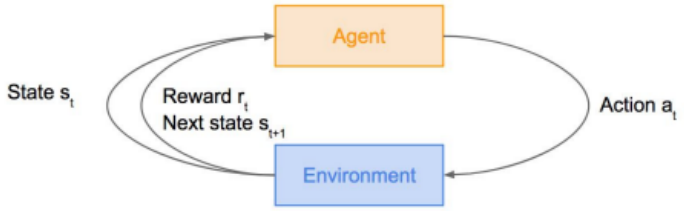
Reinforcement Learning

: Machine Learning의 한 범주로서 어떤 환경 안에서 정의된 에이전트가 현재의 상태를 인식하여, 선택 가능한 행동들 중 보상을 최대화하는 행동 혹은 행동 순서를 선택하는 방법



Agent : Environment에서 Action을 취할 수 있는 물체. 어떠한 Action에 의해 Reward를 최대로 받는 것이 목적
Environment : Agent와 상호작용하는 것. Agent에게 적절한 state 부여

1. Environment로부터 Agent가 State를 부여받음
2. 상태를 받은 Agent가 어떠한 Action을 취함
3. 그 Action에 대해 Agent는 보상 받음
4. 다음 State를 부여받음

Markov Decision Process

: 강화 학습 방법을 수식화.
Markov Property : 현재 상태로 전체 상태를 나타내는 것

S: 가능한 상태 집합
A: 가능한 액션 집합
R : (상태, 액션)의 쌍이 주어졌을 때 보상에 대한 분포
P: (상태, 액션)의 쌍이 주어졌을 때 다음 상태에 대한 분포 (전이 확률)
R : 보상을 받는 시간에 대해서 얼마나 중요하게 생각하는지

A simple MDP: Grid World

Random Policy　　Optimal Policy

*쪽으로 가도록 하는 길찾기 문제를 풀 때, 정책 π가 존재한다면 길을 훨씬 수월하게 찾을 것.
π는 어느 위치에 있더라도 보상을 최대화 할 수 있는 방법 알려줌 => 미래에 받을 보상들의 합이 최대가 되도록 함

Value function : 어떤 상태 S와 정책 π가 주어졌을 때, 계산되는 누적 보상의 기댓값

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t\geq 0}\gamma^t r_t | s_0 = s, \pi\right]$$

Q-Value function : 상태 s에서 어떤 행동을 해야 가장 좋은지 알려주는 함수

$$Q^{\pi}(s,a) = \mathbb{E}\left[\sum_{t\geq 0}\gamma^t r_t | s_0 = s, a_0 = a, \pi\right]$$

Bellman equation

: 현재 state의 value function과 next state의 value function 사이의 관계식
어떤 상태 s가 주어졌을 때, Q-Value function에 의해서 Q*를 알게 되었다고 가정하면 Q*는 어떤 행동을 취할 때 미래에 받을 보상의 최대치이기 때문에 다음 s'에서도 최적의 경로를 도출할 수 있음.

최적의 정책을 구하는 방법
=> Value iteration algorithm
: 반복적인 update를 통해 벨만 방정식을 이용하여 점차적으로 Q*를 최적화 시키는 방법
scalable 하지 않음 => Q(s, a)를 계산 가능하도록 근사

Q-Learning

Q-learning: Use a function approximator to estimate the action-value function

$$Q(s,a;\theta) \approx Q^*(s,a)$$

function parameters (weights)

If the function approximator is a deep neural network => **deep q-learning**!

현재 Action-value function을 추정하는데 사용한 Q - function은 Bellman equation을 만족하도록 만들어야 함.

Forward Pass

Loss function: $L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot)}\left[(y_i - Q(s,a;\theta_i))^2\right]$

where $y_i = \mathbb{E}_{s'\sim\mathcal{E}}\left[r + \gamma \max_{a'} Q(s',a';\theta_{i-1})|s,a\right]$

Backward Pass

Gradient update (with respect to Q-function parameters θ):

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot);s'\sim\mathcal{E}}\left[r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) - Q(s,a;\theta_i))\nabla_{\theta_i} Q(s,a;\theta_i)\right]$$

손실함수(Loss function)는 Q(s,a)가 목적함수(yi)랑 얼마나 차이가 나는지 측정.

또한 yi는 Bellman equation을 만족하는 정답 방정식. Backward Pass 과정에서는 Loss값에 대하여 θ를 계속 update를 진행.

Policy Gradients

:상태에 따른 Q - value값들을 학습시키는 것이 아니라 정책(Policy) 자체를 학습시키는 방법

Policy Gradients

Formally, let's define a class of parametrized policies: $\Pi = \{\pi_\theta, \theta \in \mathbb{R}^m\}$

For each policy, define its value:

$$J(\theta) = \mathbb{E}\left[\sum_{t \geq 0} \gamma^t r_t | \pi_\theta\right]$$

J(θ)는 미래에 받을 보상을 누적으로 하여 기댓값을 나타낸 함수
구체적인 값(Q-value)을 몰라도 정책 자체의 gradient를 구해 최적의 정책을 찾을 수 있다.
분산이 너무 높다는 문제 발생 => 분산을 줄이는 것은 Policy Gradient에서 가장 중요한 요인 중 하나.
=> 해당 상태로부터 받을 미래 보상만을 고려하여 어떤 행동을 취할 확률을 키워주는 방법
=> 지연된 보상에 의해서 할인률을 적용하는 것
=> Baseline

- 중요한 것은 얻은 보상이 얻을 것이라고 예상했던 것보다 좋은건지 아닌지 판단하는 것것