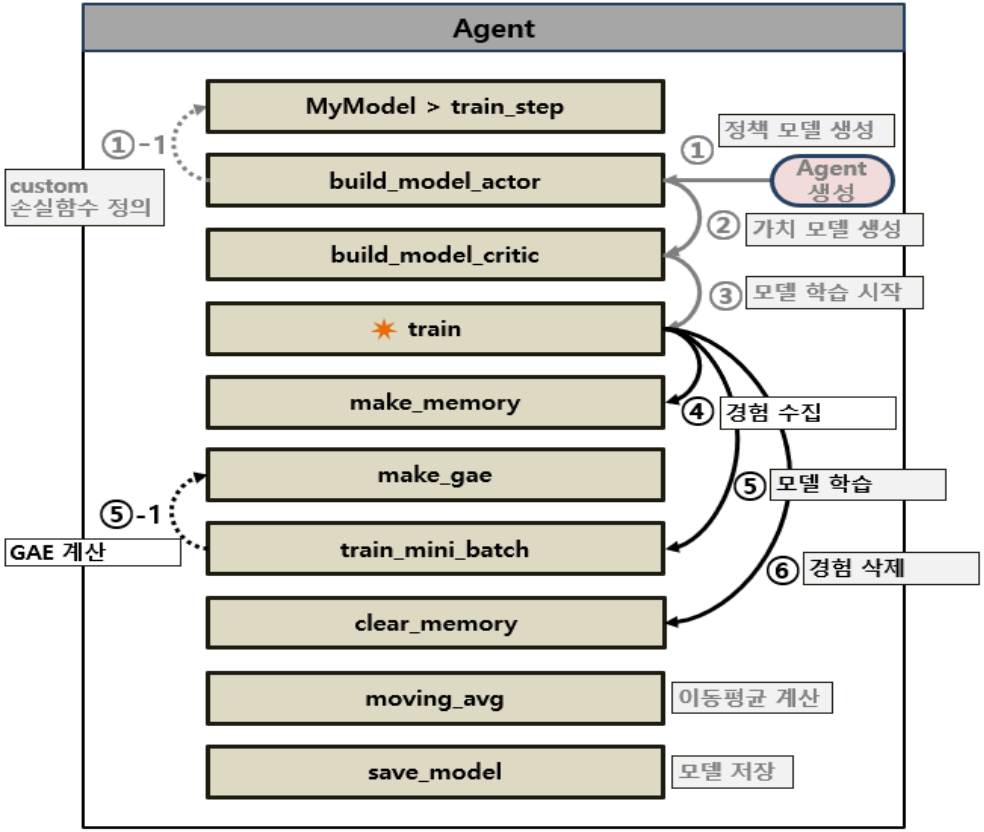
09

PPO 알고리즘

2. PPO 프로그래밍

PPO 프로그래밍

프로그램 구조



PPO 프로그래밍

전체 코드 리뷰

코드 리뷰

PPO 프로그래밍 코드분석

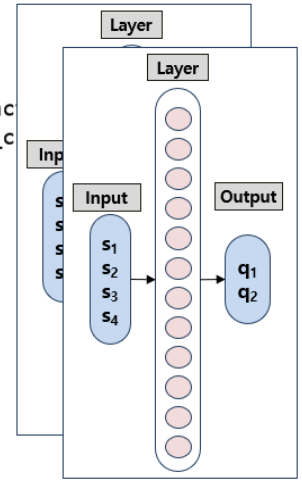
Agent 클래스 속성

프로그램 동작 설정

```
self.env = gym.make('CartPole-v1')
self.state_size = self.env.observation_space.shape[0]
self.action_size = self.env.action_space.n
self.value_size = 1
```

(1) 모델 설정

```
self.node_num = 24
self.learning_rate_actor = 0.0005
self.learning_rate_critic = 0.0005
self.epochs_cnt = 5
self.model_actor = self.build_model_ac
self.model_critic = self.build_model_c
```



(2) 학습 설정

```
self.discount_rate = 0.98
self.smooth_rate = 0.95
self.penalty = -400
```

반복 설정

```
self.episode_num = 500
self.mini_batch_step_size = 32
```

학습 모니터링 설정

```
self.moving_avg_size = 20
self.reward_list= []
self.count_list = []
self.moving_avg_list = []
```

데이터 수집 환경

```
self.states, self.states_next, self.action_matrixs = [],[],[]
self.dones, self.action_probs, self.rewards = [],[],[]
self.DUMMY_ACTION_MATRIX = np.zeros((1,1,self.action_size))
self.DUMMY_ADVANTAGE = np.zeros((1,1,self.value_size))
```

PPO 프로그래밍 코드분석

MyModel 클래스

```
LOSS_CLIPPING = 0.1                                  (1) loss clipping 파라미터 설정
class MyModel(tf.keras.Model):                       MyModel 클래스 정의(Model 함수 상속)
    def train_step(self, data):                      train_step 함수 재정의

입력 변수 설정
        in_datas, out_action_probs = data
        states, action_matrixs, advantages = in_datas[0], in_datas[1], in_datas[2]

GradientTape 설정
        with tf.GradientTape() as tape:
행동예측
            y_pred = self(states, training=True)
(2) 신규정책 확률계산
            new_policy = K.max(action_matrixs*y_pred, axis=-1)
과거정책 확률계산
            old_policy = K.max(action_matrixs*out_action_probs, axis=-1)
(3) 정책확률 비율계산
            r = new_policy/(old_policy)
(4) 정책확률 클립핑
            clipped = K.clip(r, 1-LOSS_CLIPPING, 1+LOSS_CLIPPING)
(5) 비용함수 계산
            loss = -K.minimum(r*advantages, clipped*advantages)

모델 가중치
        trainable_vars = self.trainable_variables
gradient 호출
        gradients = tape.gradient(loss, trainable_vars)
변수에 gradient 적용
        self.optimizer.apply_gradients(zip(gradients, trainable_vars))
```

PPO 프로그래밍 코드분석

train 함수

```
def train(self):
  for episode in range(self.episode_num):
    state = self.env.reset()
    self.env.max_episode_steps = 500
    count, reward_tot = self.make_memory(episode, state)
    self.train_mini_batch()
    self.clear_memory()

    if count < 500:
        reward_tot = reward_tot-self.penalty

    self.reward_list.append(reward_tot)
    self.count_list.append(count)
    self.moving_avg_list.append(self.moving_avg(self.count_list,self.moving_avg_size))

    if(episode % 10 == 0):
        print("episode:{}, moving_avg:{}, rewards_avg:{}"
         .format(episode, self.moving_avg_list[-1], np.mean(self.reward_list)))
  self.save_model()
```

(1) 최대 실행횟수 설정

(2) 경험수집

(3) 모델학습

경험삭제

결과 저장

이동평균

실행로그

모델저장

PPO 프로그래밍

make_memory 함수

```
def make_memory(self, episode, state):
  reward_tot = 0
  count = 0
  reward = np.zeros(self.value_size)
  advantage = np.zeros(self.value_size)
  target = np.zeros(self.value_size)
  action_matrix = np.zeros(self.action_size)
  done = False

  while not done:
    count+=1
    state_t = np.reshape(state,[1, 1, self.state_size])
    action_matrix_t = np.reshape(action_matrix,[1, 1, self.action_size])

    action_prob = self.model_actor.predict([state_t, self.DUMMY_ACTION_MATRIX,
                                            self.DUMMY_ADVANTAGE])
    action = np.random.choice(self.action_size, 1, p=action_prob[0][0])[0]
    action_matrix = np.zeros(self.action_size) #초기화
    action_matrix[action] = 1
    state_next, reward, done, none = self.env.step(action)

    state_next_t = np.reshape(state_next,[1, 1, self.state_size])

    if count < 500 and done:
      reward = self.penalty

    self.states.append(np.reshape(state_t, [1,self.state_size]))
    self.states_next.append(np.reshape(state_next_t, [1,self.state_size]))
    self.action_matrixs.append(np.reshape(action_matrix, [1,self.action_size]))
    self.dones.append(np.reshape(0 if done else 1, [1,self.value_size]))
    self.action_probs.append(np.reshape(action_prob, [1,self.action_size]))
    self.rewards.append(np.reshape(reward, [1,self.value_size]))

    if(count % self.mini_batch_step_size == 0):
      self.train_mini_batch()
      self.clear_memory()
    reward_tot += reward
    state = state_next

return count, reward_tot
```

행동 예측

행동 선택

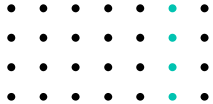
매트릭 생성

경험 저장

(1) 종료변수 설정

(2) 모델학습

PPO 프로그래밍 코드분석



make_gae 함수

```
def make_gae(self, values, values_next, rewards, dones):
  delta_adv, delta_tar, adv, target = 0, 0, 0, 0
  advantages = np.zeros(np.array(values).shape)
  targets = np.zeros(np.array(values).shape)

  for t in reversed(range(0, len(rewards))):
    delta_adv = rewards[t] + self.discount_rate * values_next[t] * dones[t] - values[t]
    delta_tar = rewards[t] + self.discount_rate * values_next[t] * dones[t]
    adv = delta_adv + self.smooth_rate*self.discount_rate * dones[t] * adv
    target = delta_tar + self.smooth_rate*self.discount_rate * dones[t] * target

    advantages[t] = adv
    targets[t] = target

  return advantages, targets
```

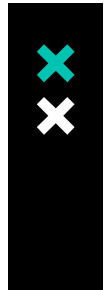
(1) 마지막 인덱스부터 반복

(2)스텝별 어드밴티지 계산

(3)스텝별 타겟 계산

(4)할인된 어드밴티지 계산

(5)할인된 타겟 계산



PPO 프로그래밍 코드분석

train_mini_batch 함수

```
def train_mini_batch(self):

  if len(self.states) == 0:
    return

  states_t = np.array(self.states)
  states_next_t = np.array(self.states_next)
  action_matrixs_t = np.array(self.action_matrixs)
  action_probs_t = np.array(self.action_probs)
  rewards_t = np.array(self.rewards)

  values = self.model_critic.predict(states_t)
  values_next = self.model_critic.predict(states_next_t)

  advantages, targets = self.make_gae(values, values_next, self.rewards, self.dones)
  advantages_t = np.array(advantages)
  targets_t = np.array(targets)
  self.model_actor.fit([states_t, action_matrixs_t, advantages_t],
                       [action_probs_t], epochs=self.epochs_cnt, verbose=0)
  self.model_critic.fit(states_t, targets_t, epochs=self.epochs_cnt, verbose=0)
```

넘파이로 변경

(1) 가치 예측

(2) GAE 계산

모델 학습