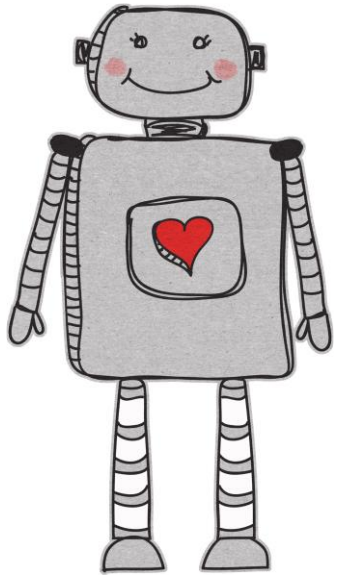
KAIST-Samsung DS AI Expert

딥 강화학습 (Deep Reinforcement Learning)

TA 강민구, 최윤선 (2020. 07. 28)
TA 이종민, 박재영 (2020. 08. 04)

강화학습 (Reinforcement Learning)



에이전트
(현재 상태 s_t)

행동 a_t

보상 r_t
다음 상태 s_{t+1}

환경

- 환경에 대해 전혀 모르는 상태에서 시작하여 환경과의 상호작용을 통해 누적 보상의 기댓값을 최대화하는 행동 정책(policy) 학습하기!

π: (state → action) mapping

오늘 실습 내용

- 딥 **강화학습** (Deep Reinforcement Learning) 알고리즘 구현
  - Deep Q-Network (DQN)
  - Proximal Policy Optimization (PPO)
  - Policy Optimization from Demonstration (POfD)

실습 주제 1:

Deep Q-Network (DQN)

Recap) Finite MDP 환경에서의 강화학습

- **벨만 최적 방정식 (Bellman Optimality Equation)**
  - Optimal policy π^*의 Q function이 만족하는 재귀 관계식

$$Q^*(s,a) = R(s,a) + \gamma \mathbb{E}_{s' \sim T(\cdot|s,a)} \left[\max_{a'} Q^*(s',a') \right]$$

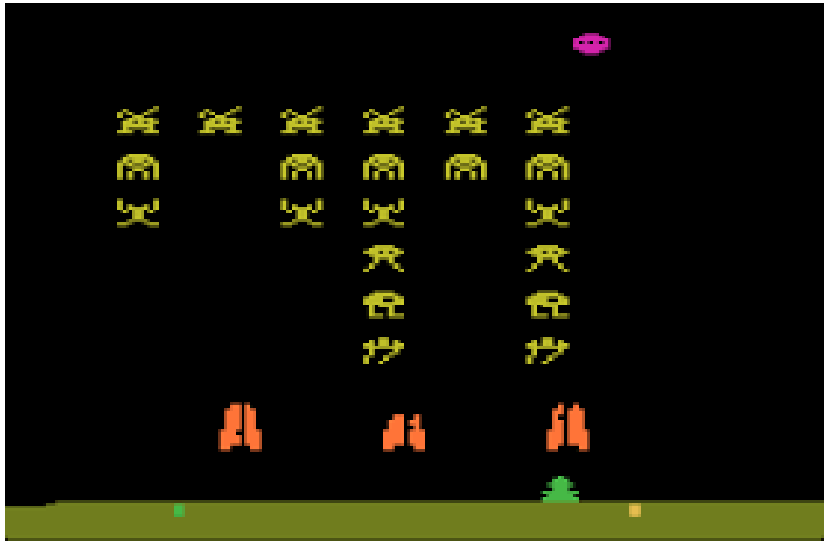
- **Q-Learning**
  - 벨만 최적 방정식을 만족하는 Q^*를 찾으면 $\pi^*(s) = \text{argmax}_a Q^*(s,a)$는 항상 optimal policy.
  - 현재 상태 s에서 행동 a를 선택한 후 보상 r 및 다음 상태 s'을 관찰할 때마다 아래와 같은 식으로 **Q 테이블** 업데이트.

$$Q(s,a) \leftarrow Q(s,a) + \alpha_t \left(r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right)$$

Deep Q-Network (DQN)

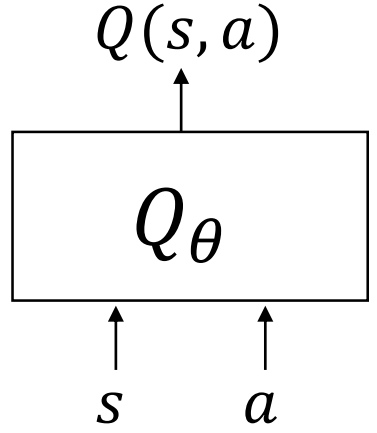
- 만약 MDP 상태의 수가 유한하지 않다면? 혹은 너무 많다면?
  - $|S||A|$ 크기의 Q 테이블을 만들고 업데이트하는 게 거의 불가능

고차원의 연속 상태 공간



이미지: $|S| = W \times H \times 255^3$

DQN의 뼈대: Neural Fitted Q Iteration



- $|S||A|$ 크기의 테이블을 만드는 대신
  - $S \times A \to \mathbb{R}$ 의 함수를 뉴럴 네트워크 Q_θ로 표현해보자!

- 파라미터 θ는 어떻게 학습? 벨만 최적 방정식을 만족시키는 방향으로

$$Q^*(s,a) = R(s,a) + \gamma \mathbb{E}_{s' \sim T(\cdot|s,a)} \left[\max_{a'} Q^*(s',a') \right]$$

좌/우변의 차이를 최소화하는 파라미터 θ를 찾자: 데이터 $\mathcal{D} = \{(s,a,r,s')\}$로부터,

$$\theta_k \leftarrow \arg\min_{\theta_k} \mathbb{E}_{(s,a,r,s') \sim \mathcal{D}} \left[\left(Q_{\theta_k}(s,a) - y_k \right)^2 \right]$$

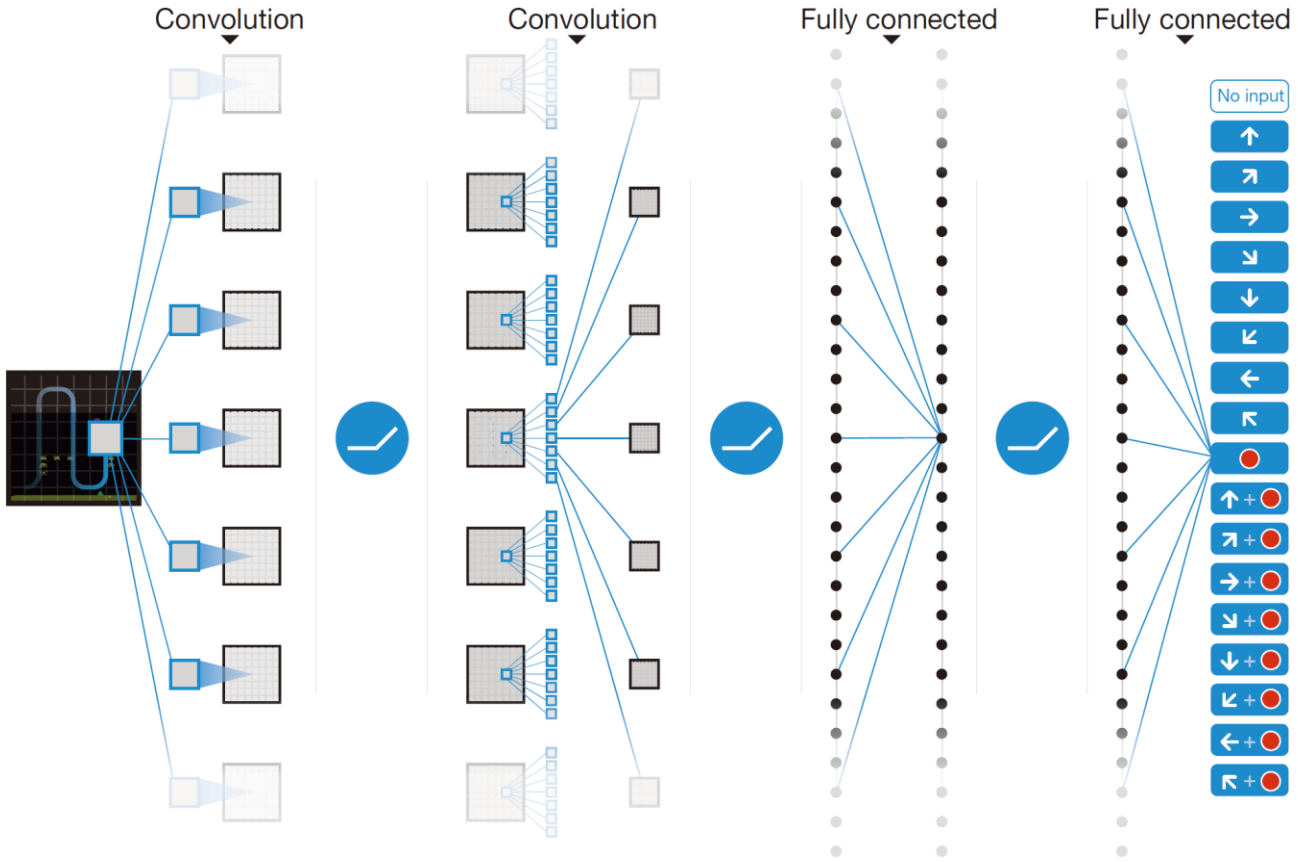
➡ **(단순 회귀문제)**

$$\text{where } y_k = \begin{cases} r & \text{if } k = 0 \text{ or } s' \text{ is terminal} \\ r + \gamma \max_{a'} Q_{\theta_{k-1}}(s',a') & \text{otherwise} \end{cases}$$

DQN의 실제 구현 – 뉴럴 네트워크 아키텍쳐

- $S \times A \rightarrow \mathbb{R}$의 함수를 학습하는 대신 $S \rightarrow \mathbb{R}^{|A|}$의 함수를 학습.

입력: 현재 상태
(이미지, 센서 관측값)



출력: 각 행동의
Q-value 들

DQN의 핵심 – Experience Replay + Target Q-Network

- 매 스탭마다 얻는 (s, a, r, s') 정보를 Replay buffer $\mathcal{D}$에 쌓아둔다.
  - (예: 매 timestep/episode마다) $\{(s, a, r, s')\}$의 mini-batch 데이터를 샘플
  - 샘플 된 mini-batch 데이터에 대하여 다음의 식 최적화

$$\min_{\theta} \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}} \left[(Q_\theta(s,a) - y)^2 \right] \text{ where } y = \begin{cases} r & \text{if } s' \text{ is terminal} \\ r + \gamma \max_{a'} Q_{\bar{\theta}}(s', a') & \text{otherwise} \end{cases}$$

  - θ를 학습할 때 $\bar{\theta}$ 는 θ에 대해 상수 취급: 단순한 회귀 문제를 푸는것과 동등

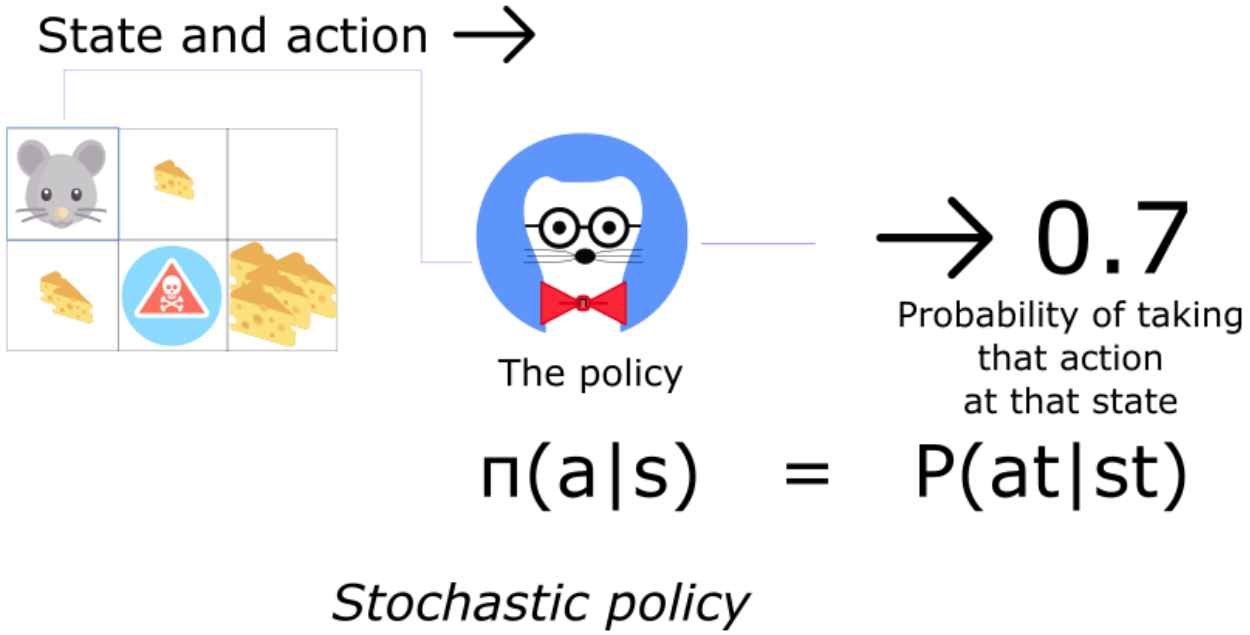
- Target Q-network의 파라미터 $\bar{\theta}$ 는 주기적으로 θ와 동기화
  - (예: 10 에피소드마다) $\bar{\theta} \leftarrow \theta$

실습 1. Deep Q-Network (DQN)

- DQN 뼈대 코드 이해하기
- 빈칸 채우기
  - q_model.get_weights
  - target_q_model.set_weights
- 여러 파라미터들을 조절해보며 실험해보기
  - 뉴럴 네트워크 히든 노드 수, epsilon, batch_size 등

실습 주제 2:
Policy Gradients (PPO)

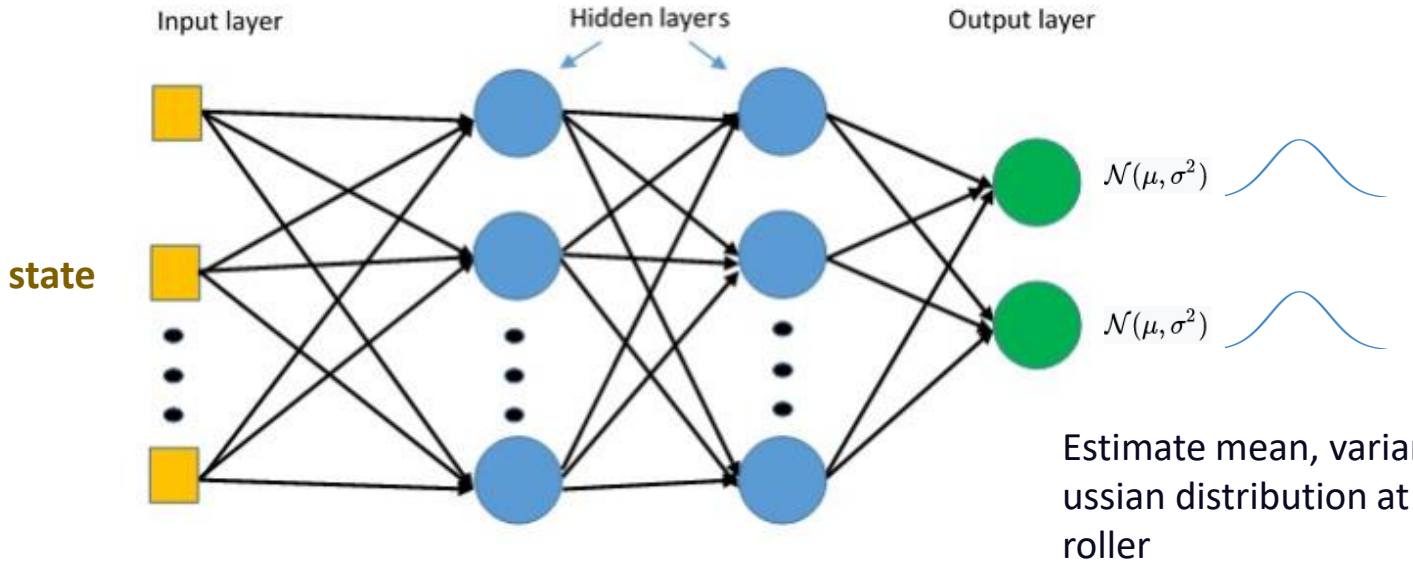
서론 (1) Stochastic policy



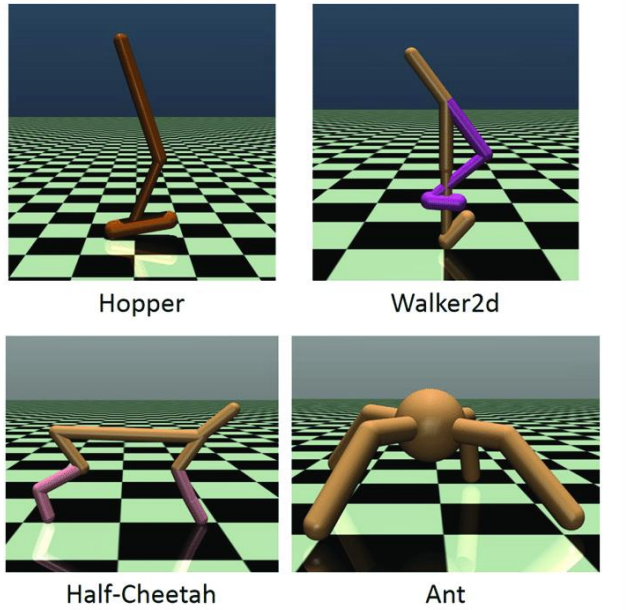
- Stochastic policy는 각 행동(actions)에 대한 확률 분포를 결정한다.
- 하나의 행동을 취하기 보다는, stochastic policy는 다른 행동들을 취할 가능성을 항상 가지고 있다.

보통, Policy gradients 방법은 stochastic policy를 학습한다!

서론 (2) Stochastic policy over continuous action space



Estimate mean, variance for Ga
ussian distribution at each cont
roller



Robotics - continuous action

- 로보틱스 분야에서는 continuous action space에서 로봇을 제어해야한다.이때, Stochastic policy는 각각의 controller가 취해야 하는 행동(actions)에 대한 확률 분포를 결정한다.
- 보통, Gaussian distribution policy를 많이 사용하며 policy는 평균과 분산을 결정한다.

실습 2-1. Gaussian Policy network 구성하기

run_traj.py (STEP 1) 가우시안 평균을 Estimate 하는 NN Policy 작성하기

```
class Policy(tf.layers.Layer):
    """ NN-based policy approximation """
    def __init__(self, obs_dim, act_dim):
        """
        Args:
            obs_dim: num observation dimensions (int)
            act_dim: num action dimensions (int)
        """
        super(Policy, self).__init__()
        self.obs_ph = tf.keras.layers.Input(obs_dim, name='obs')  # [batch_size, obs_dim]
        """
        STEP 1
        Gaussian NN Policy
        using tf.keras.layers.Dense
        """
        ########################
        # YOUR IMPLEMENTATION PART

        ### Set hid_size freely,
        hid1_size =0
        hid2_size = 0
        # 2 hidden layers with tanh activations
        h1 = 0
        h2 = 0
        means = 0
        ########################
```

(힌트) tf.keras.layers.Dense 사용

2개의 mlp를 쌓아보고, 각 layer의 unit 개수는 자유롭게 지정.

최종 아웃풋은 각 액션에 대한 Normal distribution을 나타냄.

이를 위해 means와 variances를 추정하는 NN policy를 구성

실습 2-2. 하나의 에피소드 동안 trajectory 뽑아내기

run_traj.py (STEP 2)

- 현재의 observe를 보고, policy로부터 actions을 결정하여 env로 부터 reward를 받을때, 한 에피소드 동안 경험한 observe, action, reward를 각각의 리스트에 저장
- 한 에피소드 동안 받은 총 reward의 합을 계산하여, accumulated reward(또는 return)을 구함

```
obs = env.reset()
observes, actions, rewards= [], [], []
done = False
while not done:
    ########################
    # YOUR IMPLEMENTATION PART
    pass
    ########################

accumulated_reward = np.sum(rewards)
print("=====================================")
print(f"During one episodes, The accumulated Rewards: {accumulated_reward}")
print("=====================================")
```

실습 2-3. 10개의 에피소드 동안 trajectory 뽑아내기

run_traj.py (STEP 3)

- 현재의 observe를 보고, policy로부터 actions을 결정하여 env로 부터 reward를 받음
- 한 에피소드 동안 받은 총 reward의 합을 계산하여, accumulated reward(또는 return)을 구함
- 10개의 에피소드 동안 받은 accumulated reward의 평균값을 계산하여 avg_returns을 완성

```
episodes = 10

total_reward = 0
returns = []
for e in range(episodes):
    obs = env.reset()
    done = False
    accumulated_reward = 0
    # For one episode
    for t in range(10000):
        ########################
        # YOUR IMPLEMENTATION PART
        accumulated_reward += 0
        ########################

        if done:
            break
    returns.append(accumulated_reward)

avg_returns = np.mean(returns)
```

Policy Gradients (1) Objective function

$$L^{PG}(\theta) = E_t[\log \pi_\theta(a_t|s_t) * A_t]$$

Policy Loss

Expected

log probability of taking that action at that state

Advantage if A>0, this action is better than the other action possible at that state

- Advantage Function

강화학습에서 어떤 행동이 얼머나 좋은지, 절대적인 수치로 알 필요는 없다. 단지, 모든 행동을 하였을때 평균적으로 얻어지는 리워드보다 얼마나 더 잘 하는지 알면 된다. 상대적인 지표로 어떠한 행동이 더 좋은지 나타낸 것이 advantage function 이다.

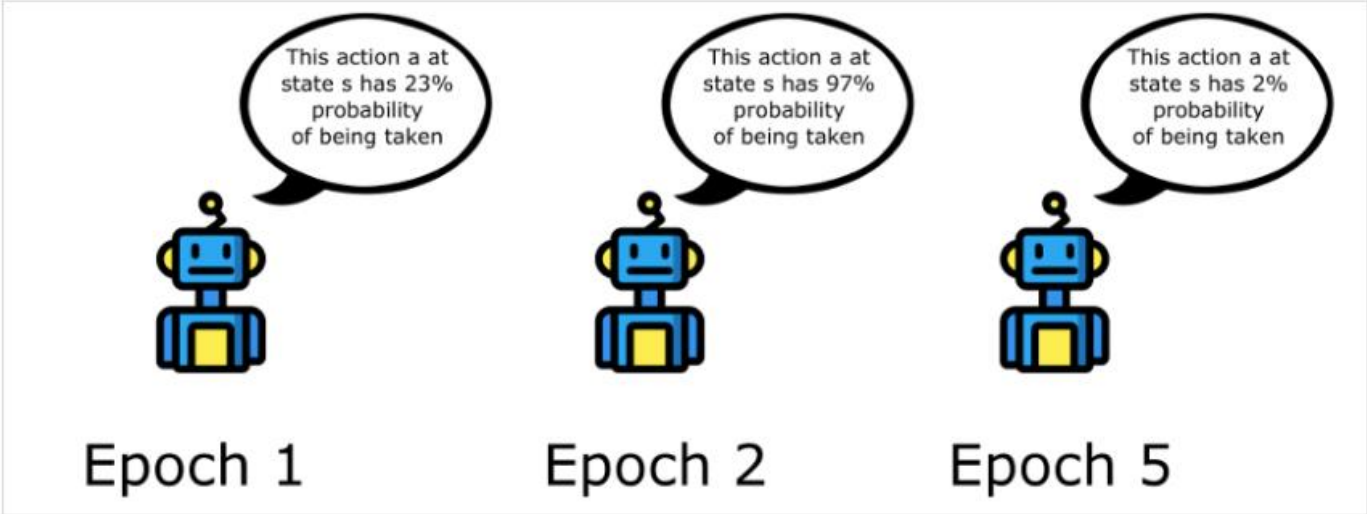
$$A_t \simeq A(s_t, a_t) = Q(s_t, a_t) - V(s_t)$$

- Policy gradients method의 objective function은 위와 같으며, 위 식을 maximize 하기 위해 gradient ascent을 취한다.
- 에이전트가 더 많은 reward값을 가지게 되는 행동에 대해서, 더 높은 확률을 가질 수 있도록 학습한다.

Policy Gradients (2) Problem of policy gradients objective

만약, gradient의 step size가

- 너무 작으면, training이 오래 걸린다.
- 너무 크면, 행동의 변화(variability)가 커져 학습이 이루어지지 않는다.



When there is enormous variability in the training (Robot Icons made by Smashicons)

안정적인 학습이 진행되도록 하기 위해,
PPO는 각 training 단계에서 업데이트 되는 policy의 변화에 제한을 둔다.

- PPO는 policy의 변화를 제한하기 위해 clip을 사용한 'Clipped surrogate objective function'을 제안한다.

Policy Gradients (3) Clipped Surrogate Objective function

rt(θ) 는 현재 policy와 이전(old) policy가 어떤 상태(st)에 그 행동(at)을 취할 확률의 비율이다.

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}, \text{ so } r(\theta_{\text{old}}) = 1.$$

따라서 rt(θ) 는,

- rt(θ) >1 이면, 그 행동이 현재 policy에서 이전 policy보다 더 취해질 가능성이 높다라는 것을 뜻한다.
- rt(θ) 이가 0 과 1 사이이면, 현재 policy에서 이전의 policy보다 그 행동이 취해질 가능성이 낮다라는 것을 뜻한다.

이전의 policy와 현재의 policy의 비율을 고려한, clippeed objective function은 다음과 같다.

$$L^{CPI}(\theta) = \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}\hat{A}_t\right] = \hat{\mathbb{E}}_t\left[r_t(\theta)\hat{A}_t\right]$$

Policy Gradients (3) Clipped Surrogate Objective function

$$L^{CPI}(\theta) = \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}\hat{A}_t\right] = \hat{\mathbb{E}}_t\left[r_t(\theta)\hat{A}_t\right]$$

- 위 식에서, 이전의 **policy**보다 현재의 **policy**에서 더 높은 확률을 가지게 되는 행동을 한다면 **gradient**의 **step**이 커져, **policy**에 큰 변화를 야기시킨다.
- 이전의 **policy**와 많이 다르지 않은 **policy**로 업데이트 하기 위해, **clip**을 사용하여 제약을 둔다.

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min(\underbrace{r_t(\theta)\hat{A}_t}_{\text{L CPI}}, \underbrace{\text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t}_{\text{Modifies the surrogate objective by clipping the prob ratio. --> Which removes the incentive for moving rt outside of the interval [1 - e, 1 + e]}})\right]$$

The Clipped Surrogate Objective function

PPO는 objective 함수에 변화할수 있는 정도에 제약을 두는 clip을 사용한다.

Policy Gradients (3) Clipped Surrogate Objective function

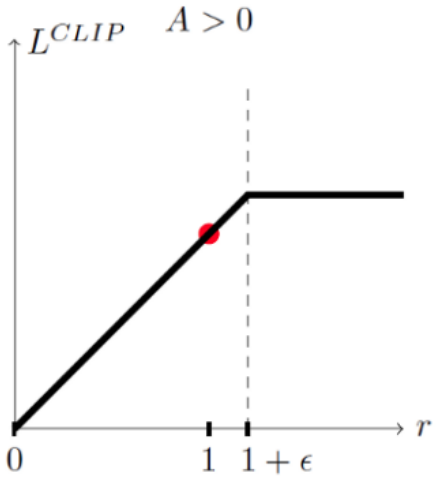
Clipped Objective 함수에는 두가지의 확률 비율이 있는데, (1) clipped 되지 않은 것과 (2) [1 — ϵ, 1+ϵ] 사이로 clip 된 것이 있다.

PPO Objective

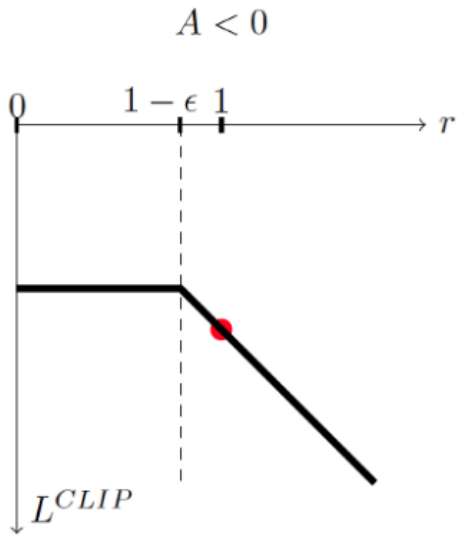
$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right]$$

그리고, (1) clip된 것과 (2) clip 되지 않은 것 중 작은 값에 대해 policy update를 진행한다.

결론적으로,
각각의 경우에 대해 그래프로 나타내면,



Case 1: When the advantage is > 0



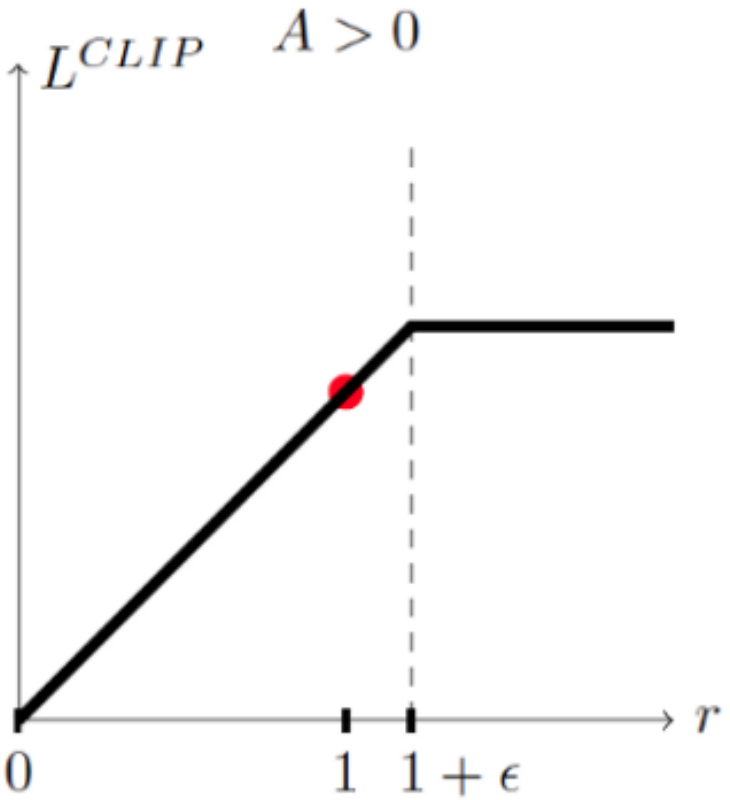
Case 2: When the advantage Ât is smaller than 0

Policy Gradients (3) Clipped Surrogate Objective function

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right]$$

Case 1: When the advantage is > 0

Ât > 0 경우, 그 행동은 모든 행동들이 그 상태에서 취했을 때 가지게 되는 기대되는 리워드 값보다 더 큰 리워드를 가지게 된다는 것을 뜻한다.
따라서, 새로운 policy는 그 상태(state)에 그 행동을 취할 확률을 높이는 방향으로 학습되어야 한다.
하지만, Objective 함수는 clip으로 인해 rt(θ)에 대해서 최대 1+ε만큼만 증가할 수 있다.

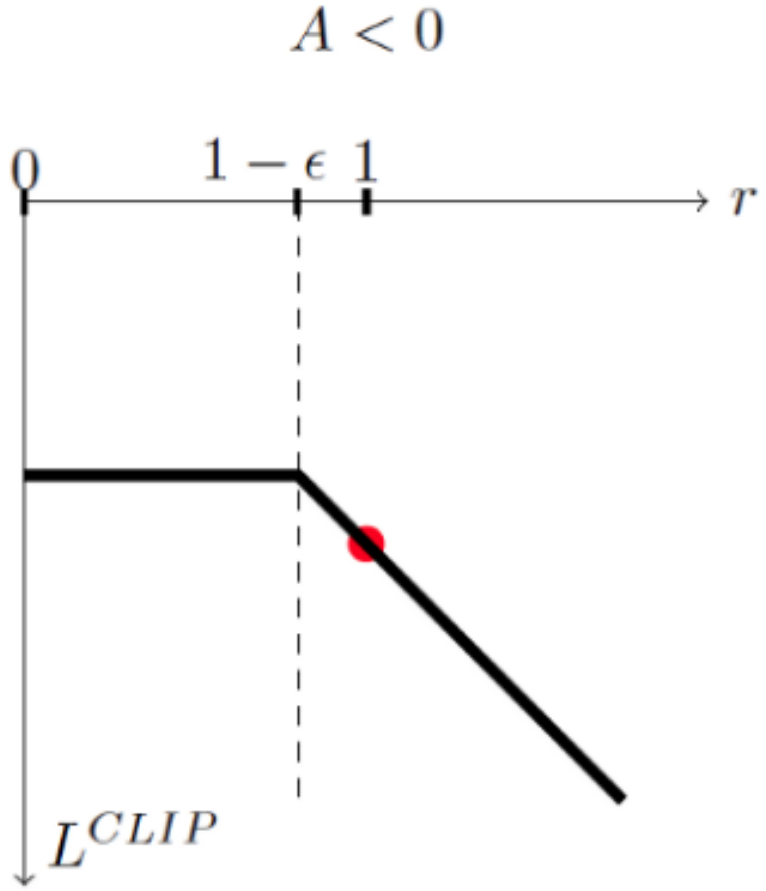


Policy Gradients (3) Clipped Surrogate Objective function

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right]$$

Case 2: When the advantage Ât is smaller than 0

Ât < 0 라면, 그 행동은 그 상태에서 어떤 행동을 취했을 때, 기대되는 리워드 값보다 작기 때문에 지양해야한다.
따라서, rt(θ)가 증가함에 따라 Objective는 감소한다.
clip으로 인해 rt(θ)는 적어도 1-ε만큼의 값을 가진다.



실습 2-4. Clipped Surrogate Objective Function

train.py (STEP 4) Clipped Surrogate Objective Function 작성하기

(Hint) 함수 사용: tf.exp, tf.clip_by_value, tf.minimum

```
class Policy(object):
    """ NN-based policy approximation """

    def __init__(self, obs_dim, act_dim, clipping=0.2):
        """
        Args:
            obs_dim: num observation dimensions (int)
            act_dim: num action dimensions (int)
        """
        self.means = tf.keras.layers.Dense(act_dim, name="means", activation="linear")(h3)
        self.log_vars = tf.get_variable('logvars', (act_dim), tf.float32,
                                        tf.constant_initializer(-1.0))
        logp = -0.5 * tf.reduce_sum(self.log_vars)
        logp += -0.5 * tf.reduce_sum(tf.square(self.act_ph - self.means) /
                                     tf.exp(self.log_vars), axis=1)
        logp_old = -0.5 * tf.reduce_sum(self.old_log_vars_ph)
        logp_old += -0.5 * tf.reduce_sum(tf.square(self.act_ph - self.old_means_ph) /
                                         tf.exp(self.old_log_vars_ph), axis=1)

        """
        STEP 4
        The Clipped Surrogate Objective Function
        """
        ########################
        # YOUR IMPLEMENTATION PART
        pg_ratio = 0
        clipped_pg_ratio = 0
        surrogate_loss = 0
        ########################
        self.loss = -tf.reduce_mean(surrogate_loss)
```

실습 주제 3:
Policy Optimization from Demonstration (POfD)

Policy Optimization from Demonstration (POfD)

- 만약 MDP 상태/행동의 수가 너무 많다면?
  - 모든 (s, a)에 대해 보상함수를 설계하는 것이 매우 어려움
  - 또한 보상함수가 조금만 잘못 주어져도, 학습한 행동이 의도와는 전혀 다르게 동작할 수 있음

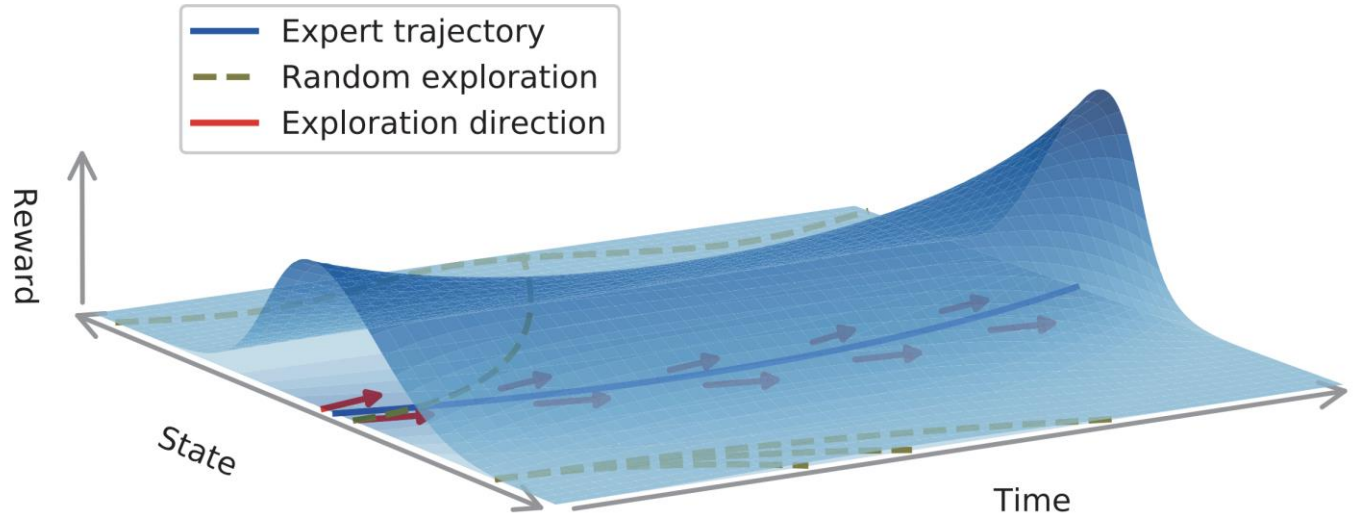
[고차원의 연속 상태 공간]

- 가장 쉽고 정확한 보상함수 설계 방법
  - 에이전트가 특정 작업을 완수하였을 때에만 보상을 제공 (sparse reward 환경)
  - 하지만, 보상을 한번이라도 받기 전까지는 학습에 활용할 학습 신호 자체가 없음
  - 탐색 과정에서 작업을 완수해 본 경험이 필요함을 의미하나 복잡한 문제 상황에서는 불가능

    ➔ 즉, 학습 자체가 불가능할 수 있음!

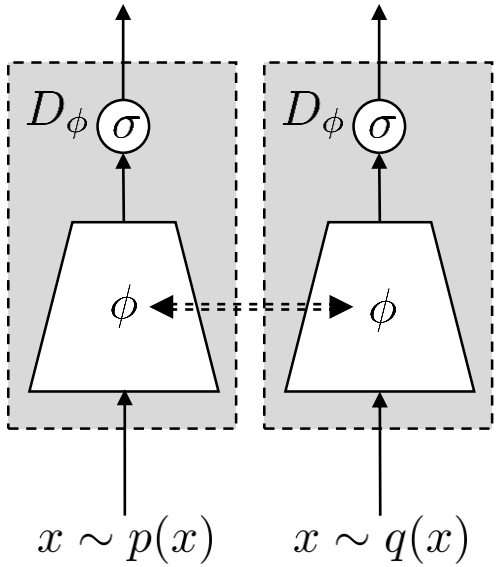
Policy Optimization from Demonstration (POfD)

- Sparse reward 환경에서도 학습이 가능하도록 demonstration을 활용
- 최초의 보상을 받을 때 까지 demonstration을 모사하도록 학습이 진행
  - Demonstration의 모사를 통해 에이전트의 탐색 과정을 유도
- 보상을 받는 경우, 해당 보상을 학습에 활용하여 실제 작업 완수를 위한 학습 진행



POfD의 뼈대: Generative Adversarial Network(GAN)의 Discriminator

- Generative Adversarial Imitation Learning (GAIL) 모사학습의 목적함수
  - $\min_\theta \mathrm{D}_{\mathrm{JS}}\left[\rho_{\pi_\theta}(s,a) \| \rho_{\pi_D}(s,a)\right]$
    - $\rho_\pi(s,a) = \pi(a|s)\sum_{t=0}^{\infty}\gamma^t P(s_t = s|\pi)$: occupancy measure
    - $\mathrm{D}_{\mathrm{JS}}\left[p\|q\right]$: 서로 다른 두 확률 분포 p와 q 사이의 차이를 측정하는 척도
      - $p = q \Rightarrow \mathrm{D}_{\mathrm{JS}}\left[p\|q\right] = 0$



  - $\mathrm{D}_{\mathrm{JS}}\left[\cdot\|\cdot\right]$ 를 어떻게 계산하지? GAN의 discriminator를 활용하자!
    - $\max_\phi \mathbb{E}_{p(x)}\left[\log D_\phi(x)\right] + \mathbb{E}_{q(x)}\left[\log\left(1 - D_\phi(x)\right)\right]$

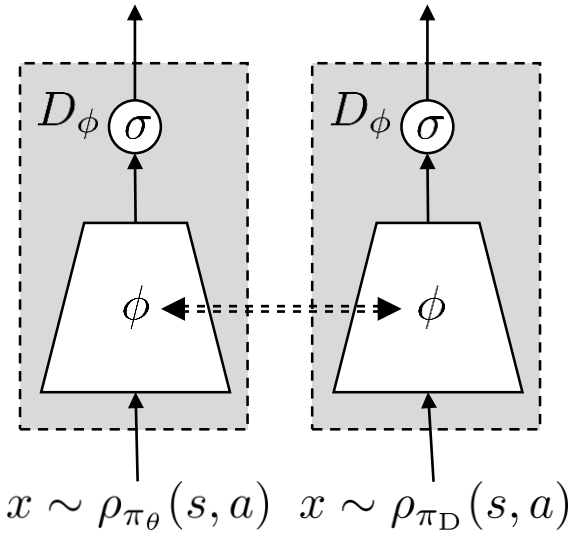
      : 최적의 파라미터 ϕ^*의 경우 $\mathrm{D}_{\mathrm{JS}}\left[p(x)\|q(x)\right]$의 값을 estimate 할 수 있음

      $(\because)\ \mathbb{E}_{p(x)}\left[\log D_{\phi^*}(x)\right] + \mathbb{E}_{q(x)}\left[\log\left(1 - D_{\phi^*}(x)\right)\right] = 2 * \mathrm{D}_{\mathrm{JS}}\left[p(x)\|q(x)\right] - \log 4$

POfD의 목적함수 및 학습 과정

- POfD의 목적함수
  - $\max_{\theta} \mathbb{E}_{\rho_{\pi_\theta}(s,a)}[r(s,a)] - \lambda \cdot \mathrm{D}_{\mathrm{JS}}[\rho_{\pi_\theta}(s,a) \| \rho_{\pi_D}(s,a)]$
    - 보상값이 0으로 주어지는 동안

      → $\max_{\theta} -\lambda \cdot \mathrm{D}_{\mathrm{JS}}[\rho_{\pi_\theta}(s,a) \| \rho_{\pi_D}(s,a)]$ 로 모사학습 수행

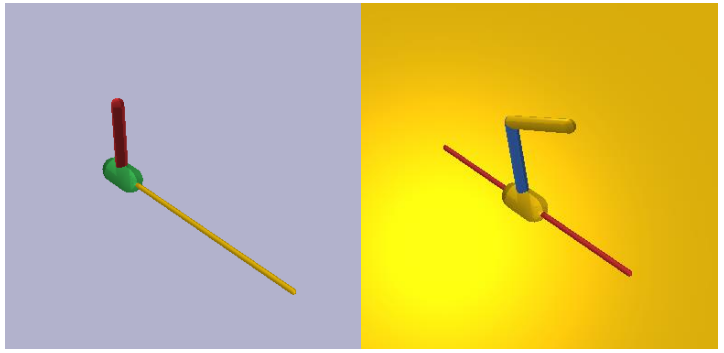


- POfD의 학습과정: 아래의 (1)-(2)를 반복하여 수행
  - (1) Discriminator 파라미터 ϕ^* 학습

    : $\max_{\phi} \mathbb{E}_{\rho_{\pi_\theta}(s,a)}[\log D_\phi(s,a)] + \mathbb{E}_{\rho_{\pi_D}(s,a)}[\log(1 - D_\phi(s,a))]$
  - (2) 학습된 discriminator 파라미터 ϕ^*를 기반으로 행동 정책 학습

    : $\max_{\theta} \mathbb{E}_{\rho_{\pi_\theta}(s,a)}[r(s,a) - \lambda \cdot \log D_{\phi^*}(s,a)]$ → 즉, discriminator 출력을 추가 보상값으로 활용

실습 3. POfD



- POfD 뼈대 코드 이해하기
  - PPO와 동일한 코드에서 다음 사항이 추가/수정
    - 모사학습을 위한 보상 함수 생성용 discriminator class가 추가됨
    - 강화학습 에이전트가 discriminator의 보상값을 활용하도록 수정
    - 학습에 활용할 demonstration은 demo 폴더에 존재

- 실습 도메인
  - InvertedPendulumPyBulletSparseEnv-v0
    - 보상함수: 200 step 동안 살아있을 때마다 +100의 보상을 지급
  - InvertedDoublePendulumPyBulletSparseEnv-v0
    - 보상함수: 두번째 폴의 높이가 0.89 이상일 때마다 +1의 보상을 지급

실습 3. POfD

- 빈칸 채우기

- 파라미터들을 조절해보며 실험해보기
  - reward_lambda (λ): discriminator의 보상값 활용 정도
    - 학습에 활용하는 보상값 = 실제 환경에서의 보상값 + λ * discriminator 보상값