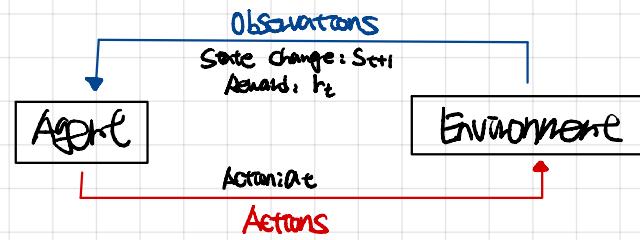


Reinforcement Learning

- Data: State-action pairs
- Goal: Maximize future rewards over many time steps
- Terminology

1. Agent: 행동을 하는 주체
2. Environment: Agent가 조작하고, 반응하는 환경
3. Actions: Agent가 취할 수 있는 행동
4. Action space: 가능한 action의 집합
5. Observation: Action을 Environment에 대한 상태
6. State: Agent가 알 수 있는 현재 상태
7. Reward: Action의 결과로 드는 보상



$$\text{Total Reward: } R_t = \sum_{i=t}^{\infty} r_i = r_t + r_{t+1} + \dots + r_\infty$$

$$\text{Discounted Total Reward: } R_t = \sum_{i=t}^{\infty} \gamma^i r_i$$

미래의 reward의 영향 ↓
→ 초기 보상이 더 중요함.
 $0 < \gamma < 1$

Defining the Q-function

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots$$

Total reward, R_t , is the discounted sum of all rewards obtained from time t

미래의 계획을 고려한 보상

$$Q(s_t, a_t) = \mathbb{E}[R_t | s_t, a_t] \quad \leftarrow \begin{array}{l} \text{St, At가 주어졌을 때} \\ \text{분명 Reward가 도출될} \end{array}$$

The Q-function captures the **expected total future reward** an agent in state, s , can receive by executing a certain action, a

$$Q(s, a)$$

s', a' : next state parameter

$$= h(s, a) + \gamma \mathbb{E}_s [\max_a Q(s', a')]$$

다른 상태에서
최대의 상승률이
증가 가능함.

(미래보상 증가)

Ultimately, the agent needs a **policy $\pi(s)$** , to infer the **best action to take** at its state, s

Strategy: the policy should choose an action that maximizes future reward

$$\pi^*(s) = \operatorname{argmax}_a Q(s, a) \quad \leftarrow \begin{array}{l} \text{State가 주어졌을 때, Q function의 가장 크게 드러난} \\ \text{action은 action space에서 찾을 수 있다.} \end{array}$$

QH Future Total Reward 인가?

→ 강화학습의 목표가 지금 이 상태에서 어떤 행동을 해야, 미래에 보상을
최대화할 수 있는지를 고민하는 것입니다.

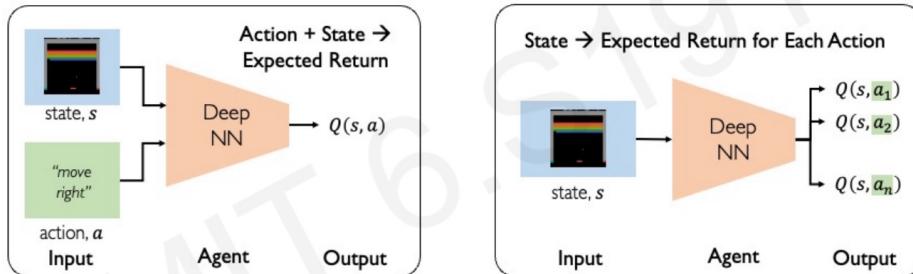
Deep Reinforcement Learning Algorithm

1. Value Learning : Find $Q(s, a)$ and $a = \operatorname{argmax}_a Q(s, a)$
→ Q function의 Estimate

2. Policy Learning : Find $\pi(s)$ and Sample $a \sim \pi(s)$
→ 주어진 상황에 어떤 주변 환경 정의

1. Deep Q Networks (DQN)

How can we use deep neural networks to model Q-functions?



- Q -function을 통해
위해서 사용된다.

$$\pi(s) = \text{argmax}_a Q(s, a)$$

↳ Optimal Policy: 어떤 행동
선택하는 정책.

- Action space가 큰 경우, Q 를
제공하기 어렵거나 비효율적이다.

=> Q -function을 이용해
optimal action을 찾는다.

$$\mathcal{L} = \mathbb{E} \left[\left\| \underbrace{(r + \gamma \max_{a'} Q(s', a'))}_{\text{target}} - \underbrace{Q(s, a)}_{\text{predicted}} \right\|^2 \right] \quad \text{Q-Loss}$$

↳ 현재 상태의 예측값

해당 행동을 했을 때
이미지는 최적의 상황이 예상

[단점]

- Action Space is discrete (Not continuous)

→ 연속적인 $Q(s, a)$ 를 의미하기 어렵다.

- Policy가 Q -function을 위해서 이미 결정되어 흐름성이 있다.

Discrete Action Spaces

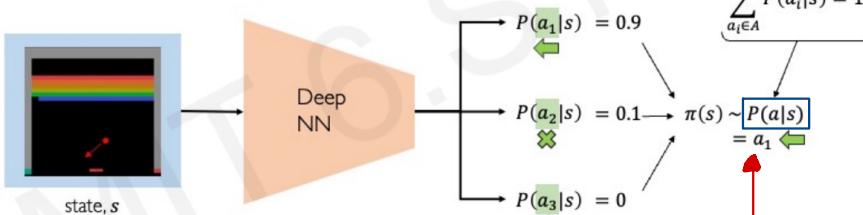
- 이진 방향을 이용한다

Continuous Action Spaces

- 주로 조정

2. Policy Gradient (PG)

Policy Gradient: Directly optimize the policy $\pi(s)$



$$\rightarrow P(a|s) = \begin{bmatrix} 0.9 \\ 0.1 \\ 0 \end{bmatrix}, \quad \text{If Continuous, } P(a|s) = N(\mu(s), \sigma^2(s))$$

- State만으로도
최적의 Action을 찾는다.

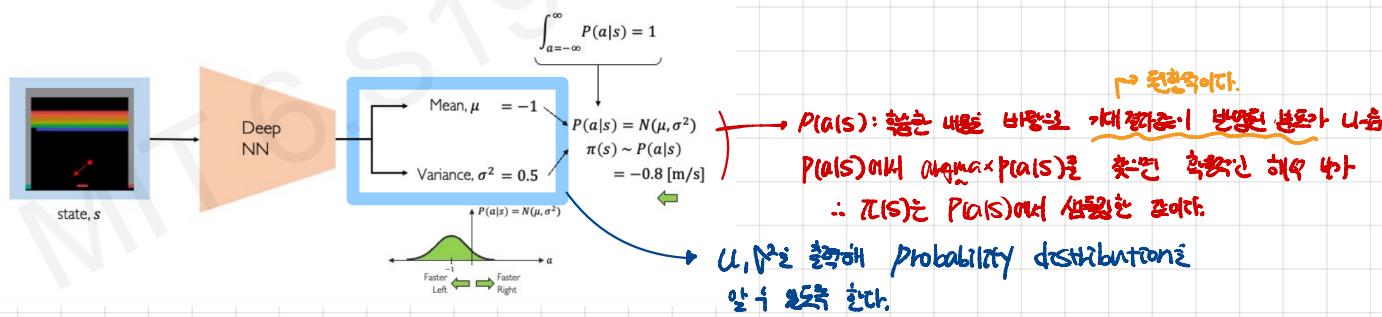
→ Policy $\pi(s)$ 를 최적화

학습 분포: Continuous action space에서 적용 가능

→ 특히 정수 Discrete action = 예상되는 최적의 속도 같은 연속적인 행동의
경우 PG가 더 효율적이다.

Continuous Case

Policy Gradient: Enables modeling of continuous action space

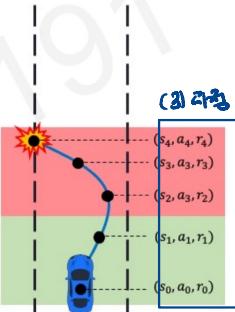


Traching Policy Gradients

Training Algorithm

1. Initialize the agent
2. Run a policy until termination
3. Record all states, actions, rewards
4. Decrease probability of actions that resulted in low reward
5. Increase probability of actions that resulted in high reward

반복



⇒ 이 모든 정보도 우리 앤드로이드 휴대폰에서 reward를 기록
기준으로 조작으로 쓰고 있습니다.

← Low reward : 훈련 차선 험

← High reward : 정상주행 험

-log P(action)

$-\log P(\text{action}) \rightarrow P(\text{action})$

$\rightarrow P(\text{action}) \uparrow \rightarrow \text{경사} \downarrow = \text{Loss가 감소(정답)}$

$\rightarrow P(\text{action}) \downarrow \rightarrow \text{경사} \uparrow = \text{Loss가 증가(오답)}$

⇒ 가중치($P(\text{action})$)가 높은 것의 Reward가 크고, 높은 것의 Reward가 낮은 것의 Loss를 통해 학습합니다!

Gradient descent update

$$w' = w - \alpha / \text{LOSS}$$

$$w' = w + \nabla \log P(\text{action}) R_t$$

단점 : 실제 시나리오에서 훈련하기 어렵다면 유전성이 보장되지 않는다.

Sol) Simulation only 훈련

Example of Reinforcement Learning: AlphaGo

