RLIF 코드 분석 자료

2024.04

CoRLHF

알고리즘 실행 코드

코드는 Parsing해서 사용

```
python3.8 -m RLIF.examples.train_rlif_main \   # RLIF.examples.train_rlif_main 모듈 실행
    --env_name "pen-expert-v1" \   # env_name: pen-expert-v1실행
    --sparse_env 'AdroitHandPenSparse-v1' \  # spare_env 버전으로 환경 설정
    --dataset_dir 'ENTER DATASET DIR' \ # data set 위치
    --expert_dir 'ENTER EXPERT DIR' \  # expert dir 위치
    --ground_truth_agent_dir 'ENTER GROUND TRUTH AGENT DIR' \ # 기준이 되는 agent 위치
    --logging.output_dir './experiment_output'  # --logging.output_dir 부분에서 에러 발생 제외하고 실행
```

알고리즘 실행 코드

Dagger 실행 코드

```
python3.8 -m RLIF.examples.train_dagger_main \
    --dense_env "pen-expert-v1" \
    --sparse_env 'AdroitHandPenSparse-v1' \
    --intervention_strategy '' \
    --dataset_dir '' \
    --expert_dir './RLIF/experts/bc_experts/960d7aba1a654a6aae323b7558bf3378/model.pkl' \
    --ground_truth_agent_dir './RLIF/experts/rlpd_experts/s24_pen-expert-v1env/model.pkl' \
    --logging.output_dir './experiment_output' \
    --logging.online
```

RLIF 실행 코드

```
python3.8 -m RLIF.examples.train_rlif_main \
    --env_name "pen-expert-v1" \
    --sparse_env 'AdroitHandPenSparse-v1' \
    --intervention_strategy '' \
    --dataset_dir '' \
    --expert_dir './RLIF/experts/bc_experts/382bae4018ad468dae384799ab8d81ba/model.pkl' \
    --ground_truth_agent_dir './RLIF/experts/bc_experts/960d7aba1a654a6aae323b7558bf3378/model.pkl'
```

configuration

```
FLAGS_DEF = define_flags_with_default(
    project_name="rlpd_itv_test",
    env_name="hopper-expert-v2",
    sparse_env='Hopper-v2',
    offline_ratio=0.5,
    seed=43,
    train_sparse=False,
    dataset_dir='',

    expert_dir='./RLIF/experts/rlpd_experts/s24_hopper-expert-v2env/model.pkl',
    ground_truth_agent_dir='./RLIF/experts/rlpd_experts/s24_hopper-expert-v2env/model.pkl',
    intervene_threshold=0.0,
    intervention_strategy='',
    intervene_n_steps=4,

    eval_episodes=100,
    log_interval=1000,
    eval_interval=10000,
    max_traj_length=200,
    batch_size=256,
    max_steps=int(1e6),
    start_training=0,
    pretrain_steps=0,

    tqdm=True,
    save_video=False,
    save_model=False,
    checkpoint_model=False,
    checkpoint_buffer=False,
    utd_ratio=1,
    binary_include_bc=True,
    )
```

- utils에서 **define_flags_with_default** 함수 import
- 각 설정 값들을 define_flags_with_default를 통해 Flag로 전환
  코드 실행 과정에서 parsing한 값들을 전역으로 사용하기 위함

```
def define_flags_with_default(**kwargs):
    for key, val in kwargs.items():
        if isinstance(val, ConfigDict):
            config_flags.DEFINE_config_dict(key, val)
        elif isinstance(val, bool):
            # Note that True and False are instances of int.
            absl.flags.DEFINE_bool(key, val, 'automatically defined flag')
        elif isinstance(val, int):
            absl.flags.DEFINE_integer(key, val, 'automatically defined flag')
        elif isinstance(val, float):
            absl.flags.DEFINE_float(key, val, 'automatically defined flag')
        elif isinstance(val, str):
            absl.flags.DEFINE_string(key, val, 'automatically defined flag')
        else:
            raise ValueError('Incorrect value type')
    return kwargs
```

```
config_flags.DEFINE_config_file(
    "config",
    "./RLIF/configs/rlpd_config.py",
    "File path to the training hyperparameter configuration.",
    lock_config=False,
)
```

Lock_config: 설정 파일의 내용을 프로그램 실행 도중에 변경할 수 있음

Combine 함수

- On-line 데이터와 off-line 데이터를 결합하는데 사용
- main 함수 379번 라인에서 사용

```
def combine(one_dict, other_dict):
    combined = {}
    for k, v in one_dict.items():
        if len(v.shape) > 1:
            tmp = np.vstack((v, other_dict[k]))
        else:
            tmp = np.hstack((v, other_dict[k]))
        combined[k] = tmp
    return combined
```

Combined dict 생성

main 함수

초기화 작업

Flag변수
플래그(flag)란 깃발이라는 뜻인데, 컴퓨터에서 무언가를 기억하거나 또는
다른 프로그램에게 약속된 신호를 남기기 위한 용도로 프로그램에 사용되는 미리 정의된 비트

```
FLAGS = flags.FLAGS
assert FLAGS.offline_ratio >= 0.0 and FLAGS.offline_ratio <= 1.0        # 플래그의 값이 0~1, 0이면 오프라인 데이터 사용 X, 1이면 오프라인 데이터만 사용

os.environ["XLA_PYTHON_CLIENT_ALLOCATOR"] = "platform"   # 메모리 할당기를 "platform“ 으로 설정
os.environ["XLA_PYTHON_CLIENT_PREALLOCATE"] = "false"   # 메모리의 사전 할당을 비활성화

set_random_seed(FLAGS.seed) # set_random_seed(seed): 시스템의 seed를 설정하는 함수

wandb.init(project=FLAGS.project_name, mode='online')  # Wandb (시각화 프로그램) 초기화 및 설정
wandb.config.update(FLAGS)

exp_prefix = f"s{FLAGS.seed}_{FLAGS.pretrain_steps}pretrain_{FLAGS.utd_ratio}utd_{FLAGS.offline_ratio}offline"
if hasattr(FLAGS.config, "critic_layer_norm") and FLAGS.config.critic_layer_norm:
    exp_prefix += "_LN"

log_dir = os.path.join(FLAGS.log_dir, exp_prefix) # 로그 디렉토리 경로를 생성

if FLAGS.checkpoint_model:
    chkpt_dir = os.path.join(log_dir, "checkpoints")
    os.makedirs(chkpt_dir, exist_ok=True)

if FLAGS.checkpoint_buffer:
    buffer_dir = os.path.join(log_dir, "buffers") # 버퍼 디렉토리 생성
    os.makedirs(buffer_dir, exist_ok=True)

if FLAGS.save_model:
    model_dir = os.path.join(log_dir, "model")   # 최종 학습된 모델을 디스크에 저장하기 위한 디렉토리를 생성
    os.makedirs(model_dir, exist_ok=True)
```

환경 생성

```
env = gym.make(FLAGS.env_name)  # FLAGS.env_name에 저장된 이름으로 gym.make
env = wrap_gym(env, rescale_actions=True)  # Wrap_gym (env): 사용자 정의 함수로 env
env = gym.wrappers.RecordEpisodeStatistics(env, deque_size=1)  # RecordEpisodeStatistics: 각 에피소드가 끝날 때마다 해당 에피소드에 대한 통계를 자동 기록
                                                               # deque_size=1로 설정, 가장 최근의 에피소드 통계 만 유지하고, 이전 에피소드의 통계는 삭제
env.seed(FLAGS.seed)
ds = D4RLDataset(env) # D4RLDataset을 사용하여 지정된 환경으로 데이터셋 생성

eval_env = gym.make(FLAGS.env_name) # gym.make 함수를 사용하여 강화학습에서 평가용 환경(eval_env)을 생성
eval_env = wrap_gym(eval_env, rescale_actions=True) # 평가용 환경(eval_env)에 wrap_gym 함수를 적용하여 액션 값들을 [-1, 1] 범위로 재 조정
eval_env.seed(FLAGS.seed + 42)  # FLAGS.seed 값에 42를 더한 값으로 시드를 설정하여, 학습 환경과는 다른 난수 패턴을 생성

sparse_eval_sampler = TrajSampler(GymnasiumWrapper(gymnasium.make(FLAGS.sparse_env).unwrapped), FLAGS.max_traj_length)
# 'FLAGS.sparse_env'를 통해 지정된 강화학습 환경을 생성하고 'unwrapped'를 호출하여 기본 환경에 접근
# 생성된 환경을 'GymnasiumWrapper'로 감싸 추가적인 기능 적용
# 'TrajSampler' 객체를 초기화, 최대 trajectory 길이는 'FLAGS.max_traj_length'로 설정
```

Wrap_gym 함수 OpenAI Gym 환경(env)에 추가적인 wrapper를 적용하는 과정

```
def wrap_gym(env: gym.Env, rescale_actions: bool = True) -> gym.Env:
    env = SinglePrecision(env) # 환경의 데이터 타입을 단일 정밀도(float32)로 설정.
    env = UniversalSeed(env)  # 환경에 일관된 난수 seed를 적용하여 재현 가능하게 만듭니다.
    env.action_space = _convert_space(copy.deepcopy(env.action_space)) # 액션 공간을 변환
    if rescale_actions:
        env = gym.wrappers.RescaleAction(env, -1, 1) # 액션 값을 -1~1로 재조정

    if isinstance(env.observation_space, gym.spaces.Dict):
        env = FlattenObservation(env) # 관측 공간이 딕셔너리 타입일 경우 평탄화하여 간소화

    env = gym.wrappers.ClipAction(env) # 액션 값을 환경의 액션 공간 내로 제한

    return env
```

class TrajSampler(object):

- 강화학습 환경에서 여러 에피소드(trajectory)의 데이터를 샘플링하는 역할
- 주어진 정책(policy)에 따라 행동을 취하고, 그 결과로 관측값, 보상, 행동 등의 데이터를 수집

class GymnasiumWrapper():

- 주어진 OpenAI Gym 환경(env)을 wrapping, 추가적인 기능을 제공하기 위함.
- 강화학습에서 주로 사용되는 환경을 사용자의 목적에 맞게 변형하고, 특정 시나리오에 대한 처리를 수행할 수 있게 도움.

D4RLDataset 함수

D4RLDataset이라는 클래스를 정의하고 있으며, 이 클래스는 OpenAI Gym 환경을 사용하여 D4RL(Datasets for Deep Data-Driven Reinforcement Learning) 데이터셋을 로드하고, 해당 데이터를 전처리함.

```
class D4RLDataset(Dataset):
    def __init__(self, env: gym.Env, clip_to_eps: bool = True, eps: float = 1e-5): # D4RL 데이터셋을 로드, 데이터셋 로드에 실패할 경우 예비 메서드로 시도
        try:
            dataset_dict = d4rl.qlearning_dataset(env)
        except:
            dataset_dict = d4rl.dataset(env)

        if clip_to_eps: # 액션 값들을 epsilon 값으로 클리핑하여 데이터를 정규화
            lim = 1 - eps
            dataset_dict["actions"] = np.clip(dataset_dict["actions"], -lim, lim)

        dones = np.full_like(dataset_dict["rewards"], False, dtype=bool) # 'dones' 배열을 초기화, 기본적으로 모든 값은 False

        for i in range(len(dones) - 1): # 'dones' 배열을 업데이트, 연속된 관측값 사이의 큰 차이나 터미널 신호가 있는 경우 True로 설정
            if (
                np.linalg.norm(
                    dataset_dict["observations"][i + 1]
                    - dataset_dict["next_observations"][i]
                )
                > 1e-6
                or dataset_dict["terminals"][i] == 1.0
            ):
                dones[i] = True

        dones[-1] = True # 마지막 'done'은 항상 True

        dataset_dict["masks"] = 1.0 - dataset_dict["terminals"] # 'masks'는 'terminals'의 반대 값으로 설정되며, 'terminals'는 삭제
        del dataset_dict["terminals"]

        for k, v in dataset_dict.items():
            dataset_dict[k] = v.astype(np.float32)

        dataset_dict["dones"] = dones # 'dones' 정보를 데이터셋에 추가

        super().__init__(dataset_dict) # 상위 클래스의 생성자를 호출하여 데이터셋을 초기화
```

load agent

Expert file 불러오기

```
# load agents
expert_model_pkl_dir = FLAGS.expert_dir  # iql, rlpd는 offline 강화학습
if 'iql' in expert_model_pkl_dir:   # expert_model_pkl_dir 경로에 'iql' 문자열이 포함되어 있는지 확인
    saved_ckpt_expert = load_model(expert_model_pkl_dir)
    intervene_policy = get_iql_policy_from_model(eval_env, saved_ckpt_expert)
elif 'rlpd' in expert_model_pkl_dir:
    saved_ckpt_expert = load_model(expert_model_pkl_dir)
    intervene_policy = get_rlpd_policy_from_model(eval_env, saved_ckpt_expert)
else:
    saved_ckpt_expert = load_model(expert_model_pkl_dir)
    intervene_policy = get_policy_from_model(eval_env, saved_ckpt_expert)  # 로드된 모델 데이터를 사용하여 평가 환경(eval_env)에서 사용할 정책을 설정
```

load agent

Ground_truth_agent_dir이 있는 경우 알고리즘에 따라서 Agent 로드

```
if FLAGS.ground_truth_agent_dir != '':  # FLAGS.ground_truth_agent_dir가 비어 있지 않은 경우 실행
    if 'iql' in FLAGS.ground_truth_agent_dir:
        ground_truth_agent = load_model(FLAGS.ground_truth_agent_dir)['iql']  # 'iql' 모델을 로드하고, 해당 모델에서 'iql' 키에 해당하는 Agent 데이터 추출
        ground_truth_policy = IQLSamplerPolicy(ground_truth_agent.actor)  # 로드된 에이전트를 사용하여 IQLSamplerPolicy 객체를 생성
        ground_truth_agent_type = 'iql'
    elif 'sac' in FLAGS.ground_truth_agent_dir or 'bc' in FLAGS.ground_truth_agent_dir:
        ground_truth_agent = load_model(FLAGS.ground_truth_agent_dir)['sac']
        ground_truth_policy = SamplerPolicy(ground_truth_agent.policy, ground_truth_agent.train_params['policy'])
        ground_truth_agent_type = 'sac'
    elif 'rlpd' in FLAGS.ground_truth_agent_dir:
        ground_truth_agent = load_model(FLAGS.ground_truth_agent_dir)['rlpd']
        ground_truth_policy = RLPDSamplerPolicy(ground_truth_agent.actor)
        ground_truth_agent_type = 'rlpd'
    else:
        raise ValueError("agent type not supported")
else:
    ground_truth_agent = FLAGS.ground_truth_agent_dir  # Ground_truth_agent_dir 비어 있는 경우 실행
    ground_truth_agent_type = ''


kwargs = dict(FLAGS.config)
model_cls = kwargs.pop("model_cls")
agent = globals()[model_cls].create(  # agent 생성
    FLAGS.seed, env.observation_space, env.action_space, **kwargs
)
```

class IQLSamplerPolicy(object):

- Implicit Q-Learning(IQL)에서 사용되는 액터 기반의 정책 샘플러를 구현
- 주어진 관측값에 기반하여 액션을 샘플링하는 역할

class RLPDSamplerPolicy(object):

- 액터 모델을 사용하여 주어진 관측값에 기반한 액션을 샘플링하는 역할
- 강화학습에서의 결정 과정을 구현

class SamplerPolicy(object):

- 강화학습에서 사용할 수 있는 일반적인 정책 샘플러(policy sampler)
- 주어진 정책(policy)을 사용하여 관측값(observation)을 기반으로 액션을 샘플링
- 정책의 매개변수를 업데이트하는 기능을 수행

Dataset

```
kwargs = dict(FLAGS.config) # FLAGS.config에서 설정된 모든 구성을 딕셔너리로 변환
model_cls = kwargs.pop("model_cls")  # 'model_cls' 키를 사용하여 모델 클래스 이름을 kwargs 딕셔너리에서 추출하고 제거
agent = globals()[model_cls].create(  # globals() 함수를 사용하여 현재 전역 심볼 테이블에서 model_cls 이름에 해당하는 클래스를 찾음
    FLAGS.seed, env.observation_space, env.action_space, **kwargs  # 찾은 클래스의 create 메소드를 호출하여 새로운 에이전트 인스턴스를 생성
)

if FLAGS.dataset_dir != '':  # FLAGS.dataset_dir 데이터셋 경로가 지정되어 있는 경우
        with open(FLAGS.dataset_dir, 'rb') as handle:
            dataset = pickle.load(handle)  # pickle을 사용하여 데이터셋 로드
else:
    dataset = get_d4rl_dataset(env)  # D4RL 라이브러리를 사용하여 환경에 맞는 데이터셋을 로드
```

데이터셋 조정

```
    dataset['actions'] = np.clip(dataset['actions'], -0.999, 0.999) # 행동 데이터를 -0.999와 0.999 사이로 클리핑
    dataset['rewards'] = np.zeros_like(dataset['rewards'])
    dataset['masks'] = 1 - dataset['dones']  # mask: 에피소드가 계속 되는지 여부 (에피소드가 종료되지 않은 상태를 1로 표시)
```

Replay Buffer

```
replay_buffer = ReplayBuffer(
    env.observation_space, env.action_space, FLAGS.max_steps
)
replay_buffer.seed(FLAGS.seed)
```

```
for i in range(len(dataset['rewards'])): # 데이터셋의 보상 길이만큼 반복하여 각 스텝의 데이터를 재생 버퍼에 삽입
    replay_buffer.insert(
        dict(
            observations=dataset['observations'][i],
            actions=dataset['actions'][i],
            rewards=0,
            masks=dataset['masks'][i],
            dones=dataset['dones'][i],
            next_observations=dataset['next_observations'][i],
        )
    )
```

def get_d4rl_dataset(env):

```
def get_d4rl_dataset(env):
    dataset = d4rl.qlearning_dataset(env)                # 주어진 환경(env)에 대해 D4RL 데이터셋 로드
    return dict(
        observations=dataset['observations’],            # 관측값
        actions=dataset['actions’],                      # 행동
        next_observations=dataset['next_observations’], # 다음 관측값
        rewards=dataset['rewards’],                      # 보상
        dones=dataset['terminals'].astype(np.float32),    # 에피소드 종료 여부 (bool에서 float32 형태로 변환)
    )
```

class ReplayBuffer(Dataset):

- 강화학습에서 중요한 구성 요소인 재생 버퍼(replay buffer) 구현
- 에이전트가 경험한 상태, 행동, 보상, 다음 상태 및 완료 여부를 저장
- 경험을 임의로 추출하여 학습을 안정화시키고 효율을 높이는 데 사용

에이전트 오프라인 강화학습

```
for i in tqdm.tqdm( # 프로그레스 바 생성
    range(0, FLAGS.pretrain_steps), smoothing=0.1, disable=not FLAGS.tqdm
):
    offline_batch = ds.sample(FLAGS.batch_size * FLAGS.utd_ratio)
    batch = {}
    for k, v in offline_batch.items(): # 샘플링된 배치에서 각 키(k)와 값(v)을 추출하여 배치 딕셔너리에 저장
        batch[k] = v
        if "antmaze" in FLAGS.env_name and k == "rewards":
            batch[k] -= 1

    agent, update_info = agent.update(batch, FLAGS.utd_ratio) # 에이전트를 업데이트하고 업데이트 정보를 반환

    if i % FLAGS.log_interval == 0:
        for k, v in update_info.items():
            wandb.log({f"offline-training/{k}": v}, step=i)

    if i % FLAGS.eval_interval == 0:# 에이전트 평가
        eval_info = evaluate(agent, eval_env, num_episodes=FLAGS.eval_episodes)  # eval_env 환경에서 FLAGS.eval_episodes 만큼 에이전트 평가

        for k, v in eval_info.items():# 평가 결과를 로깅
            wandb.log({f"offline-evaluation/{k}": v}, step=i)

        sampler_policy = RLPDSamplerPolicy(agent.actor)  # RLPDSamplerPolicy을 사용하여 에이전트의 행동 정책 설정
        sparse_trajs = sparse_eval_sampler.sample(  # policy을 사용하여 특정 에피소드 수만큼 Trajectory를 샘플링
                sampler_policy,
                FLAGS.eval_episodes, deterministic=False
            )
        avg_success = evaluate_policy(sparse_trajs,  # 샘플링된 Trajectory를 평가하여 평균 성공률을 계산
                                      success_rate=True,
                                      success_function=lambda t: np.all(t['rewards'][-1:]>=10),
                                      )
        wandb.log({f"offline-evaluation/avg_success": avg_success}, step=i)  # 평균 성공률 로깅
```

학습 전 초기화

```
all_observations = []
all_actions = []
all_rewards = []
all_masks = [] # 에피소드가 종료되지 않으면 1, 종료되었을 때 0을 저장하는 리스트
all_dones = [] # 에피소드의 종료 여부를 저장하는 리스트
all_next_observations = []
all_intervene = [] # 개입을 저장하는 리스트

observation, done = env.reset(), False
t = 0
intervene = False
prev_intervene = False  # 이전 스텝 개입 여부
stop_intervene_time = -1
first_intervene_action_mask = []
for i in tqdm.tqdm(  # tqdm를 사용해 반복 과정을 시각화
    range(1, FLAGS.max_steps + 1), smoothing=0.1, disable=not FLAGS.tqdm
):
```

에이전트의 행동 선택과 개입

```
policy_action, agent = agent.sample_actions(observation) # 에이전트의 행동 샘플링

expert_action = intervene_policy(observation.reshape(1, -1), deterministic=False).reshape(-1)
ground_truth_action = ground_truth_policy(observation.reshape(1, -1), deterministic=False).reshape(-1)

if 'ref' in FLAGS.intervention_strategy: # Flag.intervention strategy ref가 포함되어 있으면 expert action 그렇지 않으면 ground_truth
    reference_action = expert_action
else:
    reference_action = ground_truth_action  # 실행할 때 intervention_strategy parse 빈칸으로 두면 ground_truth_agent_dir 사용
```

Agent의 q-value, q-function

```
if not intervene: # intervene가 False인 경우 실행
    if ground_truth_agent_type == 'iql':  # iql = Independent Q-Learning agent
        gt_q1, gt_q2 = ground_truth_agent.critic(observation, reference_action)
        gt_q = np.min([gt_q1, gt_q2]) # ground truth agenet에서 두 개의 q값을 구하고 작은 것을 선택
        policy_q1, policy_q2 = ground_truth_agent.critic(observation, policy_action)

        policy_q = np.min([policy_q1, policy_q2])
    elif ground_truth_agent_type == 'sac':
        gt_q1 = ground_truth_agent.qf.apply(ground_truth_agent.train_params['qf1'], observation, reference_action)
        gt_q2 = ground_truth_agent.qf.apply(ground_truth_agent.train_params['qf2'], observation, reference_action)
        gt_q = np.min([gt_q1, gt_q2])

        policy_q1 = ground_truth_agent.qf.apply(ground_truth_agent.train_params['qf1'], observation, policy_action)
        policy_q2 = ground_truth_agent.qf.apply(ground_truth_agent.train_params['qf2'], observation, policy_action)
        policy_q = np.min([policy_q1, policy_q2])
    else:
        gt_qs = ground_truth_agent.critic.apply_fn(
            {"params": ground_truth_agent.critic.params},
            observation,
            reference_action,
            True,
        )
        gt_q = gt_qs.mean(axis=0)

        policy_qs = ground_truth_agent.critic.apply_fn(
            {"params": ground_truth_agent.critic.params},
            observation,
            policy_action,
            True,
        )
        policy_q = policy_qs.mean(axis=0)
```

실제 agnet의 학습

```
        if policy_q < gt_q * FLAGS.intervene_threshold: # 실제 policy의 q값과 Agent의 q값을 비교하여 개입 여부를 결정
            intervene = np.random.choice([0, 1], p=[0.05, 1-0.05])
        else:
            intervene = np.random.choice([0, 1], p=[1-0.05, 0.05])

        intervene = bool(intervene) # 개입 여부 결정

        if intervene: # 개입이 종료되는 시점 정의
            stop_intervene_time = t + FLAGS.intervene_n_steps

    if t == stop_intervene_time: # 개입 종료
        intervene = False
```

실제 agnet의 학습

```
if intervene:
    if t != 0 and not prev_intervene:  # t가 0이 아니고 이전 개입이 False였을 경우
        # append state action pair that led to previous intervention
        first_intervene_action_mask[-1] = 1

        replay_buffer.insert(    # 개입이 발생한 시점에서의 상태, 행동, 보상 등을 replay버퍼에 추가
           dict(
                observations=all_observations[-1],
                actions=all_actions[-1],
                rewards=-1,
                masks=all_masks[-1],
                dones=all_dones[-1],
                next_observations=all_next_observations[-1],
            )
        )
    if 'label' in FLAGS.intervention_strategy:  # 개입 전략에 따라 행동을 선택
        action = policy_action
    else:
        action = expert_action
else:  # 개입이 발생하지 않은 경우
    action = policy_action

    if t != 0:
        replay_buffer.insert(
           dict(
                observations=all_observations[-1],
                actions=all_actions[-1],
                rewards=0,
                masks=all_masks[-1],
                dones=all_dones[-1],
                next_observations=all_next_observations[-1],
            )
        )

next_observation, _, done, info = env.step(action)
```

학습 결과 저장

```
# 에피소드의 종료 여부와 관련된 처리 및 정보를 저장
  if not done or "TimeLimit.truncated" in info: # 에피소드 종료 여부 확인
      mask = 1.0
  else:
      mask = 0.0

  prev_intervene = intervene # 이전 개입 여부 기록
  all_observations += [observation] # 에피소드 정보 기록
  all_actions += [action]
  all_rewards += [0]
  all_masks += [mask]
  all_dones += [done]
  all_next_observations += [next_observation]
  first_intervene_action_mask.append(0)
  all_intervene += [intervene]
  t += 1  # 시간 스텝 증가 및 다음 관측 설정

  observation = next_observation  # 다음 관측을 현재 관측으로 업데이트
```

학습 결과 저장

```
if done or t > FLAGS.max_traj_length:  # 에피소드가 종료되었거나 최대 시간 스텝에 도달했을 때
    observation, done = env.reset(), False  # 환경을 reset하고 새로운 에피소드 시작
    intervene = False # 초기화
    prev_intervene = False
    stop_intervene_time = -1
    t = 0
    try:  # 에피소드 정보 로깅
        for k, v in info["episode"].items():
            decode = {"r": "return", "l": "length", "t": "time"}
            wandb.log({f"training/{decode[k]}": v}, step=i + FLAGS.pretrain_steps)
    except:
        pass

online_batch = replay_buffer.sample(  # 온라인 데이터와 오프라인 데이터 배치 샘플링
    int(FLAGS.batch_size * FLAGS.utd_ratio * (1 - FLAGS.offline_ratio))
)
offline_batch = ds.sample(
    int(FLAGS.batch_size * FLAGS.utd_ratio * FLAGS.offline_ratio)
)

batch = combine(offline_batch, online_batch)  # 온라인 배치와 오프라인 배치를 결합하여 전체 배치 생성
```

학습 결과 저장

```
if "antmaze" in FLAGS.env_name: # 환경이 antmaze일 경우 보상 수정
    batch["rewards"] -= 1

agent, update_info = agent.update(batch, FLAGS.utd_ratio)

if i % FLAGS.log_interval == 0:
    for k, v in update_info.items():
        wandb.log({f"training/{k}": v}, step=i + FLAGS.pretrain_steps)

if i % FLAGS.eval_interval == 0:
    eval_info = evaluate(
        agent,
        eval_env,
        num_episodes=FLAGS.eval_episodes,
        save_video=FLAGS.save_video,
    )

    for k, v in eval_info.items():
        wandb.log({f"evaluation/{k}": v}, step=i + FLAGS.pretrain_steps)

    wandb.log({f"evaluation/intervene_rate": np.mean(all_intervene[-FLAGS.eval_interval+1:])}, step=i + FLAGS.pretrain_steps)

    sampler_policy = RLPDSamplerPolicy(agent.actor)
    sparse_trajs = sparse_eval_sampler.sample(
            sampler_policy,
            FLAGS.eval_episodes, deterministic=False
        )
    avg_success = evaluate_policy(sparse_trajs,
                                  success_rate=True,
                                  success_function=lambda t: np.all(t['rewards'][-1:]>=10),
                                  )
    wandb.log({f"evaluation/avg_success": avg_success}, step=i + FLAGS.pretrain_steps)
```

학습 결과 저장

```
if FLAGS.checkpoint_model:
    try:
        checkpoints.save_checkpoint(
            chkpt_dir, agent, step=i, keep=20, overwrite=True
        )
    except:
        print("Could not save model checkpoint.")

if FLAGS.checkpoint_buffer:
    try:
        with open(os.path.join(buffer_dir, f"buffer"), "wb") as f:
            pickle.dump(replay_buffer, f, pickle.HIGHEST_PROTOCOL)
    except:
        print("Could not save agent buffer.")

if FLAGS.save_model:
    save_data = {'rlpd': agent}
    with open(os.path.join(model_dir, "model.pkl"), 'wb') as fout:
        pickle.dump(save_data, fout)
```