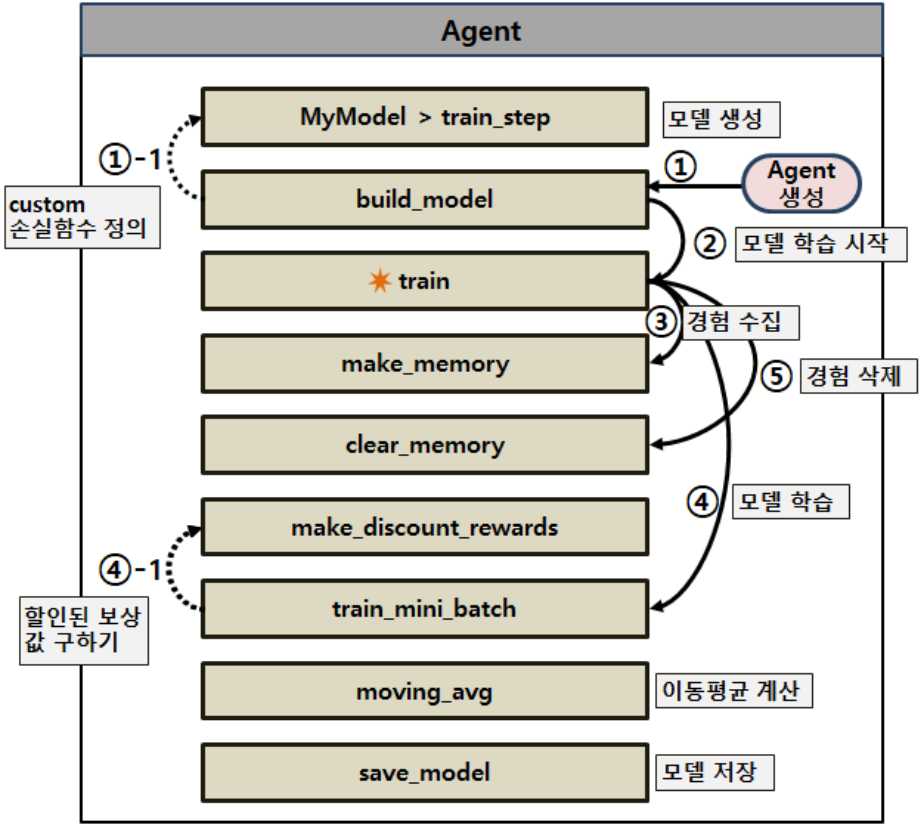
07

REINFORCE 알고리즘

2. 프로그래밍

프로그래밍

프로그램 구조



프로그래밍

전체 코드 리뷰

코드 리뷰

프로그래밍 코드분석

Agent 클래스 속성

(1) 프로그램 동작 설정

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

모델 설정

```
self.node_num = 12
self.learning_rate = 0.0005
self.epochs_cnt = 5
self.model = self.build_model()
```

학습 설정

```
self.discount_rate = 0.95
self.penalty = -10
```

반복 설정

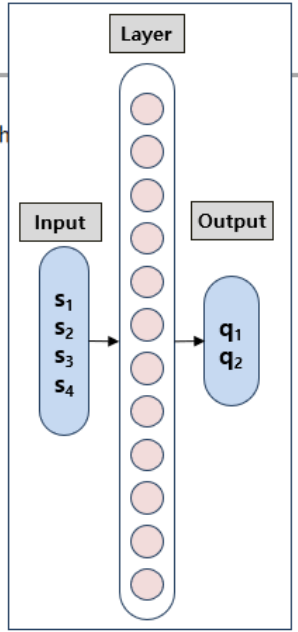
```
self.episode_num = 500
```

학습 모니터링 설정

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

(2) 데이터 수집 환경

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



프로그래밍 코드분석

MyModel 클래스

```
class MyModel(tf.keras.Model):                          (1) MyModel 클래스 정의(Model 함수 상속)
    def train_step(self, data):                         (2) train_step 함수 재정의
        in_datas, out_actions = data                    (3) 입력 변수 설정
        states, action_matrix, rewards = in_datas[0], in_datas[1], in_datas[2]

        with tf.GradientTape() as tape:                 (4) GradientTape 설정
            y_pred = self(states, training=True)        (5) 행동예측
            action_probs = K.sum(action_matrix*y_pred, axis=-1)   (6) 확률계산
            loss = -K.log(action_probs)*rewards         (7) 비용함수

        trainable_vars = self.trainable_variables       (8) 모델 가중치
        gradients = tape.gradient(loss, trainable_vars) (9) gradient 호출
        self.optimizer.apply_gradients(zip(gradients, trainable_vars))   (10) 변수에 gradient 적용
```

build_model : 모델생성

```
model = self.MyModel(inputs=[input_states, input_action_matrixs, input_rewards],
                     outputs=out_actions)
```

train_mini_batch : 모델학습

```
self.model.fit(x=[states_t, action_matrixs_t, discount_rewards_t], y=[action_probs_t],
               epochs=self.epochs_cnt, verbose=0)
```

프로그래밍 코드분석

GradientTape 동작 방식

- with 지정자를 통해 GradientTape 클래스 사용을 선언하고, watch 함수를 통해 동작을 모니터링해야 하는 변수를 지정하면, 모든 연산이 GradientTape 클래스에 기록된다

```
import tensorflow as tf
import tensorflow as tf
import tensorflow.keras.backend as K
import numpy as np
x = tf.constant([1.0, 2.0, 3.0])
with tf.GradientTape() as tape:
    tape.watch(x) #동작 모니터링을 위한 설정
    y = (x*x)      #x2
z = tape.gradient(y, x)
print(z)
```

```
tf.Tensor([2. 4. 6.], shape=(3,), dtype=float32)
```

프로그래밍 코드분석

확률적 정책 선택

- 확률적(Stochastic) 정책결정 방식에서는 정책[0.6, 0.4]일 경우 0번째 정책이 60% 확률로 선택되고 1번째 확률이 40% 확률로 선택되도록 한다

```
import tensorflow as tf
import tensorflow.keras.backend as K
import numpy as np

y_pred = np.array([[0.6,0.4],
                   [0.3,0.7]])
action_matrix = np.array([[1,0],
                          [0,1]])

action_probs = K.sum(action_matrix*y_pred, axis=-1)
print("*action_probs:", action_probs)
```

```
*action probs: tf.Tensor([0.6 0.7], shape=(2,), dtype=float64)
```

프로그래밍 코드분석

zip 함수

- zip 함수는 동일한 개수의 여러 데이터를 하나의 자료구조로 묶어주는 역할을 하는 함수이다.

```
b= zip([1, 2, 3], [4, 5, 6])
c = list(b)
print(c)
```

```
[(1, 4), (2, 5), (3, 6)]
```

프로그래밍 코드분석

build_model 함수

(1) 입력 값 설정

(2) 네트워크 구성

(3) 모델 생성

```
def build_model(self):
        input_states = Input(shape=(1,self.state_size), name='input_states')
        input_action_matrixs = Input(shape=(1,self.action_size),
                                     name='input_action_matrixs')
        input_rewards = Input(shape=(1,self.value_size), name='input_rewards')

        x = (input_states)
        x = Dense(self.node_num, activation='relu')(x)
        out_actions = Dense(self.action_size, activation='softmax', name='output')(x)

        model = self.MyModel(inputs=[input_states, input_action_matrixs, input_rewards],
                             outputs=out_actions)

        model.compile(optimizer="adam")

        model.summary()
        return model
```

프로그래밍 코드분석

train 함수

(1) 데이터 수집
(2) 모델학습

```
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()
```

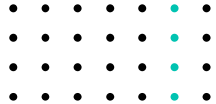
(3) 데이터 삭제

프로그래밍 코드분석

make_memory 함수

```
def make_memory(self, episode, state):
    reward_tot = 0
    count = 0
    reward = 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])

(1) 행동 예측   action_prob = self.model.predict([state_t, self.DUMMY_ACTION_MATRIX, self.DUMMY_REWARD])

(2) 행동 선택   action = np.random.choice(self.action_size, 1, p=action_prob[0][0])[0]
(3) 매트릭 생성  action_matrix = np.zeros(self.action_size)
                action_matrix[action] = 1

        state_next, reward, done, none = self.env.step(action)

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

        self.states.append(np.reshape(state_t, [1,self.state_size]))
        self.action_matrixs.append(np.reshape(action_matrix, [1,self.action_size]))
        self.action_probs.append(np.reshape(action_prob, [1,self.action_size]))
        self.rewards.append(reward)
        reward_tot += reward
        state = state_next
    return count, reward_tot
```

프로그래밍 코드분석



random.choice 함수

- 확률적으로 행동을 선택하기 위해 넘파이(numpy)에서 제공하는 random.choice 함수를 사용한다

```
import numpy as np
acton_prob = [0.7, 0.3]
for i in range(10):
    d = np.random.choice(a=2, size=1, p=acton_prob)[0]
    print(d, end=', ')
```

```
1, 1, 0, 0, 0, 0, 1, 0, 1, 0,
```

프로그래밍 코드분석

clear_memory 함수

- 에피소드가 끝나고 저장된 데이터를 활용해 모델을 학습하면 새로운 경험을 저장하기 위해 데이터를 삭제해야 한다

```
def clear_memory(self):
    self.states, self.action_matrixs, self.action_probs, self.rewards = [],[],[],[]
```

프로그래밍 코드분석

make_discount_rewrads 함수

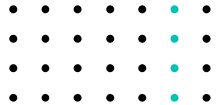
- 함수는 카트폴 실행 시점 별로 할인된 반환값을 계산하는 함수

```
def make_discount_rewards(self, rewards):
    discounted_rewards = np.zeros(np.array(rewards).shape)
    running_add = 0
    for t in reversed(range(0, len(rewards))):
        running_add = running_add * self.discount_rate + rewards[t]
        discounted_rewards[t] = running_add

    return discounted_rewards
```

(1) 마지막 인덱스부터 반복

(2) 할인된 반환 값 계산

프로그래밍 코드분석

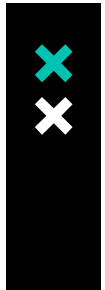


reversed 함수

- 내장 함수인 revered를 사용해서 마지막 인덱스부터 하나씩 꺼내 반환 값을 계산한다

```
a = [1,2,3,4,5]
for t in reversed(range(0, len(a))):
    print(t, end=', ')
```

```
4, 3, 2, 1, 0,
```



프로그래밍 코드분석

train_mini_batch 함수

- train_mini_batch 함수는 수집된 데이터를 활용해 모델을 학습하는 기능을 한다

(1) 할인된 반환 값 계산

(2) 데이터 모양 변경

(3) 리스트를 넘파이로 변경

(4) 모델 학습

```
def train_mini_batch(self):
    discount_rewards = np.array(self.make_discount_rewards(self.rewards))
    discount_rewards_t = np.reshape(discount_rewards, [len(discount_rewards),1,1])
    states_t = np.array(self.states)
    action_matrixs_t = np.array(self.action_matrixs)
    action_probs_t = np.array(self.action_probs)

    self.model.fit(x=[states_t, action_matrixs_t, discount_rewards_t],
                   y=[action_probs_t], epochs=self.epochs_cnt, verbose=0)
```