 좋은 정책을 찾는 것은 지도학습에서 목적함수의 최적 해를 찾는 일에 비유할 수 있다. 지도학습에서 확률적 경사 하강법을 이용하여 최적해를 찾아가듯이 강화학습도 적절한 최적화 알고리즘을 사용하여 최적 정책을 찾는다.
- 이 절에서는 정책공간이 일반적으로 매우 방대하여 최적 정책을 직접 찾는 것은 어렵기 때문에 최적 정책을 찾는 데 길잡이 역할을 하는 가치함수를 소개하고, 가치함수를 이용하여 최적 정책을 찾는 방법을 설명한다.

1. 정책

- 7.2절에서 정의한 결정적 정책과 확률적 정책을 여기에 다시 언급한다. MDP와 관련된 정책은 일반적으로 시점 t 에 의존하지 않는 정상 정책이므로 시점 t 를 제거한 π 를 사용한다.

  - 결정적deterministic 정책: $a = \pi(s)$
  - 확률적stochastic 정책: $\pi(a \mid s) = P(A_t = a \mid S_t = s)$ (7.3.1)

3. 정책과 가치함수

- 결정적 정책은 결정적 환경에서 사용된다. 결정적 환경은 취한 행동이 결과를 결정하는 환경이다. 따라서 불확실성이 존재하지 않는다. 반면, 확률적 정책은 행동에 대한 확률분포를 결과로 제공한다. 확률적 정책은 환경이 불확실성을 가지고 있을 때 사용된다. 우리는 이러한 확률 과정을 POMDP라고 부른다.
- 확률적 정책은 일반적으로 두 가지 주요 문제에서 결정적 정책보다 강건하다.

  • 확률적 환경: 환경 자체가 확률적일 때 결정적 정책은 똑같은 상태에서는 항상 똑같은 행동을 선택하므로 실패할 것이다. 확률적 정책은 학습된 확률분포에 따라 행동을 선택한다. 예를 들어, 가위바위보 게임을 생각해 보자. 최적 정책은 항상 가위, 바위, 보를 동일한 확률로 선택한다. 이러한 종류의 행동 선택은 확률적 정책으로 쉽게 학습할 수 있지만, 결정적 정책으로는 불가능하다. 실제 응용에서 환경의 확률적 특성은 매우 보편적이다.

  • 부분 관측 가능 상태: 완전 관측 가능 시스템, 즉 MDP에서는 관측값으로 상태를 얻는다. 그러나 부분 관측 가능 시스템에서는 관측값으로 상태의 함수를 얻는다. 즉, 관측값이 모든 정보를 포함하는 것은 아니며 상태의 일부가 숨겨져 있다. 따라서 확률적 정책은 숨겨진 상태를 추론하는 것에 대해 불확실성을 자연스럽게 고려하므로 확률적 정책이 더 강건하다. 봇이 포커를 한다고 생각해 보자. 봇은 다른 참가자가 가지고 있는 카드에 대해 추측하게 된다. 이때 행동 선택은 확률적이어야 할 것이다.

3. 정책과 가치함수

2. 가치함수

- 가치함수는 특정 정책의 성능을 평가하는 함수이며, 여기에는 두 종류가 있다. 상태 가치 함수와 행동 가치함수가 그것이다. 구체적으로 말하면 상태 가치함수는 특정 정책에서 상태가 얼마나 좋은지를 평가하는 가치함수이며, 행동 가치함수는 상태 s에서 행동 a를 취했을 때 이 행동이 얼마나 좋은지를 평가하는 가치함수이다.
- 상태 또는 상태-행동 쌍이 얼마나 좋은지에 대한 개념은 기대 반환값 측면에서 주어진다. 따라서 가치함수는 구체적인 행동방식과 관련하여 정의한다. 가치함수는 특정 정책 π와 관련 있으므로 식 (7.3.2)처럼 $v_\pi(s)$ 와 $q_\pi(s,a)$ 로 표기한다. 상태 가치함수 $v_\pi(s)$ 는 정책 π에서의 상태 s의 가치를 나타내며, 행동 가치함수 $q_\pi(s,a)$ 는 정책 π에서의 행동 a의 가치를 나타낸다. 일반적으로 강화학습에서 에이전트가 행동을 선택하는 기준으로 상태 가치함수보다 행동 가치함수를 사용한다

- 상태 가치함수: $v_\pi(s) = E_\pi[G_t \mid S_t = s]$
- 행동 가치함수: $q_\pi(s,a) = E_\pi[G_t \mid S_t = s, A_t = a]$

(7.3.2)

3. 정책과 가치함수

A. 가치함수를 이용한 정책 평가

- 가치함수를 도입하는 이유는 최적 정책을 찾기 위해서이다. 최적 가치함수를 알면 최적정책도 쉽게 찾을 수 있다. 주어진 정책을 가치함수로 평가하는 것을 생각해 보자. 예제를 통하면 쉽게 이해할 수 있다. 다음 [예제 7-1]은 상태 가치함수를 사용하여 주어진 정책을 평가하는 예제이다.

예제 7-1 **가치함수로 정책 평가하기**

로봇이 돌아다닐 수 있는 4×4 격자세계의 예제를 생각해 보자. [그림 7-5]에서 회색으로 표시된 칸이 종료 상태이다. 종료 상태는 $\{1, 16\}$이고 비종료 상태는 $\{2, 3, \cdots, 15\}$이다. 로봇은 현재 위치에서 동서남북 네 방향으로 이동할 수 있다. 이 격자세계는 결정적deterministic이라고 가정한다. 즉, 특정 상태에서 특정 행동을 취하면 항상 동일한 다음 상태로 가게 된다. 종료 상태에 도착할 때까지 모든 전이에 대해 보상 -1을 받는다. 바깥으로 나가는 행동을 취하면 제자리로 돌아온다고 가정한다. 전이 종료 상태에 들어가면 보상 5를 받는다. 할인율은 $\gamma = 1$이다.

3. 정책과 가치함수

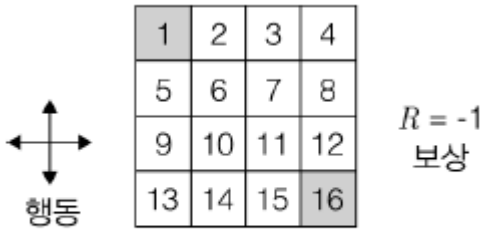


그림 7-5 4×4 격자세계 예제

- 격자세계에 대해 가치함수를 사용하여 정책을 비교해 보자. [그림 7-6]은 정책 π_1 과 정책 π_2를 보여 주는데, π_2 는 종료 상태 1로만 향하는 정책이라는 것을 알 수 있다.

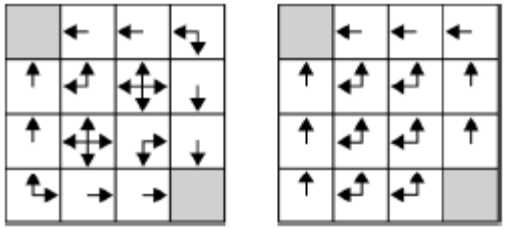


그림 7-6 정책 π_1(왼쪽)과 정책 π_2(오른쪽)

3. 정책과 가치함수

- 먼저 [그림 7-6]의 정책 π_1 에 대해 생각해 보자. 확률을 사용하여 이 정책을 나타내면 다음과 같다.

$$\pi_1 = \begin{cases} P(\text{서} \mid i) = 1, & i = 2, 3 \\ P(\text{북} \mid i) = 1, & i = 5, 9 \\ P(\text{남} \mid i) = 1, & i = 8, 12 \\ P(\text{동} \mid i) = 1, & i = 14, 15 \\ P(\text{서} \mid i) = P(\text{남} \mid i) = 1/2, & i = 4 \\ P(\text{서} \mid i) = P(\text{북} \mid i) = 1/2, & i = 6 \\ P(\text{동} \mid i) = P(\text{남} \mid i) = 1/2, & i = 11 \\ P(\text{동} \mid i) = P(\text{북} \mid i) = 1/2, & i = 13 \\ P(\text{동} \mid i) = P(\text{서} \mid i) = P(\text{남} \mid i) = P(\text{북} \mid i) = 1/4, & i = 7, 10 \end{cases}$$

- 상태 2에 대한 가치함수의 값, 즉 $v_{\pi_1}(2)$ 를 계산해 보자. 상태 2에서는 서쪽으로 가는 길만 있고 바로 종료 상태에 도달하여 누적 보상 5를 얻으므로 $v_{\pi_1}(2) = 5$ 이다. 상태 3은 서서라는 외길이 있고 누적 보상은 -1+5 이므로 $v_{\pi_1}(3) = 4$ 이다.
- 이제 상태 6에 대해 생각해 보자. 상태 6에서는 0.5 확률로 서 또는 북으로 갈 수 있다. 종료 상태로 이어지는 경로는 서북과 북서 2개이고 둘은 0.5라는 확률로 발생할 것이다. 따라서 $v_{\pi_1}(6)$ 은 다음과 같이 계산할 수 있다.

$$v_{\pi_1}(6) = \frac{1}{2}(-1+5) + \frac{1}{2}(-1+5) = 4$$

3. 정책과 가치함수

- 그리고 상태 11에서는 종료 상태에 도달하는 길이 동남과 남동 두 가지이다. 따라서 $v_{\pi_1}(11)$ 은 다음과 같이 계산하여 4가 된다.

$$v_{\pi_1}(11) = \frac{1}{2}(-1+5) + \frac{1}{2}(-1+5) = 4$$

- 이제 정책 □ π_2 에 대해 생각해 보자. 이 정책의 경우 상태 11에서 종료 상태에 도달하는 길은 북북서서, 북서북서, 북서서북, 서북북서, 서북서북, 서서북북의 여섯 가지가 있으므로 다음과 같이 계산하여 $v_{\pi_2}(11) = 2$ 를 얻는다.

$$\begin{aligned} v_{\pi_2}(11) = & \frac{1}{2}\frac{1}{2}(-1-1-1+5) + \frac{1}{2}\frac{1}{2}\frac{1}{2}(-1-1-1+5) \\ & + \frac{1}{2}\frac{1}{2}\frac{1}{2}(-1-1-1+5) + \frac{1}{2}\frac{1}{2}\frac{1}{2}(-1-1-1+5) \\ & + \frac{1}{2}\frac{1}{2}\frac{1}{2}(-1-1-1+5) + \frac{1}{2}\frac{1}{2}(-1-1-1+5) = 2 \end{aligned}$$

- 따라서 $v_{\pi_1}(11) = 4$ 인 점을 고려하면 상태 11에서 π_1 이 π_2 보다 더 좋다. 다른 상태에 대해서 가치함수를 계산하면 모든 상태에서 π_1 이 π_2 보다 더 좋다는 사실을 확인할 수 있다. [예제 7-1]을 통해 가치함수로 정책을 평가할 수 있음을 확인하였다.

3. 정책과 가치함수

B. 벨만 기대 방정식

- 이제 가치함수를 즉각적인 보상 R_{t+1}과 다음 상태에서의 가치의 할인된 값을 더하는 식으로 표현하고자 한다. 먼저 상태 가치함수에 대하여 생각해 보면, 상태 가치함수는 다음과 같이 표현할 수 있다.

$$\begin{aligned} v_\pi(s) &= E_\pi[G_t \mid S_t = s] = E_\pi[R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \cdots) \mid S_t = s] \\ &= E_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] = E_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \end{aligned} \qquad (7.3.3)$$

- 식 (7.3.3)은 강화학습에서 매우 중요한 역할을 하는 상태 가치함수에 대한 벨만 기대 방정식이다. 벨만 기대 방정식은 현재 상태 s와 가치함수와 다음 상태 S_{t+1}의 가치함수 사이의 관계 방정식이다.

$$q_\pi(s,a) = E_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a] \qquad (7.3.4)$$

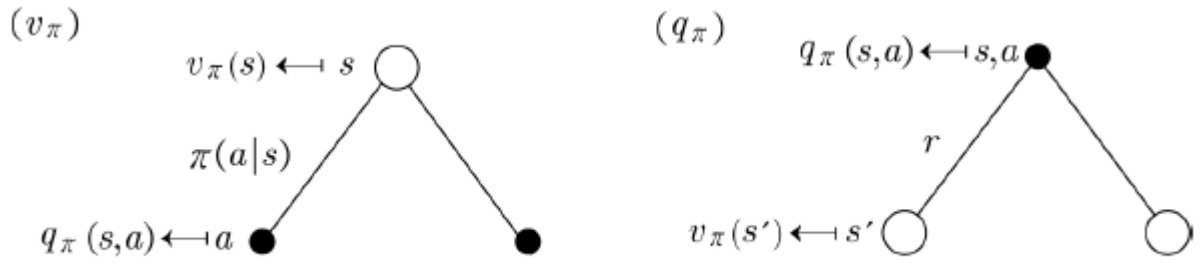


그림 7-7 v_π와 q_π의 백업 다이어그램

3. 정책과 가치함수

- [그림 7-7]에서 흰 원은 상태를 의미하고 검은 원은 행동을 나타낸다. 왼쪽 그림은 특정 상태의 가치는 그 상태에서 가능한 행동, 즉 상태-동작 쌍의 가치와 현재의 정책에서 각 행동을 취할 확률 $\pi(a \mid s)$에 의존한다는 점을 설명한다. [그림 7-7]을 통해 볼 때 상태 가치 함수는 모든 행동에 대해 $q_\pi(s,a)$와 $\pi(a \mid s)$를 곱하여 더한 것이라는 것을 알 수 있다.
- 즉, 상태 가치함수와 행동 가치함수 사이의 관계는 식 (7.3.5)와 같이 나타낼 수 있다.

$$v_\pi(s) = \sum_{a \in A} \pi(a \mid s) q_\pi(s,a) \qquad (7.3.5)$$

- 비슷한 방법으로 행동 가치함수와 상태 가치함수 사이의 관계는 식 (7.3.6)과 같이 표현할 수 있다. 상태 s에 □□ 행동 a를 선택했을 때 그 행동의 가치는 두 가지 요소로 구성된다. 즉, 상태 s에서 행동 a를 선택했을 때의 보상과 그 다음 상태의 가치함수이다. 그런데 다음 상태의 가치함수 $v_\pi(s')$은 $t+1$ 시점에서의 가치함수이므로 할인율을 적용해야 한다.

$$q_\pi(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_\pi(s') \qquad (7.3.6)$$

- 이제 식 (7.3.6)을 식 (7.3.5)에 대입하여 정리하면 상태 가치함수는 다음과 같다.

$$v_\pi(s) = \sum_{a \in A} \pi(a \mid s)[R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_\pi(s')] \qquad (7.3.7)$$

3. 정책과 가치함수

- 비슷한 방법으로 식 (7.3.5)를 식 (7.3.6)에 대입하여 정리하면, 행동 가치함수에 대해 다음 관계식을 얻을 수 있다.

$$q_\pi(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \sum_{a' \in A} \pi(a' \mid s') q_\pi(s',a') \qquad (7.3.8)$$

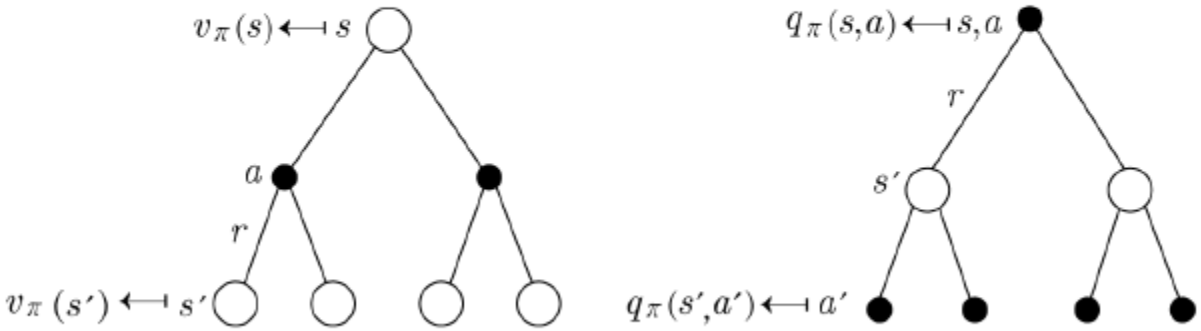


그림 7-8 v_π 및 q_π 관련 연립 방정식의 백업 다이어그램

3. 정책과 가치함수

C. 최적 정책 및 벨만 최적 방정식.

- 최적 상태 가치함수를 알면 최적 정책을 쉽게 결정할 수 있다. 따라서 강화학습에서는 최적 상태 가치함수를 찾는 일이 매우 중요하다. 식 (7.3.9)는 최적 상태 가치함수를 정의한다. 최적 상태 가치함수를 보통 $v_*(s)$ 로 표기하는데, 최적 정책 π_* 에 대해 가치를 계산하므로 $v_{\pi_*}(s)$ 로도 표기한다.

$$v_*(s) = v_{\pi_*}(s) = \max_\pi v_\pi(s), \quad \forall s \in \boldsymbol{S} \tag{7.3.9}$$

- 최적 행동 가치함수도 같은 방법으로 식 (7.3.10)과 같이 정의할 수 있다.

$$q_*(s,a) = q_{\pi_*}(s,a) = \max_\pi q_\pi(s,a), \quad \forall s \in \boldsymbol{S}, \ \forall a \in \boldsymbol{A}(s) \tag{7.3.10}$$

- 가장 큰 행동 가치함수인 최적 행동 가치함수를 에이전트가 찾았다고 가정해 보자. 그러면 최적 정책은 주어진 상태 [s 에서의 최적 행동 가치함수 중 가장 큰 최적 행동 가치함수를 가지는 행동을 택하는 것이다. 따라서 최적 정책은 식 (7.3.11)과 같이 구할 수 있다.

$$\pi_*(s,a) = \begin{cases} 1, & \text{만약} \quad a = \arg\max_{a \in \boldsymbol{A}} q_*(s,a) \\ 0, & \text{그 외} \end{cases} \tag{7.3.11}$$

3. 정책과 가치함수

- 이제 최적의 상태 가치함수 또는 행동 가치함수를 어떻게 구할 수 있는지 알아보자. 이것을 구하는 것은 순차적 행동 결정 문제를 푸는 것이다. 최적의 가치함수를 구하는 구체적인 방법에 대해서는 이후에 동적 계획법과 강화학습에서 설명하겠다. 여기서는 최적의 가치함수들 간의 관계를 생각해 보자.
- 현재 상태의 가치함수가 최적이라는 것은 에이전트가 가장 좋은 행동을 선택한다는 것이다. 에이전트는 행동 가치함수를 기준으로 어떤 행동이 가장 좋은지를 알 수 있다. 즉, 행동 선택의 기준이 되는 행동 가치함수 중 가장 큰 값을 가지는 행동 가치함수를 취하는 것이 최적의 가치함수가 된다. 따라서 식 (7.3.12)와 같이 최적의 행동 가치함수 중 행동 a□ 대해 최대가 되는 것이 최적의 상태 가치함수이다.

$$v_*(s) = \max_{a \in \boldsymbol{A}(s)} q_*(s,a) \qquad (7.3.12)$$

3. 정책과 가치함수

- 식 (7.3.12)에서 행동 가치함수를 상태 가치함수로 바꾸어 나타내면 다음과 같으며, 이를 상태 가치함수에 대한 벨만 최적 방정식이라고 한다.

$$
\begin{aligned}
v_*(s) &= \max_{a \in A(s)} q_*(s,a) \\
&= \max_a E_{\pi_*}[G_t \mid S_t = s, A_t = a] \\
&= \max_a E_{\pi_*}[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a] \\
&= \max_a E_{\pi_*}[R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \mid S_t = s, A_t = a] \\
&= \max_a E[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \qquad (7.3.13) \\
&= \max_{a \in A(s)} \sum_{s'} P^a_{ss'} [R^a_{ss'} + \gamma v_*(s')] \qquad (7.3.14)
\end{aligned}
$$

- 여기서 [$R^a_{ss'} = E(R_{t+1} \mid S_t = s, A_t = a, S_{t+1} = s')$ 이다. 식 (7.3.13)과 식 (7.3.14)는 v_* 에 대한 두 가지 형태의 벨만 최적 방정식이다. 비슷한 방법으로 행동 가치함수에 대해서도 벨만 최적 방정식을 유도할 수 있다.

$$
\begin{aligned}
q_*(s,a) &= E[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \mid S_t = s, A_t = a] \\
&= \sum_{s'} P^a_{ss'} [R^a_{ss'} + \gamma \max_{a'} q_*(s', a')] \qquad (7.3.15)
\end{aligned}
$$

3. 정책과 가치함수

- 식 (7.3.13)과 식 (7.3.15)에 기댓값 기호를 사용한 이유는 괄호 안의 값이 에이전트가 선택하는 것이 아니라 환경이 주는 값이기 때문이다.
- 지금까지 벨만 기대 방정식과 벨만 최적 방정식을 살펴보았다. 다음 장에서 다루게 될 동적 계획법은 이 방정식들을 이용하여 MDP로 정의되는 문제를 계산으로 푸는 방법이다. 한편 벨만 최적 방정식은 최댓값을 구해야 하기 때문에 비선형 방정식이며 정확한 형태의 해법이 존재하지 않는다. 따라서 벨만 최적 방정식을 위한 다양한 풀이 방법이 제안되었다.

4. 동적 계획법

- 순차적 행동 문제를 푸는 방법에는 여러 가지가 있다. 대표적인 방법은 동적 계획법과 강화학습이다. 동적 계획법은 MDP를 계산으로 푸는 방법이고, 강화학습은 MDP를 학습으로 푸는 방법이라고 생각할 수 있다. 그리고 동적 계획법은 초창기 강화학습이 주로 사용하던 방법이기도 하다.
- 동적 계획법을 처음 제안한 사람은 벨만 방정식을 만든 벨만이다. 벨만 방정식은 동적 계획법 방정식이라고도 불리며, 최적화와 관련된 방정식이다. 동적 계획법을 적용하려면 MDP가 주어져야 하며 상태와 행동의 개수가 적어 계산 시간과 메모리 요구량이 현실적이어야 한다. 이 조건은 강한 제약으로서 오늘날 기계학습의 규모에 비추어 보면 비교적 간단한 문제에나 적용할 수 있다. 예를 들어, 장기나 바둑에서 MDP는 주어지는데 상태와 행동의 개수가 방대하여 동적 계획법을 적용하면 계산 시간과 메모리 요구량을 감당할 수 없다.
- 동적 계획법은 복잡한 문제를 다루기 쉬운 하위 문제로 나누어 반복적으로 해결한다. 즉, 각 상태의 가치함수 $v_{\pi_*}(s)$ 를 구할 때 한 번에 구하는 것이 아니라 다음과 같이 작은 문제로 나누어 반복적으로 구한다.

4. 동적 계획법

$$v_{\pi_0}(s) \rightarrow v_{\pi_1}(s) \rightarrow \cdots \rightarrow v_{\pi_k}(s) \rightarrow \cdots \rightarrow v_{\pi_*}(s)$$

- 여기서 화살표는 한 번의 계산을 의미하며 각 반복의 진행을 나타낸다. 동적 계획법은 나중에 설명할 몬테카를로 방법과 시간차 학습과 같은 알고리즘의 토대를 이루는 핵심 개념과 아이디어를 제공한다. 따라서 동적 계획법을 제대로 이해하는 것이 필요하다. MDP로 정의되는 순차적 행동 결정 문제를 푸는 동적 계획법에는 두 가지 방법이 있는데, 정책 반복과 가치 반복이다. 정책 반복은 벨만 기대 방정식을 이용하여 문제를 풀고, 가치 반복은 벨만 최적 방정식을 이용하여 푼다.

4. 동적 계획법

1. 정책반복

- 정책은 모든 상태에서 에이전트가 어떻게 행동할지에 대한 정보이다. MDP로 정의되는 문제에서 궁극적으로 구하고 싶은 것은 가장 높은 보상을 주는 정책을 찾는 것이다. 처음에는 이 정책을 알 수 없기 때문에 어떤 특정 정책부터 시작하여 계속 정책을 개선하는 방법을 사용한다. 즉, 임의의 초기 정책 [π_0으로 시작하고 □ v_{π_0}을 사용하여 개선된 정책 π_1을 찾는다.
- 다시 v_{π_1}을 사용하여 개선된 정책 [π_2를 찾는다. 이러한 과정을 반복하면 정책은 최적 정책으로 수렴한다. 따라서 정책 반복 과정을 다음과 같이 나타낼 수 있다.

$$\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \cdots \xrightarrow{I} \pi_* \xrightarrow{E} v_{\pi_*}$$

- 여기서 $\xrightarrow{E}$ 는 정책 평가를 나타내고 $\xrightarrow{I}$ 는 정책 개선을 나타낸다.

4. 동적 계획법

A. 정책 평가

- 정책이 얼마나 좋은지 판단하기 위해 사용하는 도구가 상태 가치함수이다. 이것은 현재의 정책 π 를 따를 때 받을 보상에 대한 기댓값이다. 이 기댓값은 다음과 같이 합의 형태로 표현된다는 것을 식 (7.3.7)에서 확인하였다.

$$v_\pi(s) = \sum_{a \in A} \pi(a \mid s)[R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_\pi(s')], \quad \forall s \in S$$

- 정책 평가는 벨만 기대 방정식을 사용하여 정책 Iπ 의 상태 가치함수를 계산하는 것이다. 즉, v_π 를 추정하는 것이다. 정책 평가는 실제로 정책 π 에 대해 반복적으로 수행하므로 상태 가치함수를 계산하는 방정식은 다음과 같다. 즉, 다음과 같이 k□□ 상태 가치함수를 사용하여□ $k+1$ 번째 상태 가치함수를 계산한다.

$$v_{k+1}(s) = \sum_{a \in A} \pi(a \mid s)[R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_k(s')] \qquad (7.4.1)$$

4. 동적 계획법

B. 정책 개선

- 정책에 대한 평가를 마쳤다면 정책을 개선해야 한다. 가장 많이 사용되는 정책 개선 방법은 탐욕 정책 개선이다. 정책 평가를 통해 구한 것은 에이전트가 정책 π를 따랐을 때의 모든 상태에 대한 상태 가치함수의 값이다. 이것을 사용하여 각 상태에 대해 어떤 행동이 좋은지를 알아보자. 한편 행동 가치함수를 사용하면 어떤 행동이 좋은지 알 수 있다. 행동 가치함수는 다음과 같이 상태 가치함수를 사용하여 표현된다는 것을 식 (7.3.6)에서 확인하였다.

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

- 탐욕 정책 개선은 상태 s에서 선택 가능한 모든 행동에 대해 $q_\pi(s,a)$를 비교하고, 그중 가장 큰 행동 가치함수를 가지는 행동을 선택하는 것과 관련되어 있다. 즉, 탐욕 정책 개선은 다음 관계식

$$\pi'(s) = \arg\max_{a \in A} q_\pi(s,a) \qquad (7.4.2)$$

를 만족하는 $\pi'(s)$를 구했을 때 모든 $s \in S$에 대해 부등식 $q_\pi(s,\pi'(s)) \geq v_\pi(s)$가 성립하면 π'이 π보다 더 나은 정책이라고 간주한다. 따라서 탐욕 정책 개선은 모든 $s \in S$에 대해 관계식 $v_{\pi'}(s) \geq v_\pi(s)$를 만족하는 정책 π'을 찾아내는 것이다.

4. 동적 계획법

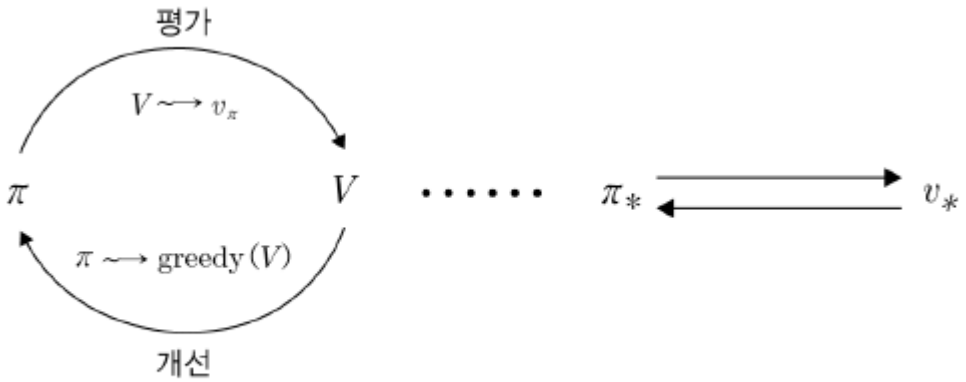


그림 7-9 동적 계획법의 GPI 다이어그램

- 앞에서 살펴본 바와 같이 정책 반복은 정책 평가와 정책 개선을 무한히 반복하면서 최적정책을 찾는 방법이다. 이러한 과정을 일반화 정책 반복으로 간주할수 있다. GPI는 정책 평가와 정책 개선이 상호작용하도록 하는 일반적인 개념을 의미한다. 거의 모든 강화학습 방법은 GPI로 잘 설명된다. [그림 7-9]는 동적 계획법의 GPI를 설명하며 [알고리즘 7-1]은 동적 계획법의 정책 반복 알고리즘을 설명한다. 대부분 정책 반복은 적은 횟수의 반복으로도 수렴한다.

4. 동적 계획법

알고리즘 7-1 동적 계획법의 정책 반복 알고리즘

1. 초기화

   모든 $s \in \mathcal{S}$에 대해 $v_0(s) \in R$ 및 $\pi_0(s) \in \mathcal{A}(s)$를 초기화한다. $\pi_0(s)$는 결정적 정책이다.

2. 정책 평가

$$v_{\pi_{k+1}}(s) = \sum_{s' \in \mathcal{S}} P_{ss'}^{\pi_k(s)} \left[R_{ss'}^{\pi_k(s)} + \gamma v_{\pi_k}(s') \right]$$

3. 정책 개선

$$\pi_{k+1}(s) = \arg\max_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{S}} P_{ss'}^{a} \left[R_{ss'}^{a} + \gamma v_{\pi_k}(s') \right]$$

4. 모든 $s \in \mathcal{S}$에 대해 $\pi_{k+1}(s) = \pi_k(s)$가 성립할 때까지 정책 평가와 정책 개선을 반복한다.

2. 가치 반복

- 정책 반복은 반복할 때마다 정책 평가가 필요하다는 단점이 있다. 그리고 정책 평가는 반복적인 계산이 필요하다. 만약 정책 평가가 반복적으로 수행된다면 정책은 무한히 반복한 후에 최적 정책 □ π_* 로 수렴하게 된다. 그러나 가치 반복은 벨만 최적 방정식을 이용하며 평가를 한 번만 진행한다.
- 즉, 가치 반복은 벨만 최적 방정식 (7.3.14)를 사용하여 다음과 같이정책 개선 단계와 절단된 정책 평가 단계를 결합한 간단한 연산으로 표현할 수 있다.

$$v_{k+1}(s) = \max_{a \in \boldsymbol{A}(s)} \sum_{s' \in \boldsymbol{S}} P^a_{ss'} \left[R^a_{ss'} + \gamma v_k(s') \right], \quad \forall s \in \boldsymbol{S} \qquad (7.4.3)$$

- 임의의 ⊏ v_0 에 대해 ⌈ v_* 의 존재성을 보장해 주는 조건에서 수열 $\{v_k\}$⌉ 가 v_*⌉ □ 수렴하는 것을 증명할 수 있다.

4. 동적 계획법

- 가치 반복을 이해하는 또 다른 방법은 정책 반복은 벨만 기대 방정식을 사용하지만 가치반복은 벨만 최적 방정식을 사용한다는 것이다. 벨만 최적 방정식은 최적 가치 함수들 사이의 관계이다. 단순히 이 관계식을 반복적으로 변환해 주면 된다.
- 정책 반복의 경우 정책을평가할 때 수많은 계산이 필요하다는 단점이 있는데, 가치 반복은 그 평가를 단 한 번만 한다. 따라서 현재 가치 함수를 계산하고 갱신할 때 max를 취함으로써 탐욕적으로 개선하는효과를 주므로 가치 반복은 한 번의 평가와 한 번의 개선을 수행한다.
- 이제 가치 반복이 어떻게 종료되는지 생각해 보자. 가치 반복은 $\overline{\pi}_*$ 에 정확하게 수렴하기위해 무한한 반복이 필요하다. [알고리즘 7-2]는 동적 계획법의 가치 반복 알고리즘을 설명한다.

알고리즘 7-2 동적 계획법의 가치 반복 알고리즘

1. 배열 v_0을 임의로 초기화한다. 예를 들어, 모든 $s \in \mathcal{S}$에 대해 $v_0(s) = 0$으로 초기화한다.
2. $\max_{s \in \mathcal{S}} | v_{k+1}(s) - v_k(s) | > \epsilon$이 성립하는 한 다음 과정을 반복하여 $v_{k+1}(s)$를 구한다.

$$v_{k+1}(s) = \max_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{S}} P^a_{ss'} \left[R^a_{ss'} + \gamma v_k(s') \right], \quad \forall s \in \mathcal{S}$$

3. 다음 조건을 만족하는 결정적 정책 π_*를 출력한다.

$$\pi_*(s) = \arg\max_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{S}} P^a_{ss'} \left[R^a_{ss'} + \gamma v_k(s') \right]$$

4. 동적 계획법

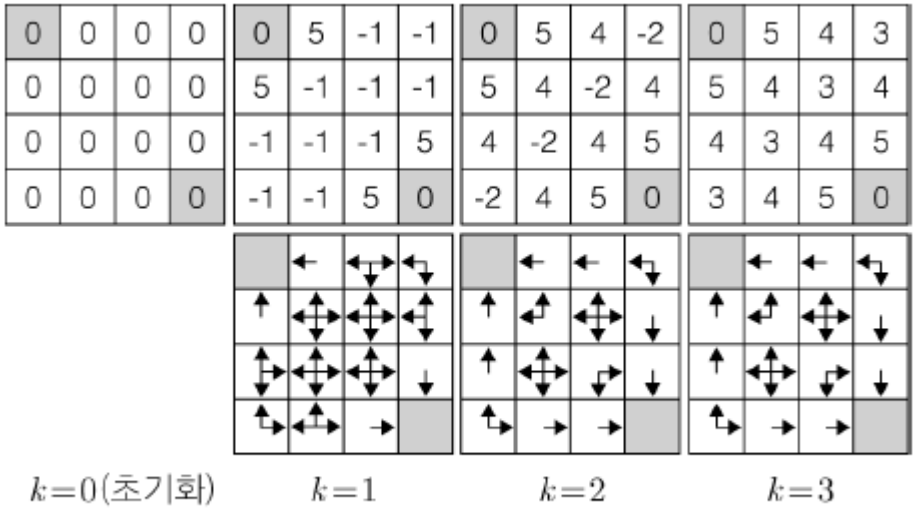


그림 7-10 가치 반복 예제

예제 7-2 **최적 가치함수로 최적 정책 구하기**

[예제 7-1]에서 다룬 4×4 격자세계의 예제를 다시 생각해 보자. [그림 7-10]은 가치 반복 알고리즘의 각 반복 단계에 대한 상태 가치함수와 정책을 보여 준다. $t=0$은 초기화 단계이다. 모든 상태에 대해 $v_0(s)=0$으로 초기화한다. 각 반복 단계에 대한 상태 가치함수와 정책을 구해 보자. 먼저 첫 번째 반복 단계에서 상태 2와 상태 6에 대해 가치 반복 알고리즘을 적용해 보자.

$$
\begin{aligned}
v_1(2) &= \max_{a\in\{\text{동},\text{서},\text{남},\text{북}\}} \sum_{s'\in S} P^a_{2s'}\left[R^a_{2s'}+\gamma v_0(s')\right] \\
&= \max\left\{P^{\text{동}}_{23}[-1+v_0(3)], P^{\text{서}}_{21}[5+v_0(1)], P^{\text{남}}_{26}[-1+v_0(6)], P^{\text{북}}_{22}[-1+v_0(2)]\right\} \\
&= \max\{1\times(-1+0), 1\times(5+0), 1\times(-1+0), 1\times(-2+0)\}=5 \\
v_1(6) &= \max_{a\in\{\text{동},\text{서},\text{남},\text{북}\}} \sum_{s'\in S} P^a_{6s'}\left[R^a_{6s'}+\gamma v_0(s')\right] \\
&= \max\left\{P^{\text{동}}_{67}[-1+v_0(7)], P^{\text{서}}_{65}[5+v_0(5)], P^{\text{남}}_{610}[-1+v_0(10)], P^{\text{북}}_{62}[-1+v_0(2)]\right\} \\
&= \max\{1\times(-1+0), 1\times(-1+0), 1\times(-1+0), 1\times(-1+0)\}=-1
\end{aligned}
$$

즉, $v_1(2)=5$와 $v_1(6)=-1$을 얻는다. 따라서 $\boldsymbol{A}(2)=\{서\}$와 $\boldsymbol{A}(6)=\{동,서,남,북\}$이 성립한다. 같은 방법으로 다른 상태에 대해서도 계산하면 [그림 7-8]의 $k=1$의 경우가 된다.

이제 두 번째 반복 단계에서 상태 2와 상태 6에 대해 가치 반복 알고리즘을 적용해 보자.

$$
\begin{aligned}
v_2(2) &= \max_{a\in\{동,서,남,북\}} \sum_{s'\in S} P^a_{2s'}\left[R^a_{2s'}+\gamma v_1(s')\right] \\
&= \max\left\{P^{동}_{23}[-1+v_1(3)], P^{서}_{21}[5+v_1(1)], P^{남}_{26}[-1+v_1(6)], P^{북}_{22}[-1+v_1(2)]\right\} \\
&= \max\{1\times(-1-1), 1\times(5+0), 1\times(-1-1), 1\times(-2+5)\}=5
\end{aligned}
$$

$$
\begin{aligned}
v_2(6) &= \max_{a\in\{동,서,남,북\}} \sum_{s'\in S} P^a_{6s'}\left[R^a_{6s'}+\gamma v_1(s')\right] \\
&= \max\left\{P^{동}_{67}[-1+v_1(7)], P^{서}_{65}[5+v_1(5)], P^{남}_{610}[-1+v_1(10)], P^{북}_{62}[-1+v_1(2)]\right\} \\
&= \max\{1\times(-1-1), 1\times(-1+5), 1\times(-1-1), 1\times(-1+5)\}=4
\end{aligned}
$$

즉, $v_2(2)=5$와 $v_2(6)=4$를 얻는다. 따라서 $\boldsymbol{A}(2)=\{서\}$와 $\boldsymbol{A}(6)=\{서,북\}$이 성립한다. 같은 방법으로 다른 상태에 대해서도 계산하면 [그림 7-10]의 $k=2$의 경우가 된다.

즉, $v_2(2)=5$와 $v_2(6)=4$를 얻는다. 따라서 $\boldsymbol{A}(2)=\{서\}$와 $\boldsymbol{A}(6)=\{서,북\}$이 성립한다. 같은 방법으로 다른 상태에 대해서도 계산하면 [그림 7-10]의 $k=2$의 경우가 된다.

4. 동적 계획법

세 번째 반복 결과는 [그림 7-10]의 $k=3$의 경우이다. $k=2$일 때의 결과와 비교할 때 변화가 거의 없으므로 수렴했다고 판단할 수 있다. 따라서 $k=2$일 때의 최적 가치함수를 사용하여 최적 정책을 구할 수 있다. 예를 들어, 최적 정책은 $\boldsymbol{A}(4)=\{서, 남\}$에 대해 같은 확률 $P(서 \mid 4)=P(남 \mid 4)=1/2$을 부여한다.

5. 몬테카를로 방법

- 앞 절의 동적 계획법은 환경 모형인 MDP에 대한 사전 지식을 알고 있어야만 동작한다. 다시 말해, 7.4절의 모든 알고리즘은 가치함수를 최적화하고 최적 정책을 찾기 위하여 상태전이확률과 보상함수가 필요하다. 이 절에서는 환경에 대한 사전 지식이 없는 상태에서, 즉 MDP를 모르는 상태에서 환경과 직접 상호작용하면서 가치함수를 최적화하고 최적 정책을 찾아가는 학습기반의 몬테카를로[MC] 방법에 대해 살펴본다. MC 방법은 환경과의 상호작용을 통해 얻은 상태, 행동 및 보상의 표본 에피소드를 훈련 데이터로 활용하여 최적 정책을 찾아낼 수 있다. 예를 들어, 바둑에서는 이전 기보를 훈련 데이터로 활용할 수도 있고 프로그램과 프로그램을 대결하게 하여 훈련 데이터를 만들 수도 있다. [그림 7-11]은 동적 계획법과 MC 방법의 동작을 비교한 것이다.

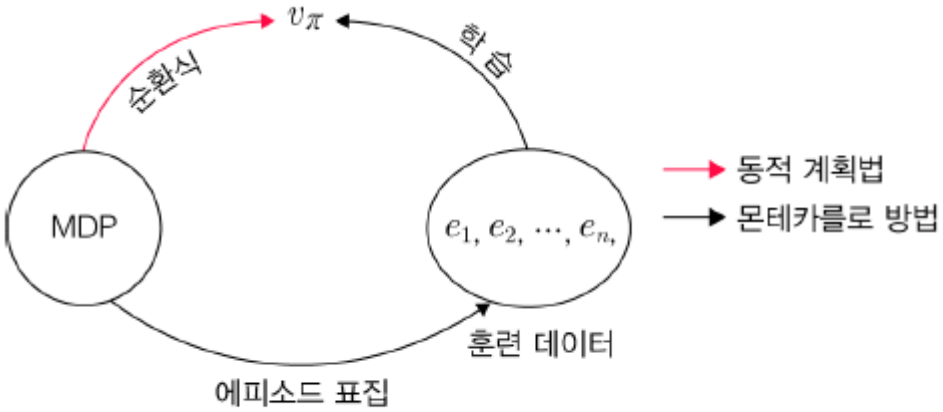


그림 7-11 동적 계획법과 몬테카를로 방법의 동작 비교

5. 몬테카를로 방법

1. MC 예측

- 주어진 정책에 대해 상태 가치함수를 학습하기 위한 MC 방법을 살펴보고자 한다. MC 방법을 사용하여 참 상태 가치함수를 추정할 때 에이전트가 환경에서 한 번 에피소드를 진행하는 것이 표집이다 적절한 방법을 사용하여 에피소드를 표집했다고 가정하자. 격자세계 예제에서는 이동 경로를 에피소드로 간주할 수 있고 바둑에서는 게임 시작부터 종료까지의 기록, 즉 기보 하나를 에피소드로 간주할 수 있다.
- 표집을 통해 얻은 표본들의 산술평균으로 참 상태 가치함수를 추정할 때 몬테카를로 근사를 사용하는 것을 MC 예측이라고 한다. 다시 상태 가치함수의 정의를 생각해 보자. 먼저 확률변수인 반환값 G_t 다시 보자.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots$$

- 반환값 G_t 는 미래 시점에서 받을 보상을 현재 가치로 할인하여 더한 장기간의 보상이다. 에피소드는 무한 시간 지속될 가능성이 있지만 여기서는 에피소드가 끝이 있다고 가정하겠다. 끝이 없는 에피소드에서는 반환값을 알기 어렵다.

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-t-1} R_T$$

5. 몬테카를로 방법

- MC 예측에서는 환경 모형을 알아야만 계산할 수 있는 기댓값 E_π 를 계산하는 것이 아니라, 가능한 많은 에피소드를 표집하여 구한 반환값의 산술평균을 통해 다음과 같이 $v_\pi(s)$ 를 추정한다.

$$v_\pi(s) = E_\pi[G_t \mid S_t = s] \approx \frac{1}{n} \sum_{i:\, S_t^i = s}^{n} G_t^i \qquad (7.5.1)$$

- 여기서 G_t^i는 t □□□□ 상태 s □ 방문한 에피소드 i 의 반환값을 나타낸다. 대수의 법칙에 의하면 $n \to \infty$ 일 때 정확한 기댓값을 구할 수 있다. 문제는 표본 반환값을 어떻게 얻는가 하는 것이다. 이를 위해서는 많은 에피소드를 고려해야 한다. 우리는 각 에피소드에 대해 상태와 보상의 수열을 얻는데, 이러한 에피소드에 대한 보상으로부터 정의에 따라 반환값을 계산할 수 있다. 즉, 어떤 상태의 가치를 추정하기 위해 그 상태에 방문하여 관측한 반환값들을 평균한다. 반환값을 평균하는 방법에는 첫-방문 MC 방법과 모든-방문 MC 방법 두 가지가 있다. 두 방법 모두 참 가치함수에 근사적으로 수렴한다.

5. 몬테카를로 방법

A. 첫-방문 MC 방법

- 같은 에피소드에서 어떤 상태 sl 口 여러 번 방문할 수 있다. 이름에서 짐작할 수 있듯이, 첫-방문 MC 예측 방법은 상태 s 에 처음 방문했을 때부터 그 이후의 반환값을 평균하여 상태 가치함수 $v_\pi(s)$ 를 추정한다. 이 방법은 처음 방문한 상태만 인정한다. 블랙잭 게임은 첫-방문 MC 방법을 적용할 수 있는 좋은 예제이다. 왜냐하면 같은 에피소드에서 같은 상태로 다시 돌아올 수 없기 때문이다. 자세한 내용을 위해 [Sutton과 Barto2018]을 참고하라. 첫-방문 MC 예측 알고리즘은 다음과 같다.

알고리즘 7-3 첫-방문 MC 예측 알고리즘

1. 정책과 상태 가치함수를 초기화한다.
2. 현재 정책에 따라 한 개의 에피소드를 생성하여 시작한다.
   (a) 해당 에피소드에서 방문한 상태를 추적한다.
3. 단계 2의 (a)에서 어떤 상태 하나를 선택한다.
   (a) 이 상태에 처음 방문한 후에 얻게 되는 반환값을 목록에 추가한다.
   (b) 모든 반환값을 평균한다.
   (c) 계산된 평균을 이 상태의 가치로 설정한다.
4. 단계 3을 반복한다.
5. 만족할 때까지 단계 2~4를 반복한다.

5. 몬테카를로 방법

B. 모든-방문 MC 방법

- 모든-방문 MC 예측 방법은 상태 s 에 대한 모든 방문에 대해 그 이후의 반환값을 평균하여 상태 가치함수 $v_\pi(s)$ 를 추정한다. 모든-방문 MC 예측 알고리즘은 [알고리즘 7-3]의 단계 3-(a)를 '이 상태에 대한 모든 방문에 대해 그 이후에 얻게 되는 반환값을 목록에 추가한다'로 변경하면 된다.
- 두 가지 방법의 개념을 이해하기 위해 간단한 예제를 생각해 보자. 2개의 상태 a□ Ib그 있는 환경을 가정하고 다음과 같은 2개의 표본 에피소드를 관측했다고 하자. 에피소드에서 괄호 안의 숫자는 보상을 나타낸다.

$$e_1 = (a,3) \rightarrow (a,2) \rightarrow (b,-4) \rightarrow (a,4) \rightarrow (b,-3) \rightarrow \text{종료}$$

$$e_2 = (b,-2) \rightarrow (a,3) \rightarrow (b,-3) \rightarrow \text{종료}$$

- 두 가지 방법을 사용하여 상태 가치함수를 구하면 다음과 같다.

첫-방문 MC	모든-방문 MC
$\hat{v}_\pi(a) = (2+0)/2 = 1$	$\hat{v}_\pi(a) = (2-1+1+0)/4 = 1/2$
$\hat{v}_\pi(b) = (-3-2)/2 = -5/2$	$\hat{v}_\pi(b) = (-3-3-2-3)/4 = -11/2$

5. 몬테카를로 방법

C. 중분 평균 방법

- 평균을 구하는 식 (7.5.1)을 좀 더 발전시켜 보면 다음과 같다. 학습 기반의 방법은 여러개의 에피소드를 모아 놓고 한 번에 평균을 계산하는 것이 아니라 하나씩 더하며 계산하기 때문에, 다음과 같이 증분 평균을 사용하여 평균을 계산할 수 있다. 편의상 식(7.5.1)의 합기호에서 상태에 대한 표현을 생략하겠다. n 개의 반환값을 평균하여 구한 상태 가치함수를 V_n으로 표기하자. 그러면 V_n은 다음과 같이 표현할 수 있다.

$$V_n = \frac{1}{n}\sum_{i=1}^{n} G_t^i = \frac{1}{n}\left(G_t^n + \sum_{i=1}^{n-1} G_t^i\right) = \frac{1}{n}\left(G_t^n + (n-1)V_{n-1}\right)$$
$$= V_{n-1} + \frac{1}{n}\left(G_t^n - V_{n-1}\right) \qquad (7.5.2)$$

- 식 (7.5.2)는 MC 예측을 수행하는 동안 계속 새로운 반환값이 들어오면 평균을 어떻게 구하는지를 보여 준다. 어떤 상태의 가치함수는 표집을 통해 에이전트가 그 상태를 방문할 때마다 갱신한다.
- 즉, 전 단계의 상태 가치함숫값 $V(s)$ 에 $\frac{1}{n}(G(s) - V(s))$ 를 더하여 갱신한다. 이때 $\frac{1}{n}$ 은 단계 크기로, 갱신할 때 오차를 얼마나 반영하여 갱신할지 결정하는 것이 라고 생각할 수 있다. 일반적으로 단계 크기를 α 로 고정하고 평균을 갱신한다. 따라서 MC예측의 상태 가치함수 갱신을 일반적인 형태로 나타내면 식 (7.5.3)과 같다.

5. 몬테카를로 방법

$$V(s) \leftarrow V(s) + \alpha(G(s) - V(s)) \qquad (7.5.3)$$

- 일반적으로 식 (7.5.3)은 첫-방문 MC 방법 또는 모든-방문 MC 방법과 함께 사용하여 상태가치함수를 추정한다.

2. 행동 가치함수의 MC 추정

- MDP 모형을 사용할 수 있을 때는 $P_{ss'}^{a}$와 $R_{ss'}^{a}$를 알기 때문에 동적 계획법에서처럼 초기정책 π 에 대해 $v_{\pi}(s)$ 를 추정한 후 정책 평가와 정책 개선을 반복하여 최적 정책을 찾을 수 있다.
- 그러나 MDP 모형을 모를 때는 상태 가치함수의 값만으로는 충분하지 않기 때문에 행동 가치함수의 값, 즉 상태-동작 쌍의 가치함숫값을 추정해야 한다. 따라서 MC 방법의 주요 목표 중 하나는 q_* 를 추정하는 것이다. 이를 위해 먼저 행동 가치함수의 값에 대한 정책평가 문제를 생각한다.
- 행동 가치함수의 값에 대한 정책 평가 문제는 $q_{\pi}(s,a)$ 를 추정하는 것이다. 이때 $q_{\pi}(s,a)$ 는 상태 s 에서 시작하여 행동 a 를 취하며 이후에는 정책 π 를 따를 때의 기대 반환값이다. 여기서의 MC 방법은 상태 가치함수에 대한 정책 평가 문제, 즉 MC 예측 문제와 본질적으로 같다. 어떤 에피소드에서 상태-행동 쌍 (s,a) 를 방문한다는 것은 에피소드에서 상태 s 를 방문하고 행동 a 를 취한다는 것을 의미한다. 모든-방문 MC 방법은 상태-행동 쌍에 대한 모든 방문 후에 얻는 반환값의 평균으로 상태-행동 쌍의 가치함숫값을 추정한다. 첫-방문 MC 방법은 각 에피소드에서 그 상태를 처음 방문하고 그 행동을 취한 후에 얻는 반환값을 평균한다. 이 방법들은 각 상태-행동 쌍에 대한 방문 횟수가 무한대에 접근함에 따라 참 기댓값에 점차 수렴한다.

5. 몬테카를로 방법

- 유일한 어려움은 관련된 많은 상태-행동 쌍이 전혀 방문되지 않을 수 있다는 것이다. 만약 π 가 결정적 정책이면 π 를 따를 때 각 상태에서 취한 행동 중 하나에 대해서만 반환값을 관측하게 된다. 만약 평균을 내는 데 사용할 반환값이 없다면 다른 행동에 대한 MC 추정값은 개선되지 않을 것이다. 이것은 심각한 문제이다. 왜냐하면 행동 가치함수의 값을 학습하는 목적은 각 상태에서 취할 수 있는 행동 중 하나를 선택하는 데 도움을 주는 것이기 때문이다. 대안을 비교하기 위해 현재 우리가 선호하는 행동뿐 아니라, 각 상태의 모든 행동에 대한 가치함수의 값을 추정할 필요가 있다.
- 이것은 탐색을 유지하는 문제이다. 정책 평가가 행동 가치함수의 값에 대해 작동하기 위해서는 지속적인 탐색이 보장되어야 한다. 이를 위한 한 가지 방법은 에피소드가 하나의 상태-행동 쌍에서 시작하며 모든 쌍이 시작 쌍으로 선택될 확률이 0이 아니라는 것을 명확하게 하는 것이다. 이것은 무한개의 에피소드가 상태-행동 쌍을 방문한다는 의미에서 모든 상태-행동 쌍이 무한 번 방문된다는 것을 보장한다. 우리는 이것을 시작 탐색의 가정이라고 부른다. 시작 탐색은 종종 유용하다. 다음 절에서 시작 탐색에 대해 논의하겠다.

5. 몬테카를로 방법

3. MC 제어

- 이제 MC 추정을 어떻게 제어에 이용할 수 있는지에 대해 생각해 보자. 여기서 제어란 최적 정책을 근사하는 것을 의미한다. 전반적인 개념은 동적 계획법에서처럼 GPI의 원리에 따라 진행하는 것이다.
- 동적 계획법에서 정책 평가와 정책 개선을 반복하는 정책 반복이있듯이 MC 제어에서는 MC 정책 평가와 MC 정책 개선을 반복하는 MC 정책 반복이 있다. 이러한 과정을 최적 정책을 얻을 때까지 반복한다. MC 정책 개선은 행동 가치함수를 사용하여 탐욕적으로 진행된다.

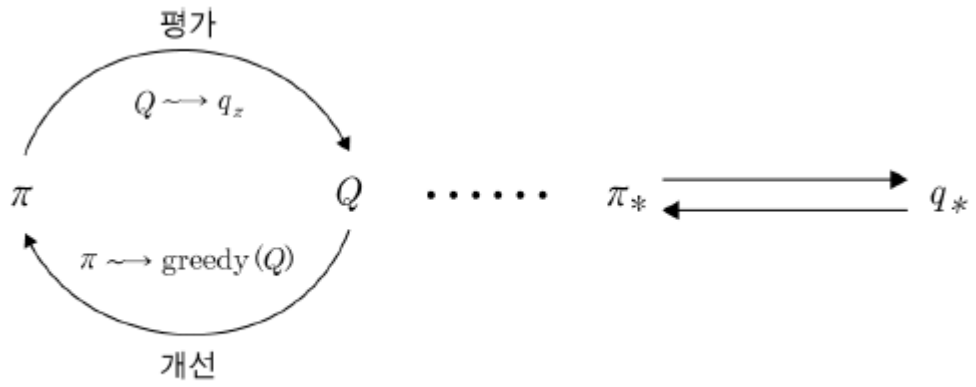


그림 7-12 MC 방법의 GPI 다이어그램

5. 몬테카를로 방법

- MC 정책 반복은 임의의 초기 정책 π_0으로 시작하고 I q_{π_0} 을 사용하여 개선된 정책 π_1 을 찾는다. 다시 q_{π_1} 을 사용하여 개선된 정책 I π_2 를 찾는다. MC 방법은 임의의 I π_k 에 대해 각 q_{π_k} 를 정확하게 계산한다. 이러한 과정을 무한히 반복하면 정책은 최적 정책으로 수렴한다. 따라서 정책 반복 과정을 다음과 같이 나타낼 수 있다.

$$\pi_0 \xrightarrow{E} q_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} q_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \cdots \xrightarrow{I} \pi_* \xrightarrow{E} q_{\pi_*}$$

- 여기서 $\xrightarrow{E}$ 는 정책 평가를 나타내고 $\xrightarrow{I}$는 정책 개선을 나타낸다. 현재는 MC 제어를 위해 두 가지를 가정한다. 첫째, 우리는 무한개의 에피소드를 관측한다는 것이고 둘째, 에피소드는 시작 탐색과 함께 생성된다는 것이다. 이러한 가정에서 MC 방법은 임의의 I π_k 에 대해 각 q_{π_k} 를 정확하게 계산한다.
- 정책 개선은 행동 가치함수를 사용하여 정책을 탐욕적으로 만들어 진행된다. 이 경우 탐욕 정책을 얻기 위해 어떤 모형도 사용하지 않는다. 임의의 행동 가치함수 qI 口 대해 대응되는 탐욕 정책은 식 (7.5.4)처럼 각 상태 s 에 대해 행동 가치함수의 최댓값을 가지는 행동을 선택하는 정책이다.

$$\pi(s) = \arg\max_a q(s,a) \qquad (7.5.4)$$

5. 몬테카를로 방법

- 정책 개선은 q_{π_k} 에 대한 탐욕 정책으로, 각 π_{k+1} 을 구축하여 이루어질 수 있다. 따라서 π_k 와 π_{k+1}| 대해 정책 개선이 이루어진다. 즉, 다음 사실이 성립한다.

$$
\begin{aligned}
q_{\pi_k}(s, \pi_{k+1}(s)) &= q_{\pi_k}(s, \arg\max_a q_{\pi_k}(s,a)) \\
&= \max_a q_{\pi_k}(s,a) \\
&\geq q_{\pi_k}(s, \pi_k(s)) \\
&\geq v_{\pi_k}(s)
\end{aligned}
\qquad (7.5.5)
$$

- 식 (7.5.5)는 π_{k+1}이 π_k 만큼 좋거나 π_k 보다 더 좋다는 것을 보증한다. 이는 결국 전체 과정이 최적 정책 및 최적 가치함수로 수렴한다는 것을 보증한다. 이러한 방식으로 MC 방법은 표본 에피소드만 주어지고 환경에 대한 다른 지식이 없을 때 최적 정책을 찾을 수 있다. MC 방법의 수렴을 보증하기 위해 두 가지 가정이 필요하다.
- 첫째, 에피소드는 시작 탐색을 한다는 것이고 둘째, 가정은 정책 평가는 무한개의 에피소드로 이루어진다는 것이다. 실질적인 알고리즘을 얻기 위해 우리는 두 가지 가정을 모두 제거해야 한다. 첫 번째 가정을 해결하려면 주어진 수준의 성능에 도달할 때까지만 갱신을 하고 에피소드마다 평가와 개선을 번갈아가며 수행해야 한다.

5. 몬테카를로 방법

A. 시작 탐색 MC

- MC 정책 평가를 위해서는 에피소드별로 평가와 개선을 번갈아 수행하는 것이 좋다. 각 에피소드가 끝나면 관측된 반환값을 사용하여 정책 평가를 수행한다. 그다음 에피소드에서 방문한 모든 상태에 대해 정책을 개선한다. 이러한 과정을 따르는 알고리즘을 시작 탐색 MC라고 하며, 다음과 같이 설명할 수 있다.
- 시작 탐색 MC에서는 반환값이 관측된 시점의 정책이 무엇인지에 상관없이 각 상태-행동쌍에 대한 모든 반환값이 누적되고 평균이 구해진다. 시작 탐색 MC는 어떤 준최적에도 수렴할 수 없음을 쉽게 알 수 있다. 만약 그렇다면 가치함수는 결국 해당 정책의 가치함수로 수렴하게 되고 정책이 변경된다. 안정성은 정책과 가치함수가 모두 최적일 때에만 성취된다.
- 이러한 최적값으로의 수렴은 시간이 지남에 따라 행동 가치함수의 변화가 감소하기때문에 불가피해 보이지만 아직 증명되지는 않았다. 이것이 강화학습에서 가장 근본적인 이론 질문 중 하나이다.

5. 몬테카를로 방법

알고리즘 7-4 $\pi \approx \pi_*$**를 추정하기 위한 시작 탐색**^{ES} **MC**

1. 모든 $s \in \mathcal{S}$와 $a \in \mathcal{A}(s)$에 대해 $Q(s,a)$, $\pi(s)$, $Returns(s,a)$를 초기화한다.
   $Q(s,a)$: 임의
   $\pi(s)$: 임의
   $Returns(s,a)$: 빈 리스트

2. 계속 반복
   (a) 0보다 큰 확률을 가진 모든 상태-행동 쌍 $S_0 \in \mathcal{S}$와 $A_0 \in \mathcal{A}(S_0)$을 선택한다.
   (b) π를 따르며 S_0, A_0으로부터 시작하는 에피소드를 생성한다.
   (c) 에피소드에 나타나는 각 쌍 s,a에 대해
   $G \leftarrow s,a$의 첫 발생에 따른 반환값
   G를 $Returns(s,a)$에 추가한다.
   $Q(s,a) \leftarrow \text{average}(Returns(s,a))$
   (d) 에피소드에 있는 각 s에 대해
   $\pi(s) \leftarrow \arg\max_a Q(s,a)$

5. 몬테카를로 방법

B. 시작 비탐색 MC

- 이제 우리는 시작 탐색의 가정을 피하고자 한다. 이를 위해 에이전트가 모든 행동을 선택하도록 보장하는 두 가지 방법, 즉 온폴리시방법과 옵폴리시방법을 소개한다. 온폴리시 방법은 결정을 내리는 데 사용되는 정책을 평가하거나 개선하는 반면, 옵폴리시 방법은 데이터를 생성하는 데 사용된 정책과는 다른 정책을 평가하거나 개선한다. 앞에서 설명한 시작 탐색 MC 방법은 온폴리시 방법의 한 예이다. 여기서는 온폴리시 MC 제어 방법에 대해서만 간단하게 살펴보고자 한다.
- 온폴리시 MC 제어 방법에서 정책은 일반적으로 모든 $s \in S$와 $a \in A(s)$에 대해 $\pi(a \mid s) > 0$ 이 성립한다는 의미에서 소프트하다고 가정한다. 온폴리시 방법은 $\epsilon-$ 탐욕 정책을 사용한다. 이때 $\epsilon-$ 탐욕 정책은 대부분 추정된 행동 가치함수의 값이 최대인 행동을 선택하지만, 확률 ϵ으로 무작위로 행동을 선택한다. 즉, 모든 비탐욕 행동에는 최소 선택확률 $\epsilon / \mid A(s) \mid$ 이 주어지며, 탐욕 행동에는 나머지 확률의 대부분인 $1-\epsilon+\epsilon / \mid A(s) \mid$ 주어진다. 탐욕 정책은 $\epsilon-$ 소프트 정책의 예이다. 이때 적당한 어떤 ϵ 에 대해 $\epsilon-$소프트 정책은 모든 상태와 행동에 대해 $\pi(a \mid s) \geq \epsilon / \mid A(s) \mid$ 이 성립하는 정책이다. 알고리즘은 다음과같다.

5. 몬테 카를로 방법

알고리즘 7-5 $\pi \approx \pi_*$를 추정하기 위한 온폴리시 첫-방문 MC 제어

1. 모든 $s \in \mathcal{S}$와 $a \in \mathcal{A}(s)$에 대해 $Q(s,a)$, $\pi(s)$, $Returns(s,a)$를 초기화한다.
   $Q(s,a)$: 임의
   $Returns(s,a)$: 빈 리스트
   $\pi(a \mid s)$: 임의의 ϵ-소프트 정책

2. 계속 반복
   (a) π를 사용하여 한 개의 에피소드를 생성한다.
   (b) 에피소드에 나타나는 각 쌍 s,a에 대해
      $G \leftarrow s,a$의 첫 발생에 따른 반환값
      G를 $Returns(s,a)$에 추가한다.
      $Q(s,a) \leftarrow \text{average}(Returns(s,a))$
   (c) 에피소드에 있는 각 s에 대해
      $A^* \leftarrow \arg\max_a Q(s,a)$
      모든 $a \in \mathcal{A}(s)$에 대해

$$\pi(a \mid s) \leftarrow \begin{cases} 1-\epsilon+\epsilon/\mid \mathcal{A}(s) \mid, & a = A^* \\ \epsilon/\mid \mathcal{A}(s) \mid, & a \neq A^* \end{cases}$$

6. 시간차 학습

- 시간차학습은 예측 기반 기계학습의 한 방법이다. TD 학습은 주로 강화학습에 사용되며 MC 방법과 동적 계획법의 조합이다. 이는 TD 학습이 어떤 정책에 따라 환경에서 에피소드를 표집하여 학습을 진행하는 MC 방법의 특징과 과거에 학습한 추정값을 사용하여 현재의 추정값을 구하는 동적 계획법의 특징을 지니고 있기 때문이다. MC 방법과 마찬가지로 TD 방법은 환경 모형 없이 원시 경험으로부터 직접 학습할 수 있다.
- TD 학습은 이어지는 예측들이 어떤 식으로 관련되어 있다고 가정한다. 일반적인 지도학습에서는 실제로 관측된 값을 통해서만 학습이 이루어진다. MC 방법은 에피소드가 끝난 후에만 추정값을 갱신하지만, TD 방법은 예측이 에피소드가 끝나기 전의 미래에 대한 예측에 잘 부합하도록 갱신한다. 이러한 과정을 부트스트랩이라고 한다.
- 먼저 주어진 정책 π 에 대한 상태 가치함수 Iv_π 를 추정하는 정책 평가 또는 예측 문제에 대해 설명한다. 그리고 최적 정책을 찾는 제어 문제 관련 GPI에 대해 설명한다. 제어 문제의 경우 동적 계획법, MC 및 TD 방법 모두 GPI의 변형을 사용한다. 이 방법들의 차이점은 주로 예측 문제에 대한 접근 방식의 차이이다.

6. 시간차 학습

1. TD 예측

- MC 및 TD 방법은 예측 문제를 풀기 위해 경험을 사용한다. 정책 π에 따라 어떤 경험이 주어질 때 두 방법 모두 그 경험에서 발생하는 비종료 상태 S_t에 대한 v_π의 추정값 V를 갱신한다. 간단히 말하면 MC 방법은 방문 후 반환값이 알려질 때까지 기다린 후 그 반환값 $V(S_t)$의 목푯값으로 사용한다. 비정상 환경에 적합한 모든-방문 MC 방법의 간단한 갱신알고리슴은 다음과 같다.

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)] \qquad (7.6.1)$$

- 여기서 G_t는 t시점 이후의 실제 반환값이며 α는 단계 크기 매개변수이다. 이 방법을 상수 α MC라고 부른다. MC 방법은 I $V(S_t)$의 증분을 결정하기 위해 에피소드가 끝날 때까지 기다려야 하지만 TD 방법은 단지 다음 시점까지만 기다리면 된다. 즉, 최종 결과를 모르더라도 학습할 수 있는 것이 TD 방법의 장점이며, 단계마다 학습할 수 있다는 것도 장점이다. TD 방법은 MC 방법에 비해 적은 메모리를 사용하고 적은 계산량을 필요로 한다. 두 방법 모두 수렴한다.

- $t+1$ 시점에서 TD 방법은 즉시 목푯값을 정하고 관측된 보상 R_{t+1}과 추정값 $V(S_{t+1})$을 사용하여 갱신한다. TD 방법의 가장 간단한 갱신 알고리즘은 식 (7.6.2)와 같다.

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \qquad (7.6.2)$$

6. 시간차 학습

- MC 알고리즘의 목푯값은 G_t 이지만 TD 알고리즘의 목푯값은 $R_{t+1}+\gamma V(S_{t+1})$ 이다. 그리고 $\delta_t = R_{t+1}+\gamma V(S_{t+1})-V(S_t)$ 는 TD 오차이다. 이 방법을 $TD(0)$ 또는 1-단계 TD라고 부른다. 이 방법은 $TD(\lambda)$ 및 $n-$ 단계 TD의 특별한 경우이다. $TD(0)$ 알고리즘은 다음과 같다.

알고리즘 7-6 v_π를 추정하기 위한 $TD(0)$

1. 입력: 평가할 정책 π
2. $V(s)$를 임의로 초기화한다. 예를 들어, $V(s)=0,\ \forall s\in S^+$
3. 각 에피소드에 대해 반복
   (a) S를 초기화한다.
   (b) S가 종료 상태가 될 때까지 에피소드의 각 단계에 대해 반복

$A \leftarrow S$에 대해 π에 의해 주어진 행동
행동 A를 선택하고 보상 R과 다음 상태 S'을 관측한다.
$V(S) \leftarrow V(S)+\alpha[R+\gamma V(S')-V(S)]$
$S \leftarrow S'$

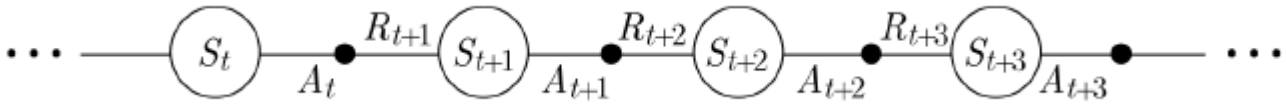
6. 시간차 학습

2. TD 제어

- 이제 TD 예측 방법을 제어 문제에 적용하는 것을 생각해 보자. 평가 또는 예측을 위해 TD방법을 사용할 때도 앞에서 설명한 방법들처럼 GPI 패턴을 따른다. MC 방법과 마찬가지로 탐색과 이용을 절충해야 할 필요성에 직면하게 된다. 그리고 TD 방법은 온폴리시 및 옵폴리시 두 가지 방법으로 나누어진다.

A. Sarsa : 온폴리시 TD 제어

- 첫 번째 단계는 상태 가치함수보다 행동 가치함수를 학습하는 것이다. 특히, 온폴리시 방법을 위해 우리는 현재 정책 π와 모든 상태 s 및 행동 a에 대해 $q_\pi(s,a)$를 추정해야 한다. 이를 위해 v_π를 학습하기 위한 TD 방법의 개념을 사용한다. 에피소드는 다음과 같이 상태 및 상태-행동 쌍의 교대 수열로 이루어진다는 것을 기억하자.



- 우리는 상태-행동 쌍에서 상태-행동 쌍으로의 전이를 생각하고 상태-행동 쌍의 가치함수의 값을 학습한다. 이것 또한 보상과정을 갖는 마르코프 과정이다. $TD(0)$하에서 상태 가치함숫값의 수렴을 보증하는 정리는 다음과 같이 행동 가치함숫값에 대한 관련 알고리즘에도 적용된다.

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha\left[R_{t+1} + \gamma Q(S_{t+1},A_{t+1}) - Q(S_t,A_t)\right] \qquad (7.6.3)$$

6. 시간차 학습

- 비종료 상태 S_t 로부터의 모든 전이를 고려한 후에 갱신이 수행된다. 만약 S_{t+1} 이 종료상태이면 $Q(S_{t+1}, A_{t+1})$ 은 0으로 정의된다.
- 이 규칙은 원소가 5개인 모든 사건, 즉 $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$ 을 사용한다. 이것은 어떤 상태-행동으로부터 다음 상태-행동으로의 전이를 구성한다. 이러한 이유로 Sarsa 알고리즘의 이름이 생겨났다. Sarsa 알고리즘은 다음과 같다.

알고리즘 7-7 $Q \approx q_*$를 추정하기 위한 Sarsa

1. 모든 $s \in S$, $a \in \boldsymbol{A}(s)$에 대해 $Q(s,a)$를 임의로 초기화하고, $Q(\text{종료 상태}, a) = 0$ 으로 설정한다.
2. 각 에피소드에 대해 반복
   (a) S를 초기화한다.
   (b) Q로부터 유도된 정책을 사용하여 S로부터 A를 선택한다(예: ϵ-탐욕).
   (c) S가 종료 상태가 될 때까지 에피소드의 각 단계에 대해 반복한다.
   행동 A를 선택한다. 보상 R과 다음 상태 S'을 관측한다.
   Q로부터 유도된 정책을 사용하여 S'으로부터 A'을 선택한다(예: ϵ-탐욕).
   $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma Q(S',A') - Q(S,A)]$
   $S \leftarrow S'$; $A \leftarrow A'$

6. 시간차 학습

B. Q-학습 : 옵폴리시 TD 제어

- 초기 강화학습의 획기적인 발전 중 하나는 다음과 같이 정의하며 Q-학습으로 알려진 옵폴리시 TD 제어 알고리즘의 개발이었다.

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1},a) - Q(S_t,A_t)] \qquad (7.6.4)$$

- 이 경우 학습된 행동 가치함수 Q는 관련 정책과는 무관하게 최적의 행동 가치함수 q_*□ 직접 근사한다. 이것은 알고리즘 분석을 획기적으로 단순화하고 수렴 증명을 가능하게 한다. 정책은 어떤 상태-행동 쌍이 방문하고 갱신되는지를 결정한다는 점에서 여전히 효과적이다. 그러나 정확한 수렴을 위해 필요한 것은 모든 쌍이 계속 갱신되는 것이다. 이러한 가정에서 Q 가 확률 1로 q_* 에 수렴하는 것이 증명되었다. Q-학습 알고리즘은 다음과 같다.

알고리즘 7-8 $\pi \approx \pi_*$를 추정하기 위한 Q-학습

1. 모든 $s \in S$, $a \in \boldsymbol{A}(s)$에 대해 $Q(s,a)$를 임의로 초기화하고, Q(종료 상태, a) = 0으로 설정한다.
2. 각 에피소드에 대해 반복
   (a) S를 초기화한다.
   (b) S가 종료 상태가 될 때까지 에피소드의 각 단계에 대해 반복한다.
   Q로부터 유도된 정책을 사용하여 S로부터 A를 선택한다(예: ϵ-탐욕).
   행동 A를 선택한다, 보상 R과 다음 상태 S'을 관측한다.
   $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
   $S \leftarrow S'$

7. 심층 Q-망

- 2013년 런던의 스타트업 기업 딥마인드는 심층 신경망을 강화학습에 적용하여 아타리게임을 실행하도록 학습할 수 있는 심층 신경망을 공개하여 기계학습 커뮤니티를 놀라게 하였다. 심층 Q-망 [DQN],이라고도 하는 이 신경망은 심층 신경망을 이용한 강화학습의 첫 번째 대규모 성공 응용 프로그램이었다.
- 비록 각 게임이 서로 다른 규칙 . 목표 . 게임플레이 구조를 가지고 있지만, DQN은 같은 아키텍처가 변경 없이 49개의 다른 아타리 게임을 학습할 수 있기 때문에 매우 주목할 만했다. 이 기술을 성취하기 위해 딥마인드는 강화학습의 많은 전통적인 아이디어를 통합하는 동시에, DQN의 성공을 입증하는 몇 가지 새로운 기술을 개발하였다. 자세한 내용을 위해 [Mnih 등2013]을 참고하라.

1. DQN

- 최적 행동 가치함수는 벨만 최적 방정식을 따른다. 즉, 만약 다음 단계에서 s' □ 최적 가치 $q_*(s',a')$ 이 모든 가능한 행동 a'□ 대해 알려진다면 최적 전략은 $r+\gamma q_*(s',a')$ 의 기댓값, 즉

$$q_*(s,a) = E_{s'\sim\varepsilon}\left[r+\gamma \max_{a'} q_*(s',a') \mid s,a\right] \qquad (7.7.1)$$

를 최대화하는 행동 a'□ 선택하는 것이다.

7. 심층 Q-망

- 강화학습의 기본개념은 반복 갱신 $Q_{i+1}(s,a) = E[r + \gamma \max_{a'} Q_i(s',a') \mid s,a]$ 를 사용하여 행동 가치함수를 추정하는 것이다. 이러한 가치 반복 알고리즘은 최적 행동 가치함수로 수렴한다.
- 즉, $i \to \infty$ 일 때 $Q_i \to q_*$가 성립한다. 실제로 이 방법은 완전히 비실용적이다. 왜냐하면 행동 가치함수는 각 수열에 대해 별도로 추정되기 때문이다. 대신 함수 근사기를 사용하여 행동 가치함수를 추정하는 것이 일반적이다.
- 즉, [$Q(s,a;\boldsymbol{\theta}) \approx q_*(s,a)$ 이다. 이때 $Q(s,a;\boldsymbol{\theta})$ □ Q-함수를 모형화한 심층 신경망을 나타내고 심층 Q-망이라고 불린다. 여기서 $\boldsymbol{\theta}$ 는 신경망의 모든 가중치와 편의로 이루어진 매개변수 벡터이다. 만약 가정해지면 상태 s 와 행동 a를 심층 Q-망에 입력으로 넣어 Q-값을 얻을 수 있다. 이를 통해 □ 의 값과 최적 행동 a_* 를 구할 수 있다.
- 다시 말해, 심층 Q-망을 올바른 방향으로 갱신하는 알고리즘을 만든다면 주어진 문제를 해결할 수 있다. 따라서 우리가 원하는 목적대로 심층 Q-망이 학습되기 위하여 i□□ 반복에서 다음과 같은 손실함수를 가지도록 설계한다. [Mnih등2013]은 현재 심층 Q-망이 항상 목표 Q-값에 가까워지도록 손실함수를 설정함으로써 마치 가치 반복이 수렴하듯이 심층 Q-망의 갱신이 수렴할 때까지 반복하도록 한다.

$$L_i(\boldsymbol{\theta}_i) = E_{s,a \sim \rho}\left[(y_i - Q(s,a;\boldsymbol{\theta}_i))^2\right] \qquad (7.7.2)$$

7. 심층 Q-망

- 여기서 $y_i = E_{s' \sim \varepsilon}\left[r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) \mid s, a\right]$는 i번째 반복에서의 목푯값이며 $\rho(s, a)$는 상태 s와 행동 a의 확률분포이다. [Mnih 등2013]에서는 $\rho(s, a)$를 행동분포라고 정의한다.
- 손실함수 $L_i(\theta_i)$를 최적화할 때 전 단계 반복에서 얻은 매개변수 벡터 θ_{i-1}은 고정된다. 목푯값이 신경망 가중치에 의존한다는 것을 주목하라. 이는 학습이 시작되기 전에 고정되어있는 지도학습에 사용되는 목푯값과는 대조적이다. 손실함수를 θ_i에 대해 미분하면 다음과 같은 기울기를 얻는다.

$$\nabla_{\theta_i} L_i(\theta_i) = E_{s,a \sim \rho; s' \sim \varepsilon}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i)\right) \nabla_{\theta_i} Q(s, a; \theta_i)\right] \qquad (7.7.3)$$

- 앞의 기울기에서 기댓값을 계산하는 대신 확률적 경사 하강법으로 손실함수를 최적화하는 것이 계산상 편리하다. 만약 매개변수가 반복 단계마다 갱신되고 기댓값이 행동분포 ρ와 에뮬레이터 ε에 대한 단일 표본으로 대체되면 우리는 익숙한 Q-학습 알고리즘에 도달한다.
- 이 알고리즘은 모형을 사용하지 않는다는 점에 유의해야 한다. 이 알고리즘은 에뮬레이터 ε의 추정값을 명시적으로 구성하지 않고도 ε에 대한 표본을 직접 사용하여 강화학습 문제를 해결할 수 있다. 또한 이것은 옵폴리시 알고리즘이다. 이 알고리즘은 상태공간의 적절한 탐색을 보장해 주는 행동분포를 따르면서 탐욕 정책 $a = \max_a Q(s, a; \theta)$에 대해 학습한다. 실제로행동분포는 확률 $1 - \epsilon$으로 탐욕 전략을 따르고 확률 ϵ로 무작위 행동을 선택하는 ϵ-탐욕 전략에 의해 선택된다.

7. 심층 Q-망

A. 심층 강화학습

- 우리의 목표는 강화학습 알고리즘을 복잡한 구조의 데이터에 직접 작동하는 심층 신경망에 연결하고 확률적 경사 하강법을 사용하여 훈련 데이터를 효율적으로 처리하는 것이다. 그러나 다음과 같은 몇 가지 문제점 때문에 심층 강화학습을 바로 사용할 수는 없다.

  - 딥러닝의 표본은 일반적으로 서로 독립이고 동일한 분포$^{i.i.d}$를 따르지만 강화학습의 표본은 일반적으로 순차적^{sequential}이고 매우 상관되어 있다.
  - 정책 변화에 따른, 즉 매개변수 벡터 θ에 따른 Q-값의 변화량이 너무 크기 때문에 정책이 진동하기 쉽다.
  - 위와 같은 설정에서는 보상과 Q-값이 엄청나게 커질 수 있으므로 안정된 갱신이 어려워진다.

- 첫 번째 문제점은 강화학습에서 당연한 것이다. 두 번째 문제점도 크게 어렵지 않게 생각할 수 있는데, 게임을 하는 방식을 아주 조금만 바꾸더라도 게임의 결과가 완전히 크게 바뀌기 때문에 이러한 현상이 발생한다. 탁구 게임에서 움직이는 속도를 조금 늦추면 바로 한 점을 잃게 될 것이다. 또한 지도학습과는 다르게 목푯값이 매개변수에 아주 민감하게 영향을받기 때문에 매개변수를 고정해 주는 것이 필요하다. 마지막 문제점은 좀 실질적인 것인데, Q-값이 얼마나 커질지 모르므로 안정된 갱신이 힘들 수도 있다. 일반적으로 다음과 같은 세가지 방법으로 각 문제점을 해결한다.

7. 심층 Q-망

- 경험 재생experience replay
- 목표 Q-망을 고정한다.
- 보상을 적당한 값으로 고정clip하거나 망을 적당한 범위로 적응적으로 정규화한다.

- 두 번째 방법은 갱신과정 동안 목푯값을 계산하기 위해 사용하는 매개변수를 고정하는 것이다. 세 번째 방법은 보상을 $\{-1, 0, 1\}$중 하나만 선택하도록 강제하는 것이다. 즉, 모든양의 보상을 1로 고정하고, 모든 음의 보상을 –1로 고정하며 보상 0은 변하지 않게 한다. 이렇게 함으로써 좀 더 안정된 갱신이 가능하다. 따라서 여러 게임에서 동일한 학습률을 사용하는 것이 더 쉬워진다.
- 이제 [Mnih 등2013]의 핵심개념이라고 할 수 있는 경험 재생에 대해 살펴보자. 경험 재생은 각 시점에서의 에이전트의 경험 $e_t = (s_t, a_t, r_t, s_{t+1})$ 을 재생 메모리의 데이터 집합 $\boldsymbol{D} = \{e_1, \cdots, e_N\}$ 에 저장한 후 이를 다시 사용하는 것이다.
- 경험 재생은 이렇게 경험 e_t를 메모리에 저장하였다가 일부를 무작위로 추출하여 미니배치를 구성한 다음 신경망의 매개변수를 갱신하는 과정을 의미한다. 경험 재생을 사용함으로써 표본의 상관관계를 없앨 수 있으며 나아가 i.i.d 설정에서 신경망을 학습할 수 있게 된다. 또한, 방대한 과거 표본을 한 번만 갱신에 사용하고 버리는 대신에 향후 갱신에도 계속 영향을 줄 수 있도록 사용하기 때문에 훨씬 효율적이라는 장점이 있다. 재생 메모리는 크기가 정해져 있어 일단 메모리가 꽉 차면 맨 처음 들어온 표본부터 메모리에서 삭제한다. 에이전트가 학습을 하여 점점 높은 점수를 받게 되면 더 좋은 표본을 재생 메모리에 저장한다. 심층 Q-학습 알고리즘은 다음과 같다. 알고리즘의 내부 반복 루프에서 표본 경험에 Q-학습, 미니배치 갱신을 적용한다.

7. 심층 Q-망

알고리즘 7-9 경험 재생을 사용하는 심층 Q-학습

1. 재생 메모리 D를 크기가 N이 되도록 초기화한다.
2. 행동 가치함수 Q가 무작위 가중치를 가지도록 초기화한다.

for 에피소드$=1, M$ do

수열 $s_1=\{x_1\}$ 및 임의의 함수를 사용하여 전처리된 수열 $\phi_1=\phi(s_1)$을 초기화한다.

for $t=1, T$ do

확률 ϵ으로 무작위 행동 a_t를 선택한다.

그렇지 않으면 $a_t=\max_a q_*(\phi(s_t), a;\theta)$를 선택한다.

에뮬레이터에서 행동 a_t를 실행하고 보상 r_t와 이미지 x_{t+1}을 관측한다.

$s_{t+1}=s_t, a_t, x_{t+1}$을 설정하고 전처리 $\phi_{t+1}=\phi(s_{t+1})$을 시행한다.

전이 $(\phi_t, a_t, r_t, \phi_{t+1})$을 D에 저장한다.

D로부터 전이 $(\phi_j, a_j, r_j, \phi_{j+1})$의 무작위 미니배치를 표집한다.

$y_j=\begin{cases} r_j, & \text{종료 } \phi_{j+1} \\ r_j+\gamma\max_{a'} Q(\phi_{j+1}, a';\theta), & \text{비종료 } \phi_{j+1} \end{cases}$ 을 설정한다.

식 (7.7.3)에 따라 $\left(y_j-Q(\phi_j, a_j;\theta)\right)^2$에 대해 기울기 하강 단계를 수행한다.

end for

end for

7. 심층 Q-망

B. 경험 재생을 사용하는 심층 Q – 학습 알고리즘의 장점

- 경험 재생을 사용하는 심층 Q-학습은 온라인 Q-학습에 대해 다음과 같은 장점이 있다.

  - 각 경험은 잠재적으로 많은 가중치 갱신에 사용되므로 데이터 효율성이 더 높다.
  - 연속된 표본을 사용하여 학습하면 표본 간의 강한 상관관계 때문에 비효율적이다. 표본을 무작위로 추출하면 이러한 상관관계가 깨져 갱신의 분산이 줄어든다.
  - 온폴리시 학습을 수행할 때 현재의 매개변수는 다음 학습을 위한 데이터 표본을 결정한다. 예를 들어, 왼쪽으로 움직이는 행동을 선택하면 다음 학습 표본들은 대부분 왼쪽에 있는 표본이 될 것이다.

2. DQN 텐서플로 예제

- 예제는 [그림 7-13]과 같이 언덕을 자유롭게 움직일 수 있는 검은색 자동차와 자동차가 도착해야 하는 지점을 깃발로 표시한 Mountain Car 예제이다. 이 예제의 목적은 자동차가 오른쪽 언덕을 올라가 깃발 지점에 도착하는 것이다. 하지만 자동차는 언덕 위로 올라갈 만큼의 힘을 가지고 있지 않다. 에이전트는 오른쪽 언덕을 올라가기 위해 자동차가 충분한 추진력을 얻는 방법을 학습하며, 추진력을 얻기 위해 자동차는 오른쪽과 왼쪽으로 움직이는 과정을 반복해야 한다.

7. 심층 Q-망

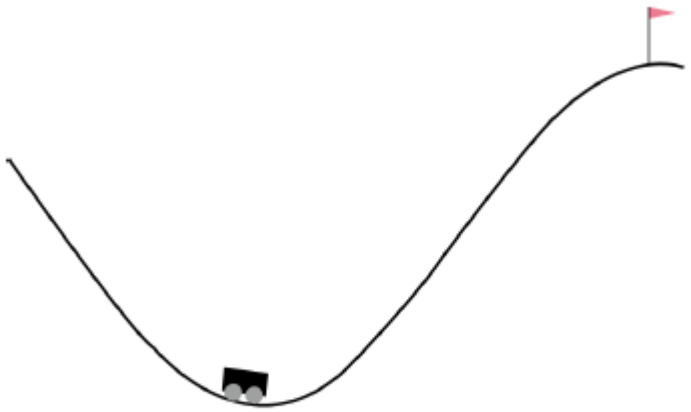


그림 7-13 Mountain Car 예제

- 우선, DQN 알고리즘에 사용되는 매개변수부터 살펴보자.
  - learning_rate: 신경망의 학습률 매개변수
  - gamma: 보상을 시간에 따라 얼마나 할인할지 정해 주는 할인율 매개변수
  - n_features: 환경으로부터 받은 에이전트의 입력값으로 자동차의 위치(0)와 속도(1)
  - n_actions: 에이전트가 선택한 행동으로서 왼쪽(0)·오른쪽으로 밀기(2)와 보류(1) 3가지
  - epsilon: 탐욕 정책 시 활용되는 탐욕의 초깃값

- batch_size: 신경망 학습을 위해 재생 메모리로부터 추출되는 표본의 크기
- experience_counter: 현재 재생 메모리에 저장된 표본의 수
- experience_limit: 재생 메모리의 최대 용량
- replace_target_pointer: 목표 신경망target network 갱신 기준 학습단계
- learning_counter: 기본 신경망primary network의 학습단계
- memory: 재생 메모리의 초깃값(0으로 설정)

예제 7-3 **TensorFlow: DQN** 텐서플로 코드

```
import numpy as np
import tensorflow as tf
import gym

class DQN:
    def __init__(self,learning_rate, gamma, n_features, n_actions, epsilon,
        parameter_changing_pointer, memory_size):
        self.learning_rate = learning_rate
        self.gamma = gamma
        self.n_features = n_features
        self.n_actions = n_actions
        self.epsilon = epsilon
        self.batch_size = 100
```

7. 심층 Q-망

```
self.experience_counter = 0
self.experience_limit = memory_size
self.replace_target_pointer = parameter_changing_pointer
self.learning_counter = 0
self.memory = np.zeros([self.experience_limit,self.n_features*2+2])

self.build_networks()
p_params = tf.get_collection('primary_network_parameters')
t_params = tf.get_collection('target_network_parameters')
self.replacing_target_parameters = [tf.assign(t,p) for t,p in
                                                    zip(t_params,p_params)]

self.sess = tf.Session()
self.sess.run(tf.global_variables_initializer())
```

7. 심층 Q-망

- 다음은 DQN 모형의 신경망을 설정하는 단계이다. DQN 알고리즘은 신경망 모형학습 시갱신의 목표인 정답이 학습단계마다 변하는 문제점이 발생한다. 이를 해결하기 위해 정답을 만들어 내는 기본 신경망과 최종 도출하고자 하는 목표 신경망을 따로 만든다. 기본 신경망을 통해 학습된 결과를 바탕으로 일정 시간마다 목표 신경망을 갱신하여 최종적으로 모형을 도출한다. 우선, 자동차의 움직임에 대한 행동을 만들어 내는 기본 신경망을 설정한다. 2개의은닉층으로 구성된 신경망 모형을 정의했으며, 각 은닉층은 10개의 은닉노드를 가지고 있다. 최적화함수로는 Adam Optimizer를 활용하였다.

예제 7-3 계속 **TensorFlow: DQN** 텐서플로 코드

```
def build_networks(self):
    hidden_units = 10
    # Primary Network
    self.s = tf.placeholder(tf.float32,[None,self.n_features])
    self.qtarget = tf.placeholder(tf.float32,[None,self.n_actions])

    with tf.variable_scope('primary_network'):
        c = ['primary_network_parameters',
                                        tf.GraphKeys.GLOBAL_VARIABLES]
```

7. 심층 Q-망

```
# first layer
with tf.variable_scope('layer1'):
    w1 = tf.get_variable('w1', [self.n_features, hidden_units],
              initializer=tf.contrib.layers.xavier_initializer(),
                         dtype=tf.float32,collections=c)

    b1 = tf.get_variable('b1', [1, hidden_units],
              initializer=tf.contrib.layers.xavier_initializer(),
                          dtype=tf.float32,collections=c)

    l1 = tf.nn.relu(tf.matmul(self.s, w1) + b1)

# second layer
with tf.variable_scope('layer2'):
    w2 = tf.get_variable('w2', [hidden_units, self.n_actions],
              initializer=tf.contrib.layers.xavier_initializer(),
```

7. 심층 Q-망

```
                                dtype=tf.float32,collections=c)

        b2 = tf.get_variable('b2', [1, self.n_actions],
                  initializer=tf.contrib.layers.xavier_initializer(),
                                dtype=tf.float32,collections=c)

        self.qeval = tf.matmul(l1, w2) + b2

with tf.variable_scope('loss'):
         self.loss = tf.reduce_mean(tf.squared_difference(
                                                self.qtarget, self.qeval))
with tf.variable_scope('optimiser'):
         self.train = tf.train.AdamOptimizer(self.learning_rate).
                                                        minimize(self.loss)
```

7. 심층 Q-망

- 기본 신경망을 설정한 후 최종 도출하고자 하는 목표 신경망을 기본 신경망과 동일한 형태로 정의한다.

예제 7-3 계속 TensorFlow: DQN 텐서플로 코드

```
# Target Network
self.st = tf.placeholder(tf.float32,[None,self.n_features])

with tf.variable_scope('target_network'):
    c = ['target_network_parameters', tf.GraphKeys.GLOBAL_VARIABLES]
    # first layer
    with tf.variable_scope('layer1'):
        w1 = tf.get_variable('w1', [self.n_features, hidden_units],
                        initializer=tf.contrib.layers.xavier_initializer(),
                          dtype=tf.float32,collections=c)

        b1 = tf.get_variable('b1', [1, hidden_units],
                        initializer=tf.contrib.layers.xavier_initializer(),
                          dtype=tf.float32,collections=c)
```

7. 심층 Q-망

```
        l1 = tf.nn.relu(tf.matmul(self.st, w1) + b1)

    # second layer
    with tf.variable_scope('layer2'):
        w2 = tf.get_variable('w2', [hidden_units, self.n_actions],
                             initializer=tf.contrib.layers.xavier_initializer(),
                               dtype=tf.float32,collections=c)

        b2 = tf.get_variable('b2', [1, self.n_actions],
                             initializer=tf.contrib.layers.xavier_initializer(),
                               dtype=tf.float32,collections=c)

        self.qt = tf.matmul(l1, w2) + b2
```

7. 심층 Q-망

- 신경망 설정단계 다음으로는 기본 신경망의 학습 결과를 목표 신경망에 대입하기 위한 세션을 구성한다. 그리고 에이전트가 시간단위마다 경험한 상태, 행동, 보상을 재생 메모리에 저장하는 함수를 정의한다.

예제 7-3 계속 TensorFlow: DQN 텐서플로 코드

```
def target_params_replaced(self):
    self.sess.run(self.replacing_target_parameters)

def store_experience(self,obs,a,r,obs_):
    index = self.experience_counter % self.experience_limit
    self.memory[index,:] = np.hstack((obs,[a,r],obs_))
    self.experience_counter+=1
```

7. 심층 Q-망

- 다음은 실제 학습이 진행되는 단계이다. 재생 메모리에 저장된 과거 경험을 설정한 배치 크기만큼 랜덤으로 추출하여 학습 데이터 집합으로 설정한다. 여기서 epsilon이 조정되며 학습이 진행되는데, 0.9 미만인 경우 epsilon은 학습단계마다 0.0002씩 증가한다.

예제 7-3 계속 TensorFlow: DQN 텐서플로 코드

```
def fit(self):
    # sample batch memory from all memory
    if self.experience_counter < self.experience_limit:
        indices = np.random.choice(self.experience_counter,size=self.batch_size)
    else:
        indices = np.random.choice(self.experience_limit,size=self.batch_size)

    batch = self.memory[indices,:]
    qt,qeval = self.sess.run([self.qt,self.qeval],
     feed_dict={self.st:batch[:,-self.n_features:],self.s:batch[:,:self.n_features]})
```

7. 심층 Q-망

```
qtarget = qeval.copy()
batch_indices = np.arange(self.batch_size, dtype=np.int32)
actions = self.memory[indices,self.n_features].astype(int)
rewards = self.memory[indices,self.n_features+1]
qtarget[batch_indices,actions] = rewards + self.gamma * np.max(qt,axis=1)

_ = self.sess.run(self.train,feed_dict = {self.s:batch[:,:self.n_features],
                                          self.qtarget:qtarget})
if self.epsilon < 0.9:
   self.epsilon += 0.0002
```

7. 심층 Q-망

- 또한, 학습 시 기본 신경망의 가중치를 가져와 목표 신경망의 가중치를 갱신한다. 목표 신경망의 가중치 갱신은 현재 기본 신경망의 학습단계(learning_counter)가 목표 신경망 갱신 기준 학습단계(replace_target_pointer)로 정확히 나누어지는 단계마다 진행된다. 예를 들어, replace_target_pointer를 500으로 설정한다면, 학습이 500번 진행될 때마다 목표 신경망의 가중치가 갱신된다.

예제 7-3 계속 TensorFlow: DQN 텐서플로 코드

```
if self.learning_counter % self.replace_target_pointer == 0:
    self.target_params_replaced()
    print("target parameters changed")
self.learning_counter += 1
```

7. 심층 Q-망

- 마지막으로 정의되는 함수는 탐욕 정책을 통해 행동을 선택하는 함수이다. 훈련 초기에는 무작위로 행동을 선택하지만, 훈련이 진행되고 epsilon 값이 높아짐에 따라 모형의 예측에 따라 행동을 선택한다.

예제 7-3 계속 TensorFlow: DQN 텐서플로 코드

```
def epsilon_greedy(self,obs):
    #epsilon greedy implementation to choose action
    if np.random.uniform(low=0,high=1) < self.epsilon:
        return np.argmax(self.sess.run(self.qeval,
                                       feed_dict={self.s:obs[np.newaxis,:]}))
    else:
        return np.random.choice(self.n_actions)
```

7. 심층 Q-망

- 다음은 DQN 객체를 생성하여 에이전트를 학습시키고 결과를 도출하는 단계이다. 우선, gym. make 함수를 통해 OpenAI GYM에서 MountainCar-v0 환경을 불러온다. 환경을 불러온 후 DQN 객체를 생성하여 에이전트를 학습시킨다. 매개변수 설정은 다음과 같다. 즉, 신경망의 학습률을 0.001로, 할인율 감마를 0.9로, 입력변수의 개수를 2로, 행동의 개수를 3으로, epsilon의 초깃값을 0으로, 목표 신경망 갱신 기준 학습단계를 500으로, 재생 메모리의 최대 용량을 5,000으로 설정한다. 그리고 총 10단계의 에피소드를 설정하여 학습을 수행하고 결괏값을 반환받는다.

예제 7-3 계속 TensorFlow: DQN 텐서플로 코드

```
if __name__ == "__main__":
    env = gym.make('MountainCar-v0')
    env = env.unwrapped
    dqn = DQN(learning_rate=0.001, gamma=0.9, n_features=env.observation_space.shape[0],
          n_actions=env.action_space.n, epsilon=0.0, parameter_changing_pointer=500,
```

7. 심층 Q-망

```
        memory_size=5000)

episodes = 10
total_steps = 0

for episode in range(episodes):
    steps = 0
    obs = env.reset()
    episode_reward = 0
    while True:
        env.render()
        action = dqn.epsilon_greedy(obs)
        obs_,reward,terminate,_ = env.step(action)
        reward = abs(obs_[0]+0.5)
```

7. 심층 Q-망

```
            dqn.store_experience(obs,action,reward,obs_)
            if total_steps > 1000:
                 dqn.fit()
            episode_reward+=reward
            if terminate:
                break
            obs = obs_
            total_steps+=1
            steps+=1
        print("Episode {} with Reward : {} at epsilon {} in steps {}".
                          format(episode+1,episode_reward,dqn.epsilon,steps))

while True:
    env.render()
```