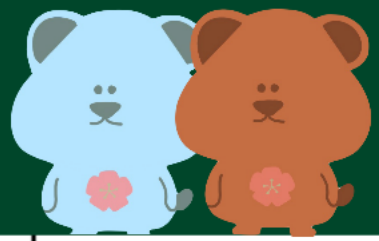
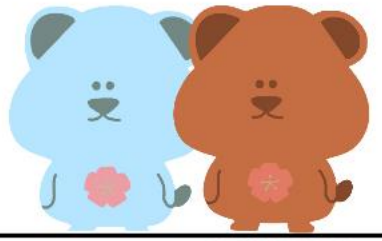


17주차 발표

박보영 오수진 오연재

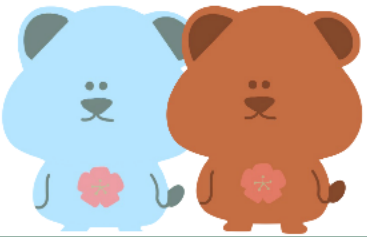
EWHA EURON

목차



EWHA EURON

01. 강화 학습

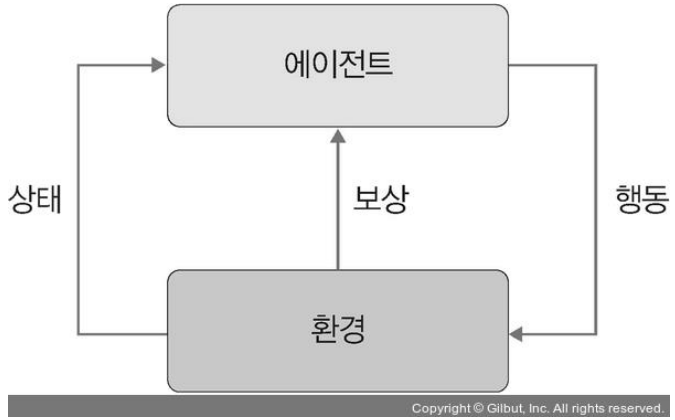


EWHA EURON

1. 강화 학습

강화학습

- 머신러닝/딥러닝의 한 종류
- 어떤 환경에서 어떤 행동을 했을 때 그것이 잘된 행동인지 잘못된 행동인지를 판단하고 보상(또는 벌칙)을 주는 과정을 반복하여 스스로 학습하는 것
- 구성요소: 환경, 에이전트
  - 환경: 에이전트가 다양한 행동을 해 보고, 그에 따른 결과를 관측할 수 있는 시뮬레이터
  - 에이전트: 환경에서 행동하는 주체
- 목표: 환경과 상호작용하는 에이전트 학습시키는 것
  - 에이전트는 상태라고 하는 다양한 상황 안에서 행동을 취하며, 조금씩 학습함
  - 에이전트 행동에 대한 응답은 양/음/0의 보상을 받음
    - 상태: 에이전트가 관찰 가능한 상태의 집합, 자신의 상황에 대한 관찰. $S_t = s \{s \in S\}$
    - 행동: 에이전트가 상태 St에서 가능한 행동. $A_t = a \{a \in A\}$
- 로봇 공학,알파고, 자율 주행 자동차 등에 활용됨



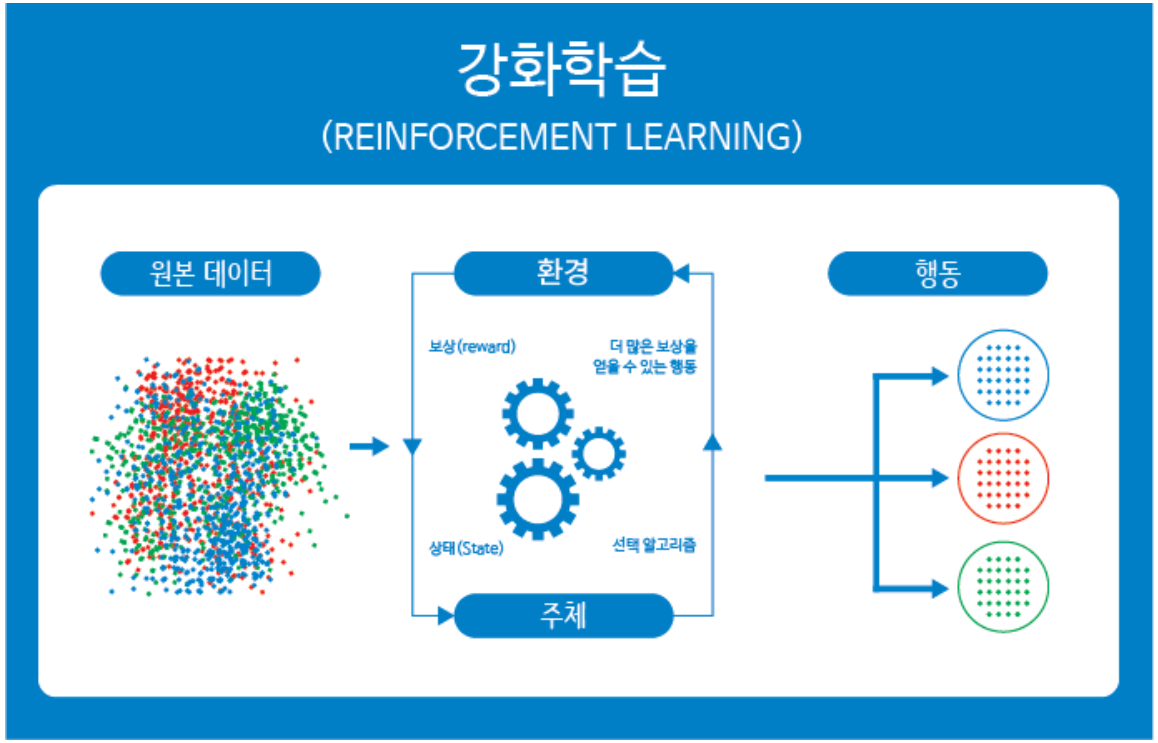
지도/비지도 학습	강화학습
학습 데이터가 주어진 상태에서, 정적 환경(환경 변화x)에서 학습 진행	환경 안에서 정의된 주체가 현재의 상태를 관찰하여 선택할 수 있는 행동 중, 최대의 보상을 가져다주는 행동 학습하는 것 → 역동적인 환경에서 동작하며 수집된 경험으로부터 학습 ➔ 훈련 전 데이터 수집, 전처리 등에 대한 필요성 해소
	사람으로부터 학습 받는 것이 아닌, 변화된 환경으로부터 보상 받아 학습함

1. 강화 학습

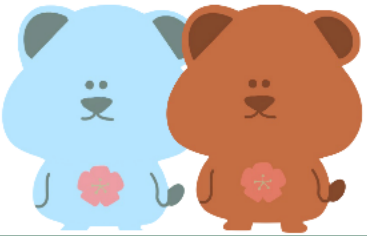
강화학습 동작 순서

1. 정의된 주체(agent)가 주어진 환경(environment)의 현재 상태(state)를 관찰하여, 이를 기반으로 행동(action)을 취함
2. 이때 환경의 상태가 변화하면서 정의된 주체는 보상(reward)을 받게 됨
3. 이 보상을 기반으로 정의된 주체는 더 많은 보상을 얻을 수 있는 방향(best action)으로 행동을 학습하게 됨

** 강화 학습에서의 '관찰-행동-보상'에 이르는 일련의 과정을 경험(experience)라고 부름.



02. 마르코프 결정 과정



EWHA EURON

1. 마르코프 프로세스(MP)

$MP \equiv (S, P)$: 마르코프 프로세스는 상태(S)와 전이확률행렬(P)로 정의

- 어떤 상태가 일정한 간격으로 변하고, 다음 상태는 현재 상태에만 의존하는 확률적 상태 변화를 의미함.

  즉, 현재 상태에 대해서만 다음 상태가 결정되며, 현재 상태까지의 과정은 고려할 필요 없음. ➔ **마르코프 체인**(변화 상태들이 체인처럼 엮여 있다)
- 마르코프 특성을 지니는 이산 시간에 대한 확률 과정
  - 확률 과정: 시간에 따라 어떤 사건의 확률이 변화하는 과정
  - 이산 시간: 시간이 이산적으로 변함
  - 마르코프 특성: 과거 상태들(S1, …, St-1)과 현재 상태(St)가 주어졌을 때, 미래 상태는 과거 상태와는 독립적으로 현재 상태로만 결정되는 것

    ➔ 즉, 과거와 현재 상태 모두를 고려했을 때 미래 상태가 나타날 확률과, 현재 상태만 고려했을 때 미래 상태가 발생할 **확률이 모두 동일**하다는 것

    ➔ $P(S_{t+1} \mid S_t) = P(S_{t+1} \mid S_1, \cdots, S_t)$
- 마르코프 체인: 시간에 따른 상태 변화를 나타냄
  - 전이: 상태 변화(상태 간 이동)
  - 상태 전이 확률(전이를 확률로 표현): 어떤 상태 i가 있을 때, 그 다음 상태 j가 될 확률

$$P(S_{t+1} = s' \mid S_t = s) = P_{ss'}$$

S: 상태의 집합

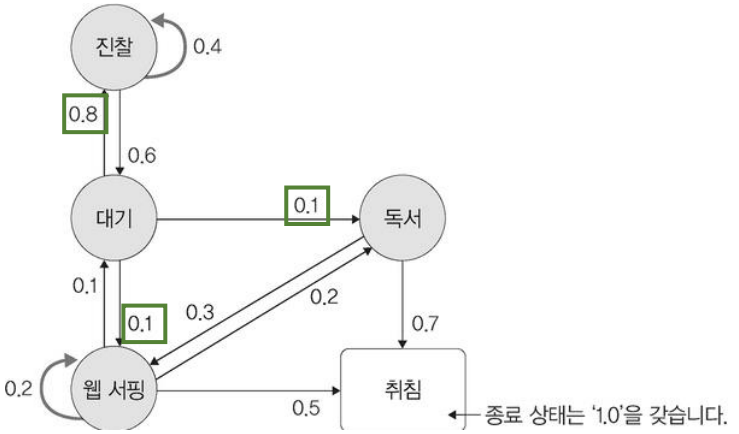
$P_{ss'}$: 상태의 변화를 확률로 표현

- *t에서의 상태: s*
- *t+1에서의 상태: s'*

1. 마르코프 프로세스(MP)

- 마르코프 체인 예시
  - 5가지 상태: 병원 대기, 진찰, 독서, 웹 서핑, 취침
  - 하나의 상태에서 다른 상태로 상태 전이가 일어남
  - '취침' 은 종료 상태
  - 하나의 상태에서 다른 상태로 이동할 확률의 합은 1을 유지, 여러 상태가 연쇄적으로 이어져 있음
- 상태 전이 확률 행렬: 각 행의 요소 모두 더하면 1이 됨
- 주의 사항:
  - 현재 상태는 바로 직전의 상태에만 의존 (O)
  - 유일하게 결정 (X)

    → 지금 현재 상태에서 다음 상태를 결정할 때는 여러가지 확률 변수가 개입함



- *마르코프 프로세스 사례*

	대기	독서	웹 서핑	진찰	취침
대기		0.1	0.1	0.8	
독서			0.3		0.7
웹 서핑	0.1	0.2	0.2		0.5
진찰	0.6			0.4	
취침					1.0

- *상태 전이 확률 행렬(상태 전이 확률을 행렬 형태로 정리)*

2. 마르코프 보상 프로세스(MRP)

- 마르코프 보상 프로세스: 마르코프 프로세스에 **보상의 개념이 추가**된 것
  - 마르코프 프로세스에서 각 상태마다 좋고 나쁨이 추가된 확률 모델
  - 어떤 상태에 도착하게 되면, 이동 결과에 대해 보상 혹은 벌칙을 주는 것

$$\mathbf{MRP} \equiv (\mathbf{S}, \mathbf{P}, \mathbf{R}, \boldsymbol{\gamma})$$

- S: 상태의 집합
- P: 전이 확률 행렬(상태 s에서 s' 로 갈 확률을 행렬 형태로 표현)
- R: 보상 함수(어떤 상태 s에 도착했을 때 받게 되는 보상) $R = E[R_t | s_t = s]$
- γ: 할인율

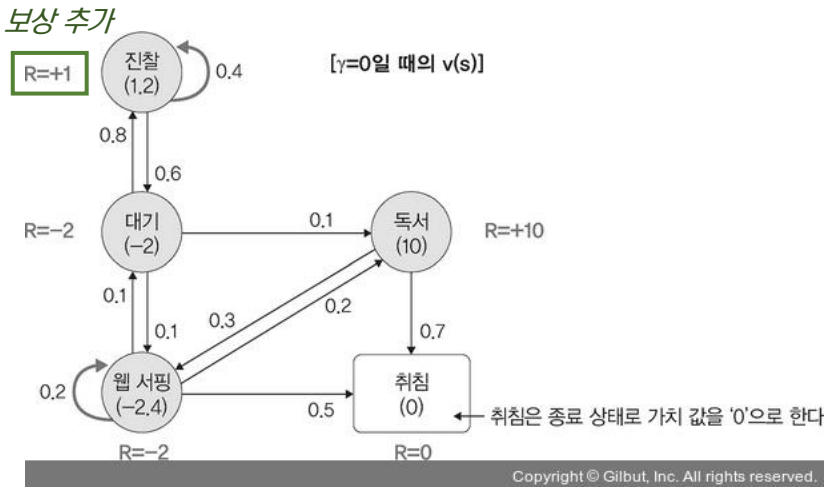
- **할인율(γ)** → 미래 가치를 현재 시점에서의 가치로 변환
  - $0 < \gamma < 1$
  - 미래 얻을 보상에 비해 당장 얻는 보상을 얼마나 더 중요하게 여길 것인지를 나타냄
  - 미래에 얻을 보상의 값에 γ가 여러 번 곱해지며 그 값을 작게 만드는 역할을 함
  - 미래에 얻게 될 보상에 비해 현재 얻는 보상에 가중치를 줌
  - γ =0: 미래 평가 절하/근시안적(미래 보상 모두 0이 됨), γ =1: 원시안적(현재와 미래 보상이 완전 대등)
  - Gt가 무한한 값 갖는 것 방지, 미래 가치를 현재 시점에서의 가치로 변환해줌

2. 마르코프 보상 프로세스(MRP)

- **에피소드** $s_0, R_0, s_1, R_1, s_2, R_2, \cdots, s_t, R_t$
  - MRP에서는 MP와 다르게 상태가 바뀔 때마다 해당하는 보상을 얻음
  - 상태 s0에서 보상 R0을 받고 시작하여 종료 상태인 st에 도착할 때 보상 Rt를 받으며 끝남
  - S0에서 st까지 가는 여정
- **리턴** $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots$
  - 각 상태의 보상 총합(t시점부터 미래에 받을 감쇠된 보상의 합)
  - T시점 이후에 발생하는 모든 보상의 값을 더해줌
  - 현재에서 멀어질수록(=미래에 발생할 보상일수록) γ가 여러 번 곱해짐
- **가치함수** $v(s) = E[G_t \mid S_t = s]$
  - 현재 상태가 s일 때 앞으로 발생할 것으로 기대되는 모든 보상의 합
  - 즉, 현재 시점에서 미래의 모든 기대되는 보상을 표현하는 미래 가치
  - 강화 학습의 핵심 및 목표: 가치 함수를 최대한 정확하게 찾는 것 ➔ **미래 가치가 가장 클 것으로 기대되는 결정을 하고 행동하는 것**

2. 마르코프 보상 프로세스(MRP)

- 할인율 0 반영 → 미래 보상 고려 x



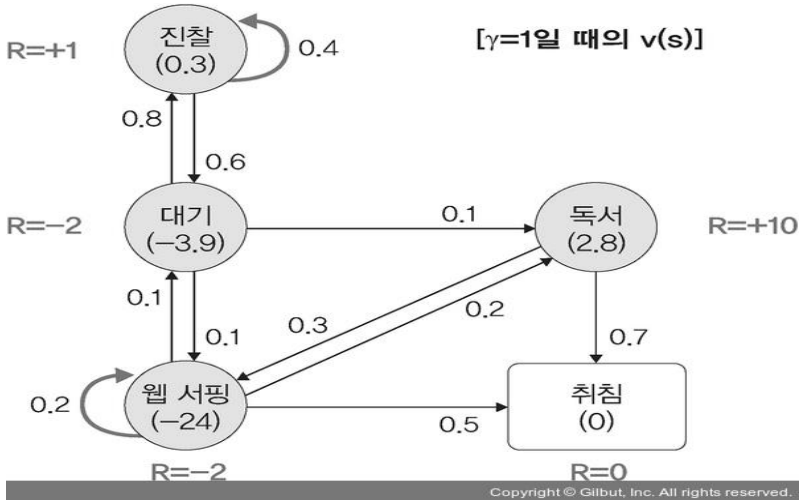
"독서" = 10 + 0×[(-2.4×0.3) + (0×0.7)] = 10

"대기" = -2 + 0×[(-2.4×0.1) + (10×0.1) + (1.2×0.8)] = -2

→ 미래 보상 모두 0

➔ 근시안적인 가치 값으로 나타남

- 할인율 1 반영 → 미래 가치에 대한 할인 고려 x



"독서" = 10 + 1×[(-24×0.3) + (0×0.7)] = 2.8

"대기" = -2 + 1×[(-24×0.1) + (2.8×0.1) + (0.3×0.8)] = -3.9

→ 모든 미래 가치에 대한 할인 고려 x

➔ 원시안적인 가치 값으로 나타남

3. 마르코프 결정 과정(MDP)

$$MDP \equiv (S, A, P, R, \gamma)$$

- A: 상태 s에서 취할 수 있는 액션들의 set
- MRP에 행동 추가된 것

- 마르코프 결정 과정: 마르코프 보상 과정에서 행동이 추가된 확률 모델
- 목표: 정의된 문제에 대해 각 상태마다 전체적인 보상을 최대화하는 행동이 무엇인지 결정하는 것
- 정책 함수: 각각의 상태마다 행동 분포(행동이 선택될 확률) 표현한 함수(즉, 각 상태에서 어떤 액션 선택할지 정해주는 함수)

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

: 상태 s에서 액션 a 선택할 확률

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

→ MDP가 주어진 π를 따를 때, s에서 s '로 이동할 확률

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

→ 이때 s에서 얻을 수 있는 보상(R)

3. 마르코프 결정 과정(MDP)

상태-가치 함수

- 상태 s에서 얻을 수 있는 리턴의 기댓값을 의미함, 주어진 정책에 따라 행동을 결정하고 다음 상태로 이동함.

$$
\begin{aligned}
v_\pi(s) &= E_\pi[G_t \mid S_t = s] \\
&= E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s] \\
&= E_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\
&= E_\pi[\underbrace{R_{t+1}}_{①} + \underbrace{\gamma v_\pi(S_{t+1})}_{②} \mid S_t = s]
\end{aligned}
$$

미래의 보상에 할인율을 곱한 것

t+1 시점에서 받는 보상, 즉각적 보상

3. 마르코프 결정 과정(MDP)

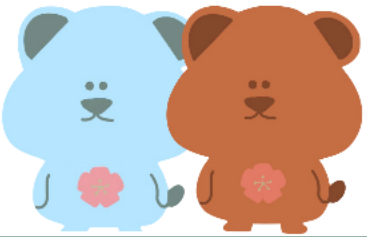
행동-가치 함수

- 상태 s에서 a라는 행동을 취했을 때, 얻을 수 있는 리턴의 기댓값을 의미함 → 시간 복잡도($O(n^3)$)가 필요하기 때문에, 상태 수가 많으면 적용하기 어려움

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

다이나믹 프로그래밍	• 마르코프 결정 과정의 상태와 행동이 많지 않고 모든 상태와 전이 확률 알고 있을 때 사용 • 대부분의 강화학습 문제 상태 많기 때문에 적용하기 어려움
몬테카를로	• 전체 상태 중 일부 구간만 방문하여 근사적으로 가치 추정함 • 초기 상태에서 시작하여 중간 상태들을 경유해서 최종 상태까지 간 후 최종 보상을 측정하고 방문했던 상태들의 가치 업데이트
시간 차 학습	• 몬테카를로 단점: 최종 상태까지 도달한 후에 방문한 상태들의 업데이트 가능 • 최종 상태에 도달하기 전에 방문한 상태의 가치 즉시 업데이트 • 다이나믹 프로그램과 몬테카를로의 중간적 특성, 본격적 강화 학습 단계
함수적 접근 학습	• 상태가 아주 많거나, 상태가 연속적인 값을 갖는 경우 → 상태와 관련된 특성 벡터 도입 • 특성의 가중치를 업데이트하여 가치의 근사치를 찾음

03. MDP를 위한 벨만 방정식



EWHA EURON

3.1 벨만 기대 방정식

벨만 방정식: 상태-가치 함수, 행동-가치 함수 간의 관계를 나타내는 방정식(벨만 기대 방정식, 벨만 최적 방정식)

- 벨만 기대 방정식: 가치함수가 다음 상태로 이동하기 위해 정책(policy)를 고려하여 다음 상태로 이동하는 것을 의미한다.

$$v_\pi(s) = E_\pi[G_t | S_t = s]$$ ---------- 현재 시점에서 미래에 기대되는 보상들의 총합을 기댓값으로 표현

현재 상태의 가치(G_t)

$$= E_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots | S_t = s]$$ ---------- 할인율을 적용하여 풀어 쓴다.

$$= E_\pi[R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \cdots) | S_t = s]$$ ------- 할인율을 기준으로 묶어준다.

다음 상태의 가치(G_{t+1})

$$= E_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s]$$

$$= E_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s]$$ ---------- $v_\pi(s) = G_t$ 라고 했으므로 다음 상태의 가치는 $v_\pi(s_{t+1}) = G_{t+1}$로 표현

→ 현재 시점의 가치는 현재의 보상과 다음 시점의 가치로 표현할 수 있다.
→ 즉, 재귀적인 형태로서 미래의 가치들이 현재의 가치에 영향을 주고 있는 형태이다.

3.1 벨만 기대 방정식

<상태-가치 함수 $v_\pi(s)$>

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

------- $R_{t+1}, \gamma G_{t+1}$과 기댓값을 분리하여 생각한다.

기댓값: 기댓값은 현재 상태에서 정책에 따라 행동했을 때 얻을 수 있는 각각의 행동($\pi(a|s)$)과 그 행동이 발생할 확률($p(s', r|s,a)$)을 곱한 것, 현재 기대되는 결과에 그 결과가 일어날 확률로 가중치가 곱해진 것과 같은 의미이다.

R_{t+1}: 현재 행동을 선택했을 때 즉각적으로 얻어지는 보상

γG_{t+1}: 더 미래에 일어날 보상에 할인율이 곱해진 것, 미래에 일어날 모든 일의 평균치와 같은 의미이다.

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

3.1 벨만 기대 방정식

<상태-가치 함수 $v_\pi(s)$>

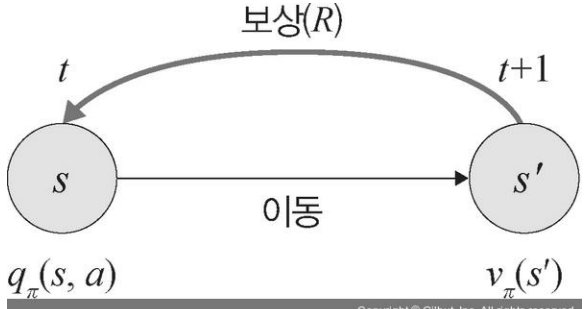
$$v_\pi(s) = E_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s]$$

$$= \sum_{a \in A} \pi(a|s) \sum_{s',r} P(s', r | s, a)[r + \gamma E_\pi[G_{t+1} | S_{t+1} = s']]$$ ------- 다음 상태인 s′ 의 상태-가치 함수

$$= \sum_{a \in A} \pi(a|s) \sum_{s',r} P(s', r | s, a)[r + \gamma v_\pi(s')]$$ ----③

$$= \sum_{a \in A} \pi(a|s) q_\pi(s,a)$$ ----④

④의 식에서 $q_\pi(s,a)$를 풀어써보면..
상태 s에서 행동 a를 했을 때의 행동에 대한 가치는 두 가지 요소로 구성되어 있는데, 상태 s에서 행동 a를 했을 때의 보상과 그 다음 상태의 가치 함수로 구성된다.
($v_\pi(s')$)는 t+1 시점에서의 가치 함수이므로 할인율과 현재(t) 상태 s에서 다음(t+1) 상태 s'로 전이될 확률도 적용해 준다.



$$q_\pi(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_\pi(s')$$ ----⑤ $$v_\pi(s) = \sum_{a \in A} \pi(a|s)(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_\pi(s'))$$

3.1 벨만 기대 방정식

<행동-가치 함수 $q_\pi(s,a)$>

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

$$
\mathrm{R}_s^a = E_\pi[R_{t+1} \mid S_t = s,\ A_t = a]
$$

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

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

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

적용

$$
q_\pi(s,a) = \mathrm{R}_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \sum_{a' \in A} \pi(a' \mid s') q_\pi(s',a')
$$

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

$$
q_\pi(s,a) = \mathrm{R}_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \sum_{a' \in A} \pi(a' \mid s') q_\pi(s',a')
$$

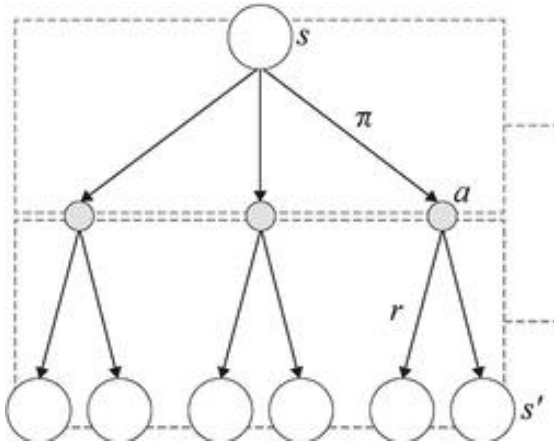
상태-가치 함수, 행동-가치 함수를 사용하여 현재와 바로 다음 상태, 그리고 행동 간 관계가 드러난다.

EWHA EURON

3.1 벨만 기대 방정식

<백업 다이어그램>

상태-가치 함수와 행동-가치 함수의 벨만 방정식을 다이어그램으로 표현한 것.



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

------- 뒤따를 행동-가치 함수의 정책 기반 가중 평균

$$q_\pi(s,a) = \mathrm{R}_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \sum_{a' \in A} \pi(a' \mid s') q_\pi(s', a')$$

------- 다음 상태-가치 함수에 대한 보상과 상태 전이 확률에 대한 결합 확률의 가중 평균

< 강화 학습 과정>

1. 처음 에이전트가 접하는 상태 s나 행동 a는 임의의 값으로 설정한다.
2. 환경과 상호 작용하면서 어떤 행동을 취하면 최대의 보상을 얻을 수 있는지 판단한다.
3. 최적의 행동을 판단하는 수단: 상태-가치 함수와 행동-가치 함수이다. 벨만 기대 방정식을 이용하여 업데이트하면서 점점 높은 보상을 얻을 수 있는 상태와 행동을 학습한다.
4. 2~3의 과정 속에서 최대 보상을 갖는 행동들을 선택하도록 최적화된 정책을 찾는다.

3.2 벨만 최적 방정식

강화학습의 목표: 최대 보상을 얻는 정책을 찾는 것

최적화된 정책과 이것을 따르는 벨만 최적 방정식

상태-가치 함수: 어떤 상태가 더 많은 보상을 받을 수 있는지 알려 준다.
행동-가치 함수: 어떤 상태에서 어떤 행동을 취해야 더 많은 보상을 받을 수 있는지 알려 준다.

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

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

$$\pi_*(a \mid s) = \begin{cases} 1 & a = \arg\max_a q_*(s,a) \text{ 일 때} \\ 0 & \text{그 외} \end{cases}$$

행동-가치 함수를 최대로 하는 행동만 취하겠다는 의미이다. (그 외의 지점에서는 모두 0으로 처리한다.)

3.3 다이나믹 프로그래밍

다이나믹 프로그래밍: 연속적으로 발생되는 문제를 수학적으로 최적화하여 풀어내는 것을 의미한다.

가정: MDP의 모든 상황에 대해 알고 있다고 가정한다. -> 계획이 가능하다.

MDP와 정책을 입력으로 하여 가치 함수를 찾아내는 것이 예측(prediction) 과정이다.

MDP를 입력으로 하여 기존 가치 함수를 더욱 최적화하는 것이 컨트롤(control) 과정이다.

<정책 이터레이션>

정책 평가와 정책 발전을 반복하여 특정 값으로 수렴하게 한다. -> 그때가 최적화된 정책과 가치이다.

- **정책 평가**

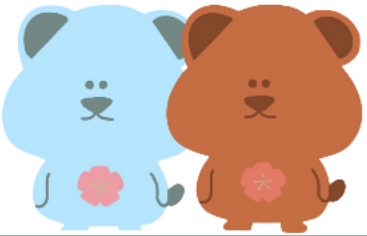
모든 상태에 대해 그 다음 상태가 될 수 있는 행동에 대한 보상의 합을 저장하는 것이 정책 평가 이다.

- **정책 발전**

욕심쟁이 정책 발전: 에이전트가 할 수 있는 행동들의 행동-가치 함수를 비교하고 가장 큰 함수 값을 가진 행동을 취하는 것이다. 따라서 업데이트 이후, 가치 함수는 무조건 크거나 같고, 장기적으로 최적화된 정책에 수렴할 수 있다.

참고: https://cding.tistory.com/53

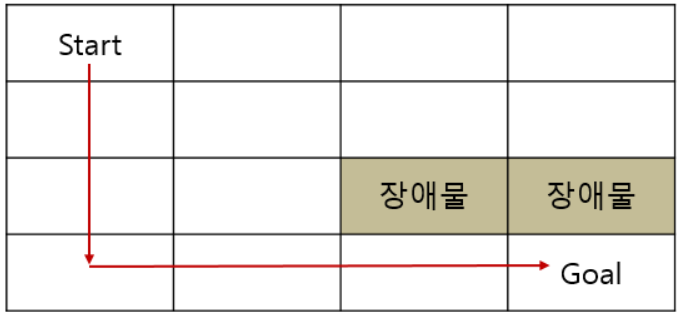
04. 큐러닝



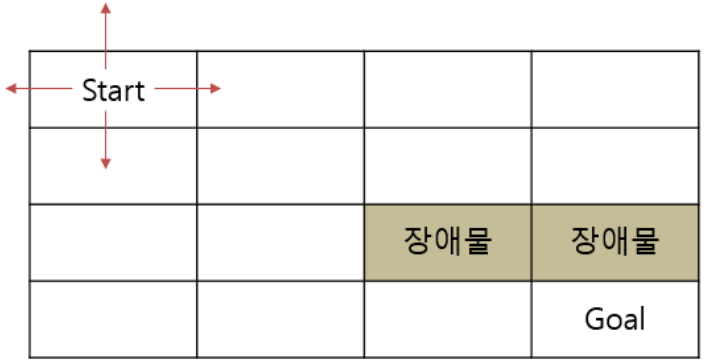
EWHA EURON

4.1. 큐러닝

큐러닝 (Q-learning)이란 강화학습 기법 중 하나로,
행동에 따른 보상의 기댓값을 예측하는 큐함수 (Q-function) 을 사용하여 최적화된 정책을 찾는다.



인간



컴퓨터

▼ 표 12-1 큐-러닝의 큐-테이블

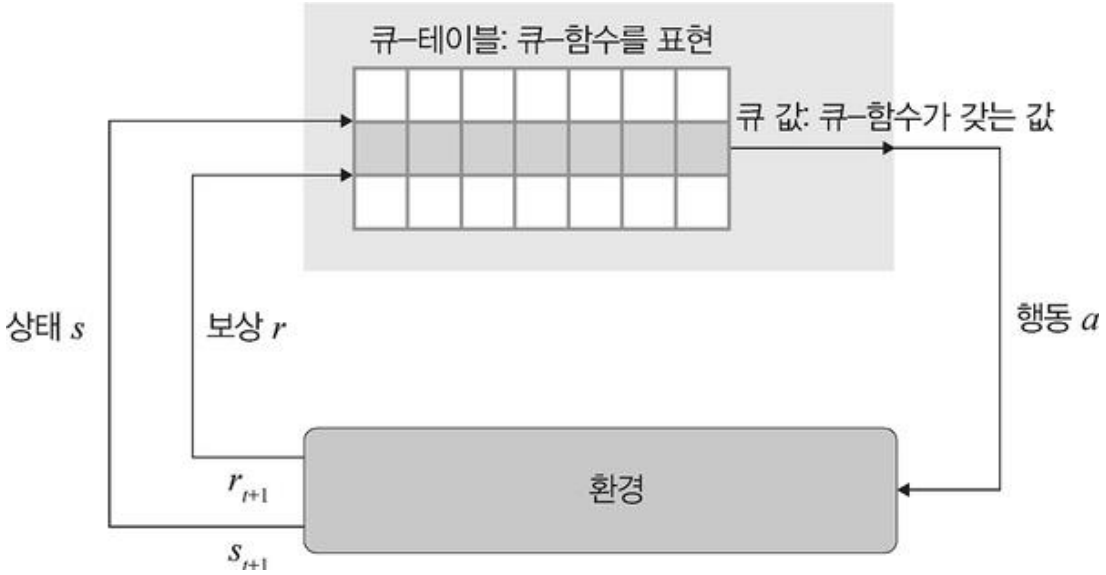
큐-테이블		행동				
		$Action_1$	$Action_2$	···	$Action_{n-1}$	$Action_n$
상태	$State_1$	0	0	···	0	0
	$State_2$	0	0	···	0	0
	···	···	···	···	···	···
	$State_{n-1}$	0	0	···	0	0
	$State_n$	0	0	···	0	0

단점 ---> 큐러닝 기반의 DQN 등장
★ 취할 수 있는 상태 개수가 많은 경우 큐 테이블 구축에 있어 한계
★ 데이터 간 상관관계로 학습이 어려움

4.1. 큐러닝

1. 현재의 위치를 상태(state)라고 정의한다
2. 랜덤으로 새로운 시도, 행동(action)을 취한다. 탐험한다.
3. 좋은 행동이었는지 아닌지 판단한다 보상한다(reward)
4. ~학습~
5. 그 행동을 취했을 때 얼마나 좋은지를 측정하는 값(Q-value) 이 학습한다
6. Q-value가 가장 높은, 최대의 보상을 받을 수 있는 행동을 다음행동으로 취한다 활용한다.
7. 먼 미래에 더 좋을 것이라는 피드백을 주기 위해 Discount factor를 고려한다.
8. 큐-값 업데이트한다.

4.1. 큐러닝



★Greedy!

★ 0~1사이 난수를 추출해 임계치보다 낮으면 랜덤하게 행동을 취한다. 임계치는 학습이 진행되면서 점점 낮은 값을 가지게 되고 나중에는 0수렴한다.

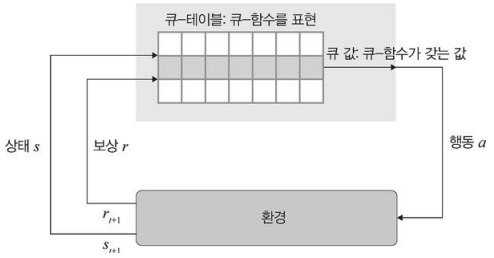
★ 행동-> 보상 -> 큐 업데이트 $Q(s,a) = r(s,a) + \gamma \max_a Q(s',a)$

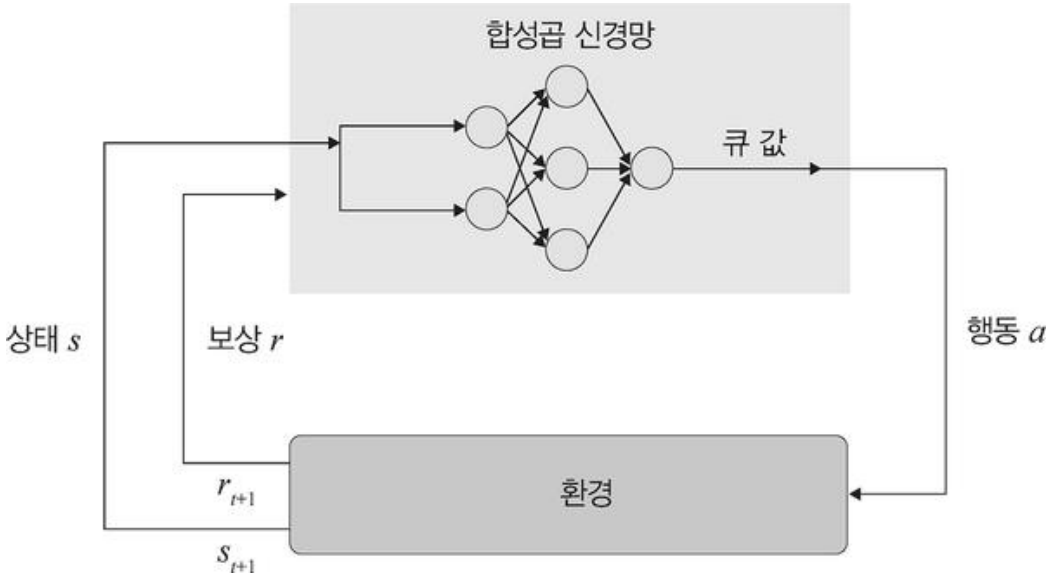
```
while not done :
   #행동 중 가장 보상(r)이 큰 행동을 고르고, 랜덤 노이즈 방식으로 활용과 탐험 구현
    action = np.argmax(Q[state, :] + np.random.randn(1, env.action_space.n) / (i+1))

    # 해당 행동을 했을 때 환경이 변하고, 새로운 상태(state), 보상(reward), 완료(done) 여부를 반환
    new_state, reward, done, _ = env.step(action)

    # Q = R + Q
    Q[state, action] = reward + dis * np.max(Q[new_state, :])
    rAll += reward
    state = new_state
```

4.2. 딥큐러닝

딥큐러닝 (Deep Q Learning, DQL)이란 큐-테이블대신 신경망을 사용하여 큐-함수를 학습하는 강화 학습 기법으로, 큐 값의 정확도를 높이는 것을 목표로한다.

★ 취할 수 있는 상태 개수가 많은 경우 큐 테이블 구축에 있어 한계 -> 신경망으로 해결





4.2. 딥큐러닝

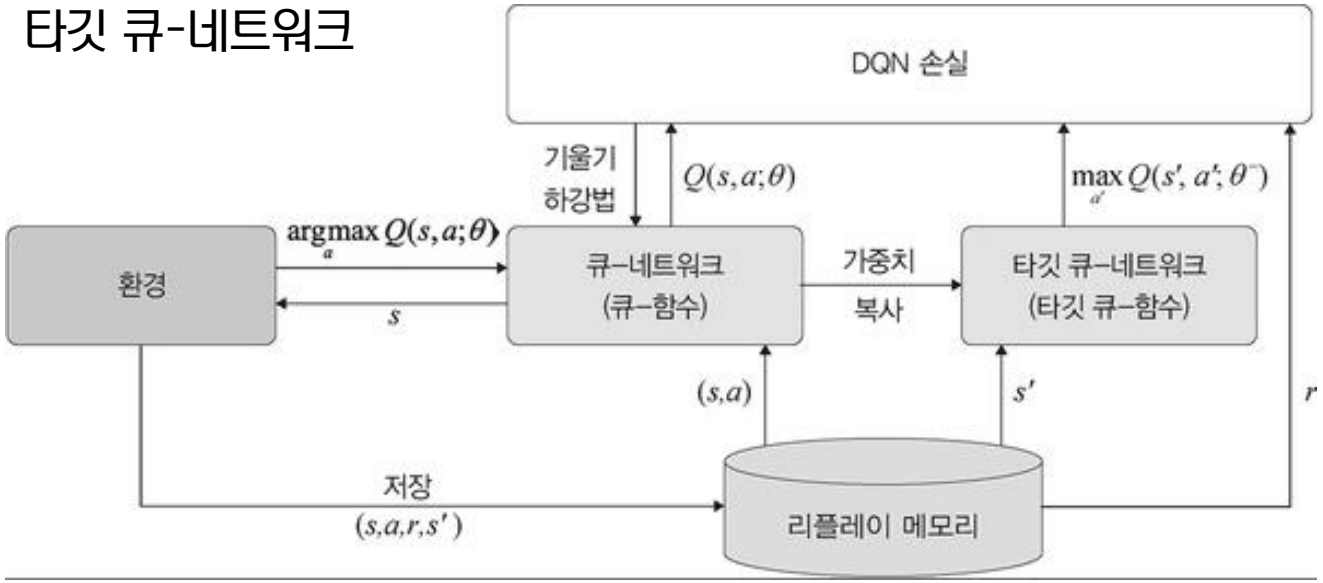
강화학습을 위한 시뮬레이션 환경을 제공

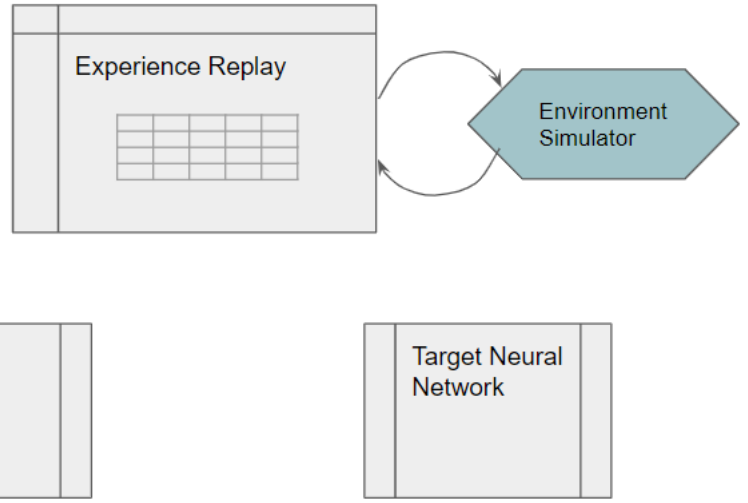
함수	설명
reset()	환경을 초기화할 때 사용합니다. 에이전트가 게임을 시작하거나 초기화가 필요할 때 reset() 함수를 사용하며, 초기화될 때는 관찰 변수(상태를 관찰하고 그 정보를 저장)를 함께 반환합니다.
step()	에이전트에 명령을 내리는 함수입니다. 따라서 가장 많이 호출되는 함수로, 이 함수로 행동 명령을 보내고 환경에서 관찰 변수, 보상 및 게임 종료 여부 등 변수를 반환합니다.
render()	화면에 상태를 표시하는 역할을 합니다.

강화 학습으로 AI 에이전트를 훈련할 때

4.2. 딥큐러닝

타깃 큐-네트워크





큐-러닝에서는 학습되면서 큐 값이 계속 바뀌는 문제가 있다.
딥 큐-러닝에서는 이를 해결하기 위해, 별도의 네트워크인, 타깃 큐-네트워크 (target Q-network)를 사용한다.
큐 네크워크, 타깃 큐-네트워크는 가중치 파라미터만 다르고 같다.
DQN에서는 수렴을 원활히 하기위해 타깃 큐-네트워크를 주기적으로 업데이트한다.

Loss function = MSE

$$Cost = \left[Q(s,a;\theta) - \left(r(s,a) + \gamma \max_{a} Q(s',a;\theta)\right)\right]^2$$

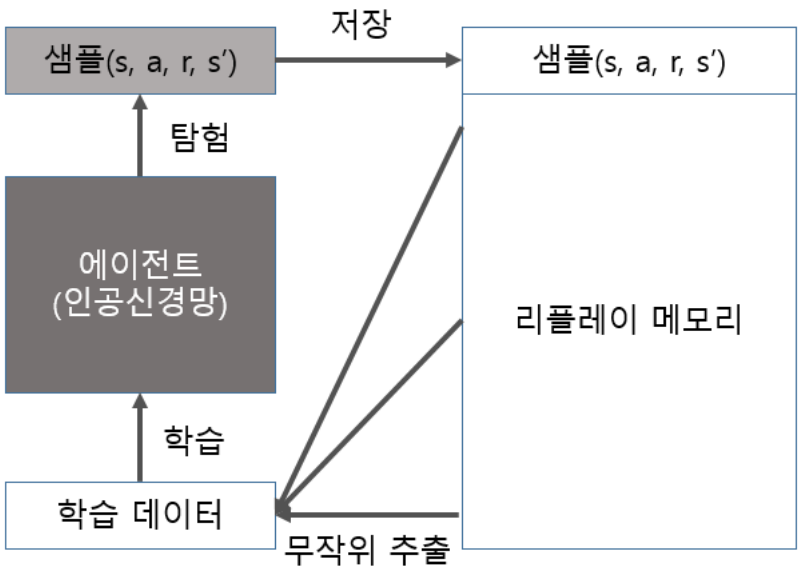
$$L_i(\theta_i) = E_{(s,a,r,s')\sim U(D)}[(r + \gamma \max_{a'} Q(s',a';\theta_i^-) - Q(s,a;\theta_i))^2]$$

4.2. 딥큐러닝

리플레이 메모리

★ 데이터 간 상관관계로 학습이 어려움 -> 리플레이 메모리로 해결

리플레이 메모리는 에이전트가 수집한 (탐험) 데이터를 저장해 두는 저장소이다.



에이전트 상태가 변경되어도 즉시 훈련시키지 않고 일정 수의 데이터가 수집되는 동안 기다린다.
나중에 일정 수의 데이터가 메모리(버퍼)에 쌓이게 되면 랜덤하게 데이터를 추출하여 미니 배치를 활용해서 학습한다.

데이터 여러 개로 훈련을 수행한 결과들을 수렴하여 결과를 도출하기 때문에 상관관계 문제 해결

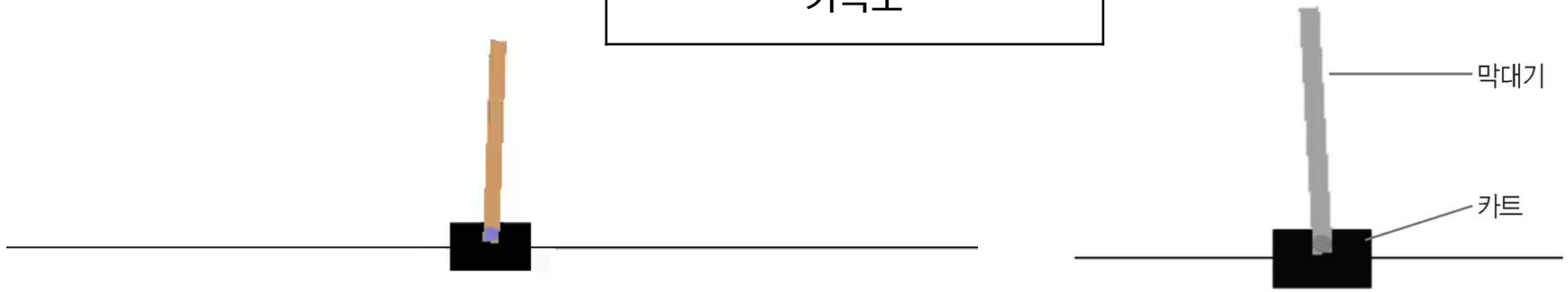
4.2. 딥큐러닝

https://tave-6th-rlstudy.github.io/2020/09/07/Reinforcement_Learning6.html

합성곱 신경망을 활용한 큐-함수

위치	왼쪽
	오른쪽
속도	
가속도	

왼쪽
오른쪽



Cartpole : 에이전트는 카트에 부착된 막대기가 수직 상태 (s)를 유지할 수 있도록 카트를 왼쪽/오른쪽 (a) 으로 이동하는 작업을 반복한다.

보상(r) ex. 카트폴이 쓰러지지 않고 버티는 시간이 10초이상이면 +10, 도중 쓰러지면 -100

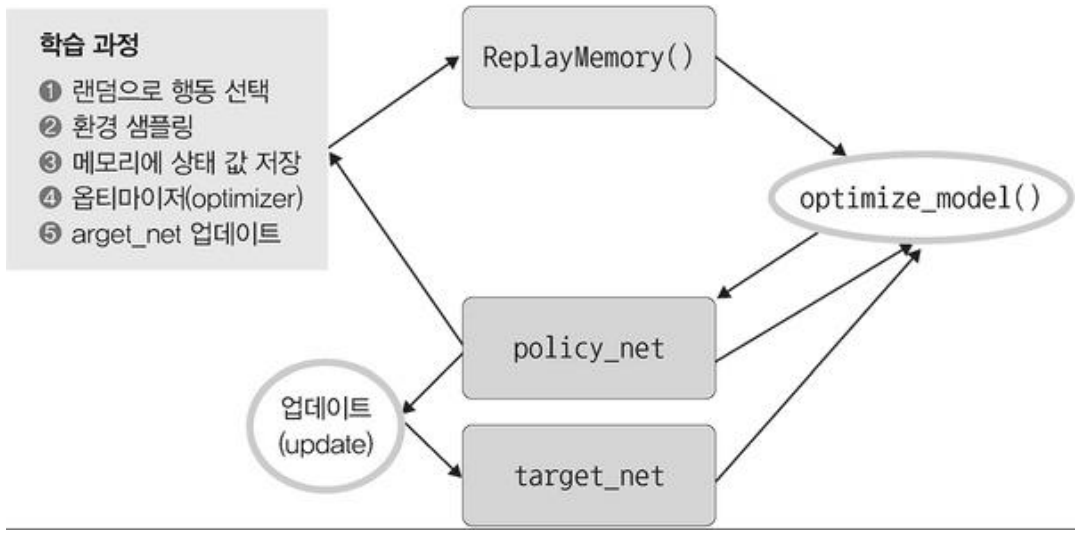
4.2. 딥큐러닝

```
#리플레이 메모리

Transition = namedtuple('Transition',
                        ('state', 'action', 'next_state', 'reward'))

class ReplayMemory(object):
    def __init__(self, capacity):
        self.capacity = capacity
        self.memory = []
        self.position = 0

    def push(self, *args):
        if len(self.memory) < self.capacity:
            self.memory.append(None)
        self.memory[self.position] = Transition(*args)
        self.position = (self.position + 1) % self.capacity

    def sample(self, batch_size):
        return random.sample(self.memory, batch_size)

    def __len__(self):
        return len(self.memory)
```

4.2. 딥큐러닝

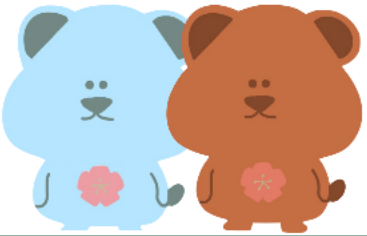
```
# DQN network

class DQN(nn.Module):
    def __init__(self, h, w, outputs):
        super(DQN, self).__init__()
        self.conv1 = nn.Conv2d(3, 16, kernel_size=5, stride=2)
        self.bn1 = nn.BatchNorm2d(16)
        self.conv2 = nn.Conv2d(16, 32, kernel_size=5, stride=2)
        self.bn2 = nn.BatchNorm2d(32)
        self.conv3 = nn.Conv2d(32, 32, kernel_size=5, stride=2)
        self.bn3 = nn.BatchNorm2d(32)

        def conv2d_size_out(size, kernel_size = 5, stride = 2):
            return (size - (kernel_size - 1) - 1) // stride  + 1

        convw = conv2d_size_out(conv2d_size_out(conv2d_size_out(w)))
        convh = conv2d_size_out(conv2d_size_out(conv2d_size_out(h)))
        linear_input_size = convw * convh * 32
        self.head = nn.Linear(linear_input_size, outputs)

    def forward(self, x):
        x = F.relu(self.bn1(self.conv1(x)))
        x = F.relu(self.bn2(self.conv2(x)))
        x = F.relu(self.bn3(self.conv3(x)))
        return self.head(x.view(x.size(0), -1))
```

4.2. 딥큐러닝



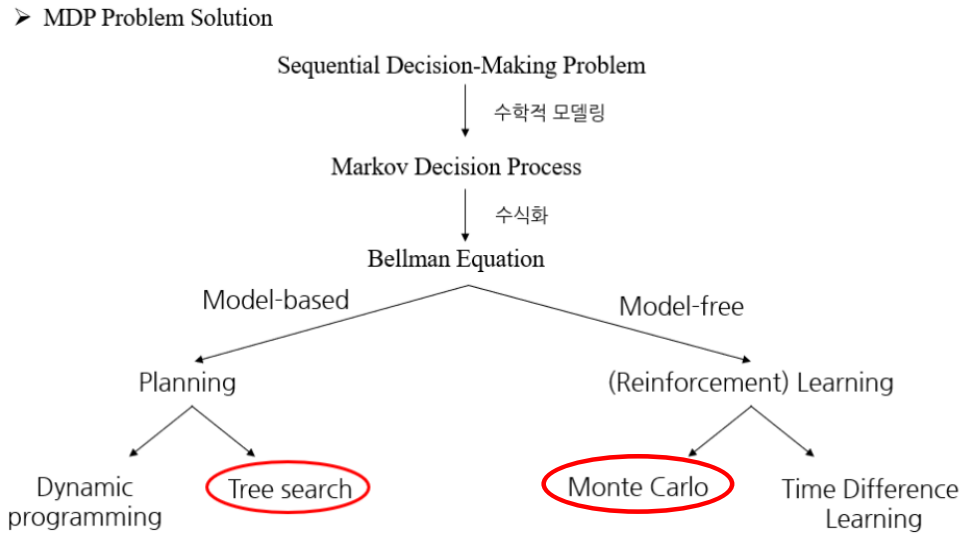

```python
#옵티마이저
#리플레이 메모리에서 무작위 데이터를 선택하여 새로운 정책을 학습.
#마지막에 최신상태를 유지하기 위해 target_net에 가중치 및 바이어스를 업데이트.


def optimize_model():
    if len(memory) < BATCH_SIZE:
        return

    transitions = memory.sample(BATCH_SIZE)
    batch = Transition(*zip(*transitions))
    non_final_mask = torch.tensor(tuple(map(lambda s: s is not None,
                                            batch.next_state)), device=device, dtype=torch.bool)
    non_final_next_states = torch.cat([s for s in batch.next_state
                                                if s is not None])
    state_batch = torch.cat(batch.state)
    action_batch = torch.cat(batch.action)
    reward_batch = torch.cat(batch.reward)

    state_action_values = policy_net(state_batch).gather(1, action_batch)
    next_state_values = torch.zeros(BATCH_SIZE, device=device)
    next_state_values[non_final_mask] = target_net(non_final_next_states).max(1)[0].detach()
    expected_state_action_values = (next_state_values * GAMMA) + reward_batch

    loss = F.smooth_l1_loss(state_action_values, expected_state_action_values.unsqueeze(1))
    optimizer.zero_grad()
    loss.backward()

    for param in policy_net.parameters():
        param.grad.data.clamp_(-1, 1)
    optimizer.step()
```

05. 몬테카를로 트리탐색



EWHA EURON

5. 몬테카를로 트리탐색

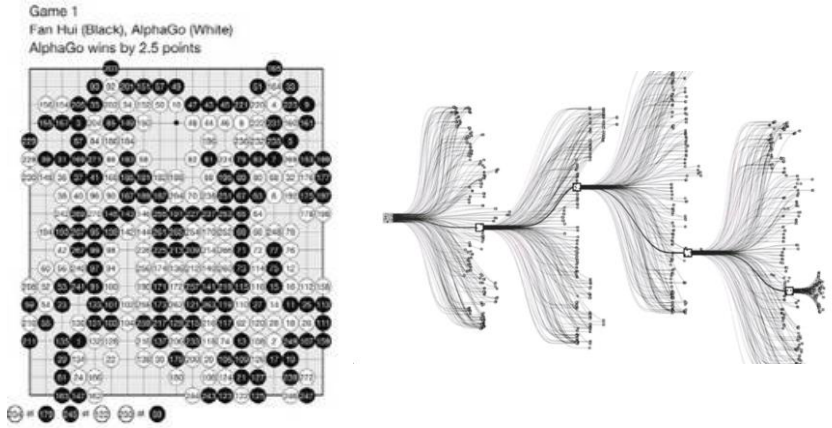


Tree Search : 현재 상황에서 가능한 모든 경우의 수들을 tree형태로 뻗어나가며 좋은 수인지 판단한 후 가장 좋은 수를 선택

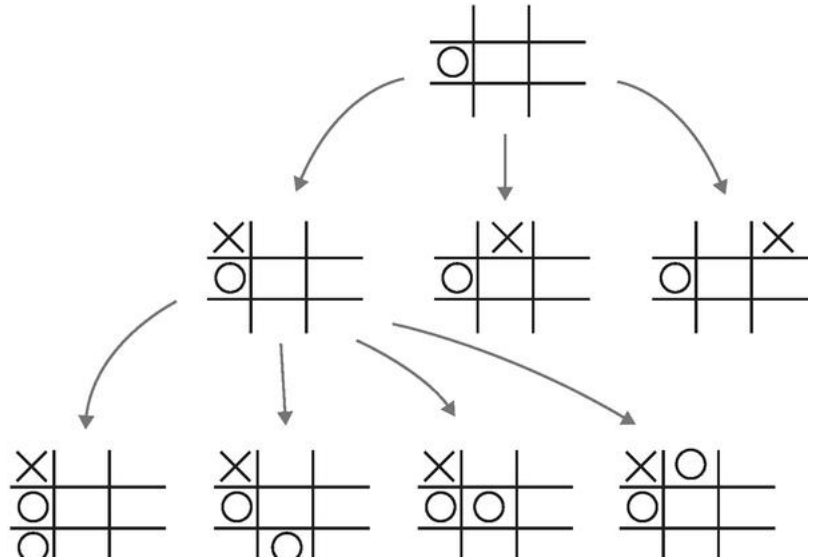
Monte Carlo : 정확한 확률 분포를 구하기 어려울 때 무작위 표본 추출을 통해 확률 분포를 도출하는 것

Monte Carlo + Tree Search = Monte Carlo Tree Search

5. 몬테카를로 트리탐색



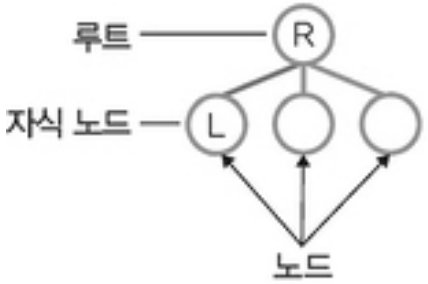
Ex.알파고
다양한 경우의 수를 고려해야 할 때 사용되는 기법

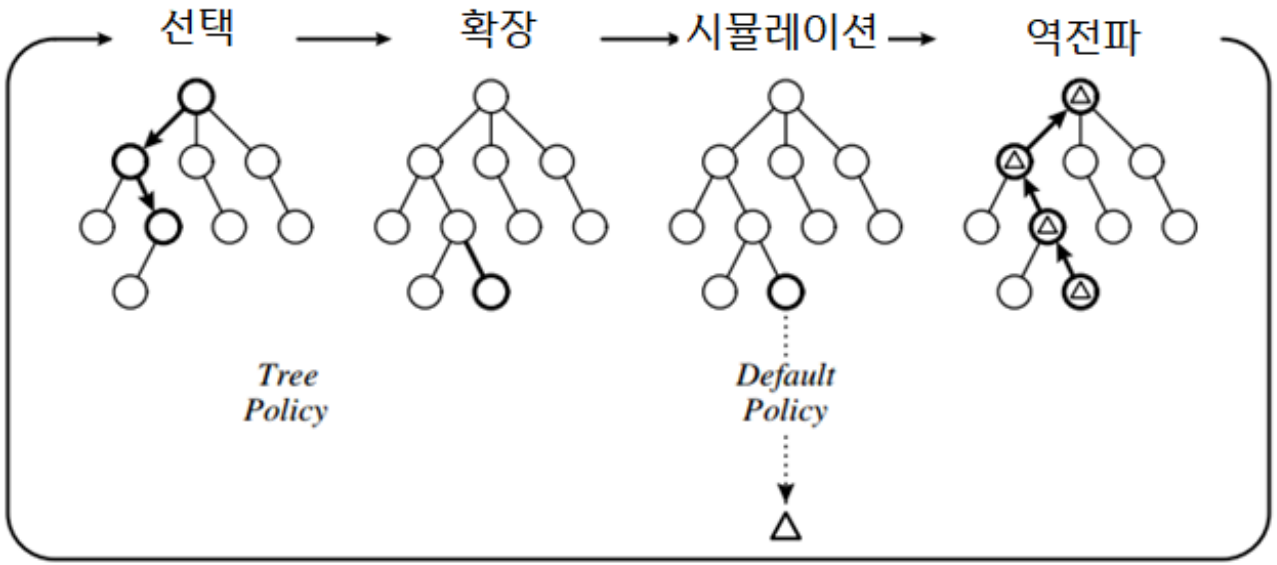


몬테카를로 트리탐색이란 경우의 수를 순차적으로 시도하는 것이 아닌 무작위 방법 중 가장 승률이 높은 값을 기반으로 시도하는 것.

랜덤 시뮬레이션을 이용하여 최적의 선택을 결정.

5. 몬테카를로 트리탐색



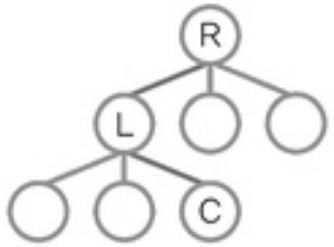


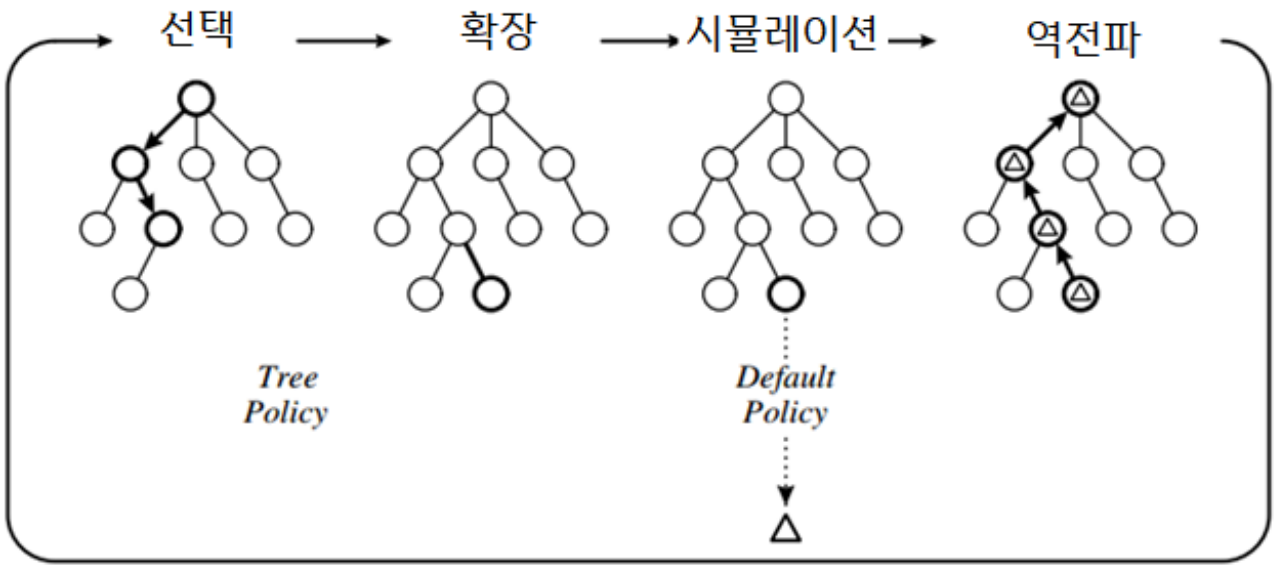
1. 선택

루트(현재게임상태) 에서 시작하여 현재 가장 승산 있는 자식 노드(아직 시뮬레이션 되지 않은) L을 선택한다.

Tree policy : 선택 단계에서 확장을 이어 나가기 위해 선택할 child node를 정할 때 사용하는 정책으로, child node를 선택할 때 아무 node나 고르는 것이 아니고 적절한 tree policy에 따라 선택을 해야 합니다.

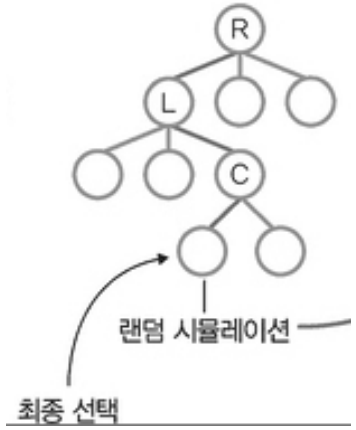
5. 몬테카를로 트리탐색

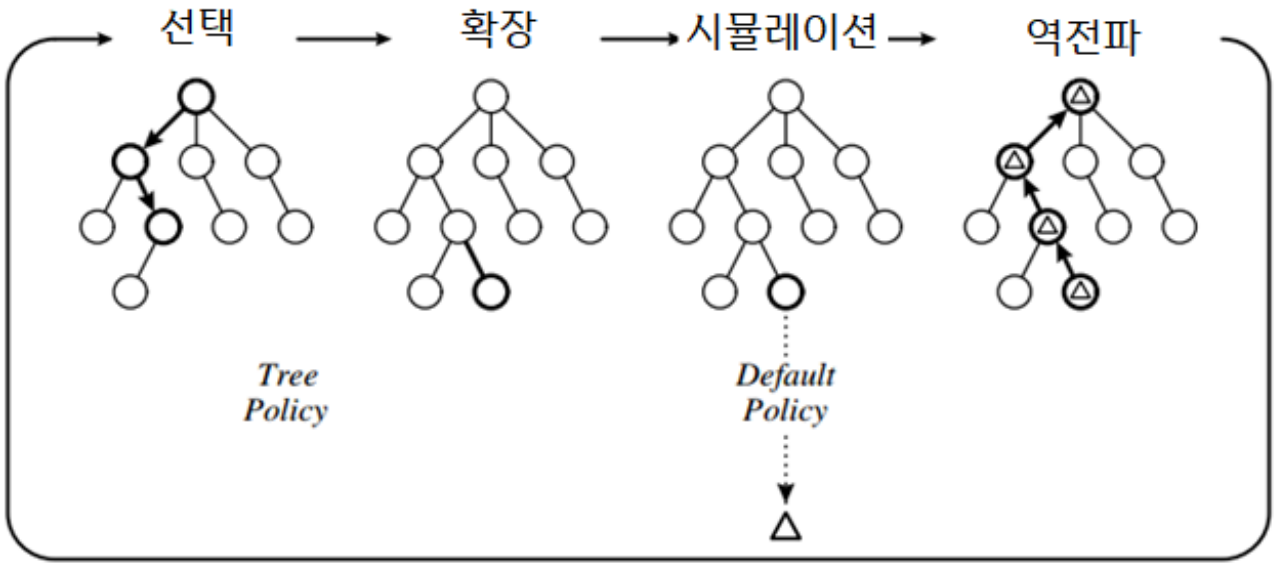




2. 확장
노드 L에서 게임이 종료되지 않는다면 하나 또는 그 이상의 자식 노드를 생성하고
그 중 하나의 노드 C를 선택한다.

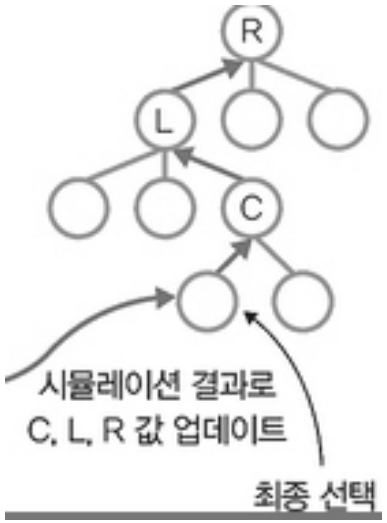
5. 몬테카를로 트리탐색

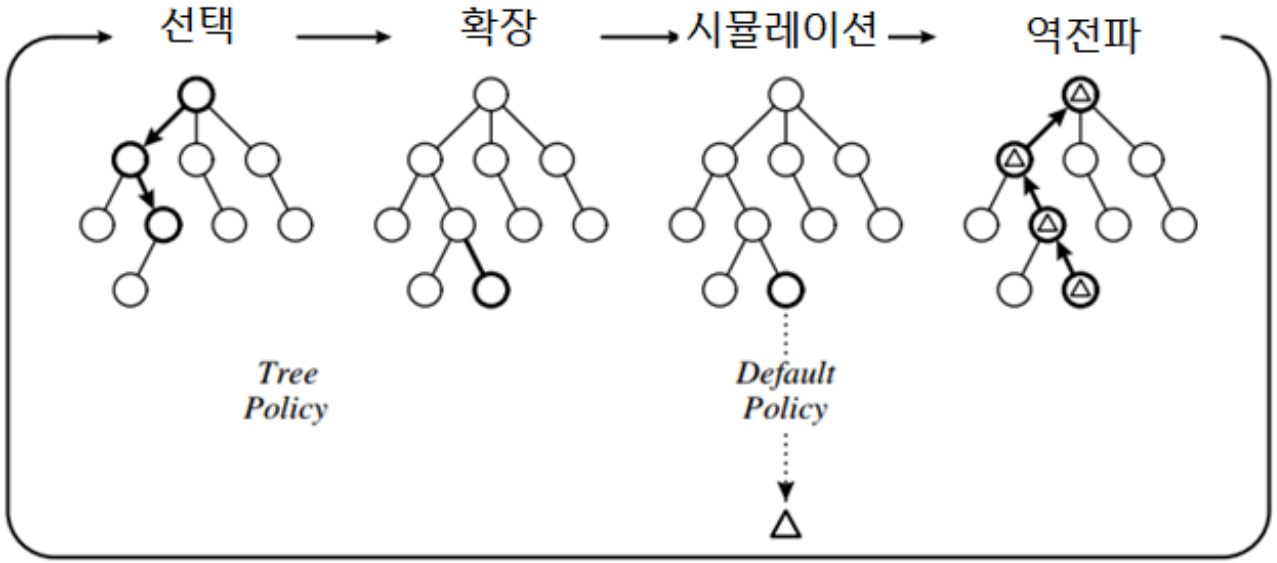




3. 시뮬레이션
노드 C에서 랜덤으로 자식 노드를 선택하여 이를 통해 게임이 종료될때까지 반복 진행한다.

5. 몬테카를로 트리탐색





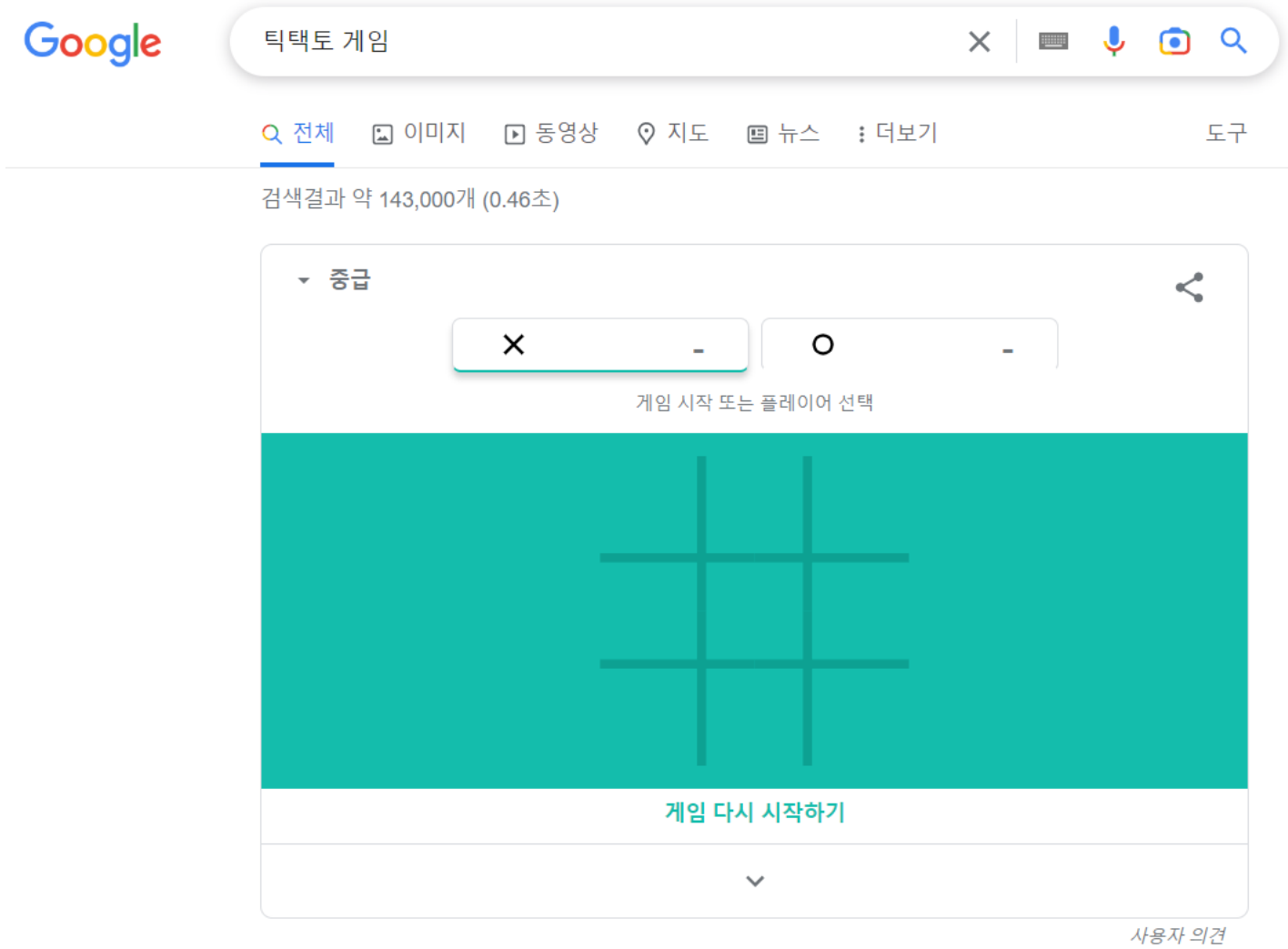
4. 역전파
시뮬레이션 결과로 C, L, R까지 경로에 있는 노드들(선택된 자식노드)의 정보를 갱신한다

https://www.youtube.com/watch?v=QH9h-XSi5nw
6m50s

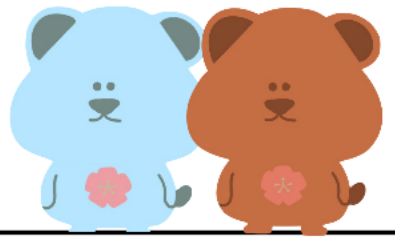
5. 몬테카를로 트리탐색

틱택토 (tic-tac-toe/ 오목)

https://diy-project.tistory.com/27
C++
https://ai-creator.tistory.com/528
python



THANK YOU



EWHA EURON