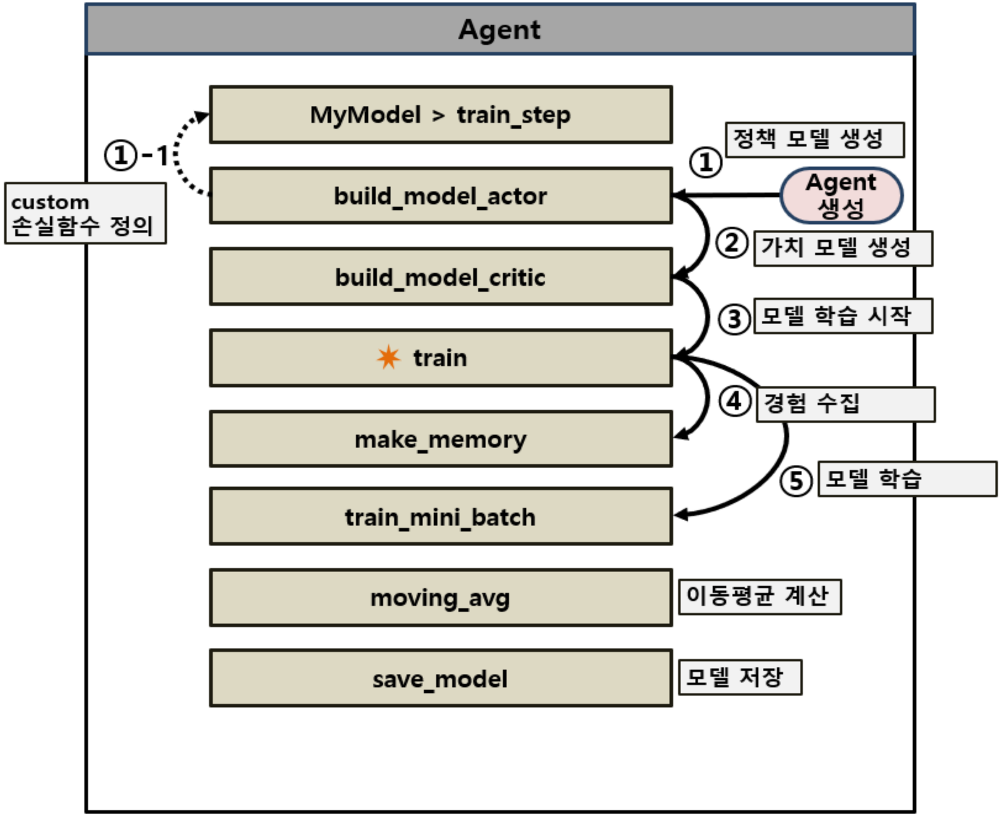
08

A2C 알고리즘

2. A2C 프로그래밍

A2C 프로그래밍

프로그램 구조



Agent

MyModel > train_step

build_model_actor

build_model_critic

✶ train

make_memory

train_mini_batch

moving_avg

save_model

①-1

custom 손실함수 정의

① 정책 모델 생성

Agent 생성

② 가치 모델 생성

③ 모델 학습 시작

④ 경험 수집

⑤ 모델 학습

이동평균 계산

모델 저장

A2C 프로그래밍

전체 코드 리뷰

코드 리뷰

A2C 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 = 12
self.learning_rate = 0.001
self.epochs_cnt = 1
self.model_actor = self.build_model_actor()
self.model_critic = self.build_model_critic()
```

학습 설정

```
self.discount_rate = 0.95
self.penalty = -20
```

반복 설정

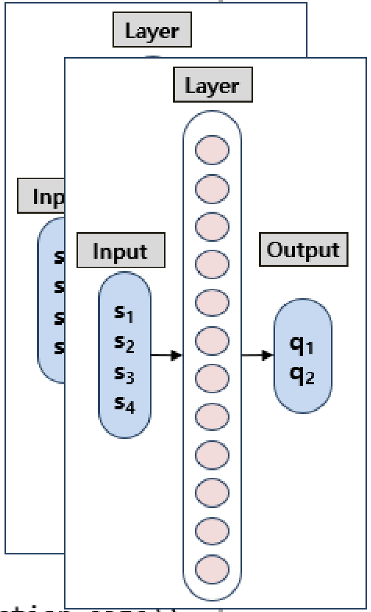
```
self.episode_num = 500
```

학습 모니터링 설정

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

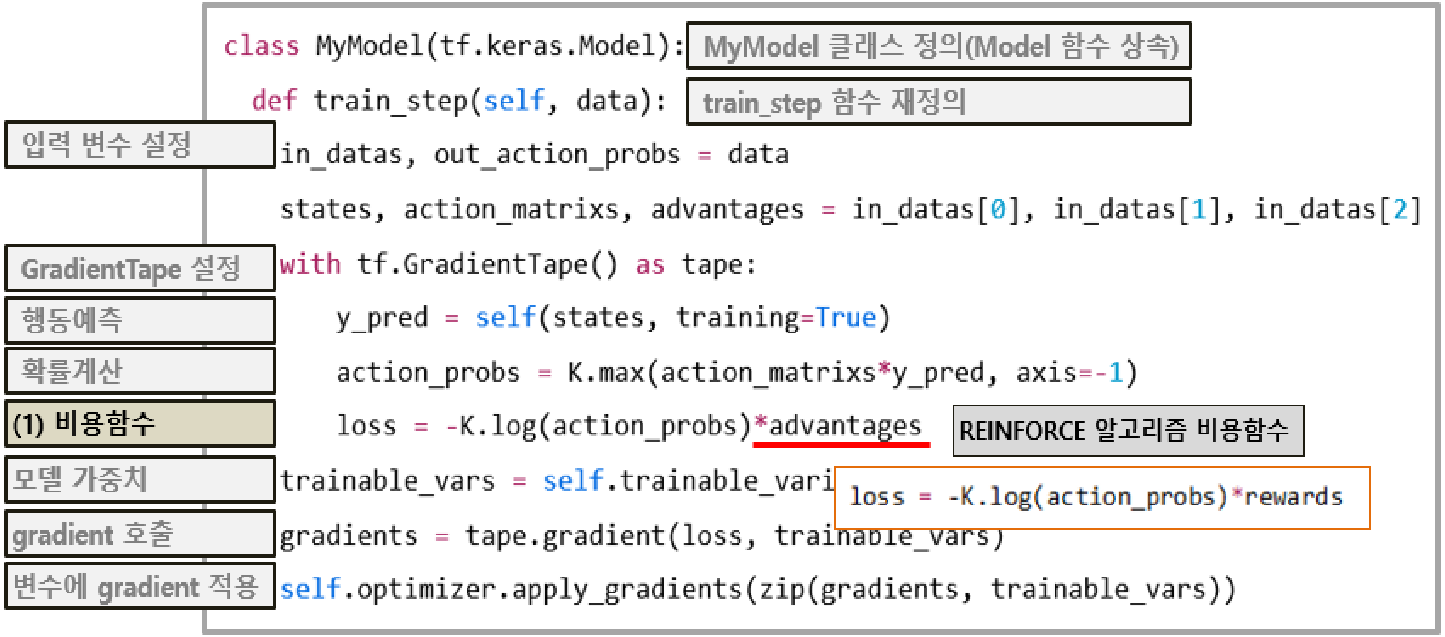
데이터 수집 환경

```
self.DUMMY_ACTION_MATRIX = np.zeros((1,self.action_size))
self.DUMMY_ADVANTAGE = np.zeros((1,self.value_size))
```



A2C 프로그래밍 코드분석

MyModel 클래스



```
class MyModel(tf.keras.Model):
  def train_step(self, data):
    in_datas, out_action_probs = data
    states, action_matrixs, advantages = in_datas[0], in_datas[1], in_datas[2]
    with tf.GradientTape() as tape:
      y_pred = self(states, training=True)
      action_probs = K.max(action_matrixs*y_pred, axis=-1)
      loss = -K.log(action_probs)*advantages
    trainable_vars = self.trainable_vari
    gradients = tape.gradient(loss, trainable_vars)
    self.optimizer.apply_gradients(zip(gradients, trainable_vars))
```

MyModel 클래스 정의(Model 함수 상속)

train_step 함수 재정의

REINFORCE 알고리즘 비용함수

```
loss = -K.log(action_probs)*rewards
```

입력 변수 설정

GradientTape 설정

행동예측

확률계산

(1) 비용함수

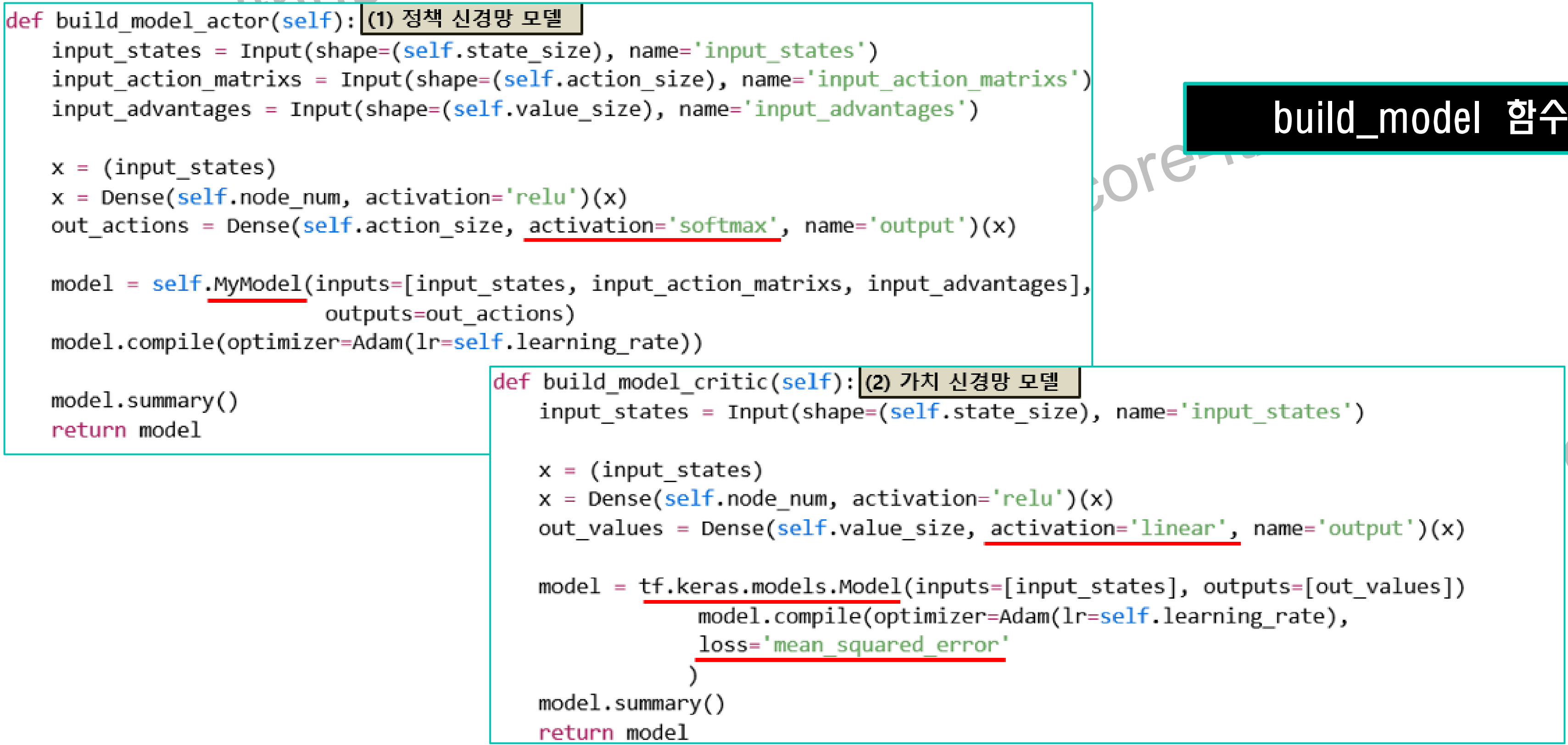
모델 가중치

gradient 호출

변수에 gradient 적용

A2C 프로그래밍 코드분석

build_model 함수



(1) 정책 신경망 모델

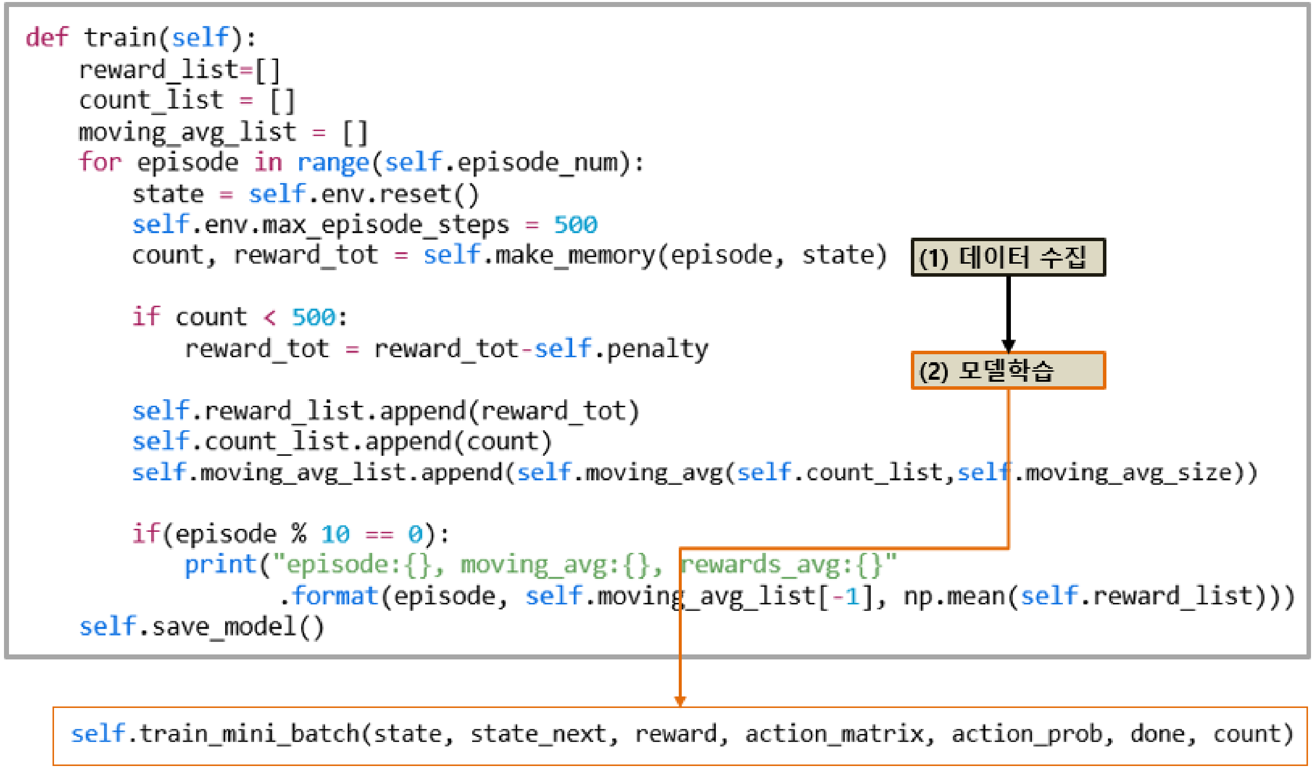
```
def build_model_actor(self):
    input_states = Input(shape=(self.state_size), name='input_states')
    input_action_matrixs = Input(shape=(self.action_size), name='input_action_matrixs')
    input_advantages = Input(shape=(self.value_size), name='input_advantages')

    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_advantages],
                         outputs=out_actions)
    model.compile(optimizer=Adam(lr=self.learning_rate))

    model.summary()
    return model
```

(2) 가치 신경망 모델

```
def build_model_critic(self):
    input_states = Input(shape=(self.state_size), name='input_states')

    x = (input_states)
    x = Dense(self.node_num, activation='relu')(x)
    out_values = Dense(self.value_size, activation='linear', name='output')(x)

    model = tf.keras.models.Model(inputs=[input_states], outputs=[out_values])
                model.compile(optimizer=Adam(lr=self.learning_rate),
                loss='mean_squared_error'
                )
    model.summary()
    return model
```

A2C 프로그래밍 코드분석

train 함수

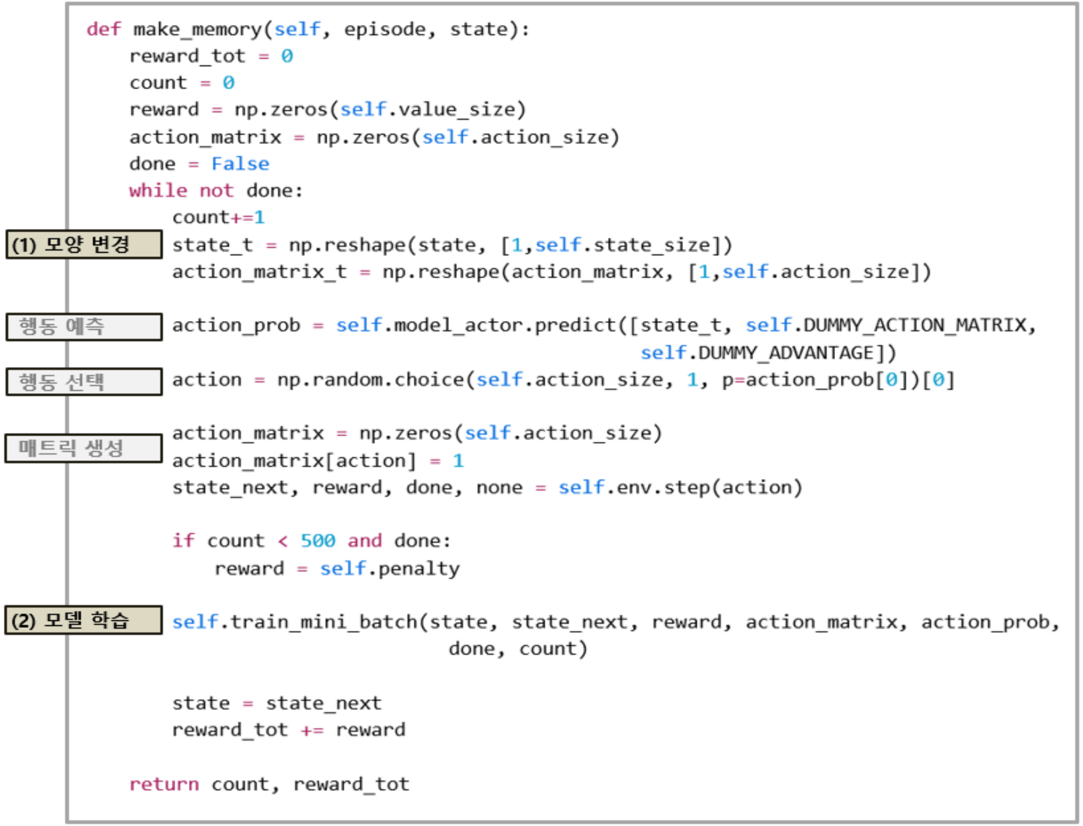


```python
def train(self):
    reward_list=[]
    count_list = []
    moving_avg_list = []
    for episode in range(self.episode_num):
        state = self.env.reset()
        self.env.max_episode_steps = 500
        count, reward_tot = self.make_memory(episode, state)

        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()
```

```python
self.train_mini_batch(state, state_next, reward, action_matrix, action_prob, done, count)
```

A2C 프로그래밍 코드분석

make_memory 함수



(1) 모양 변경

행동 예측

행동 선택

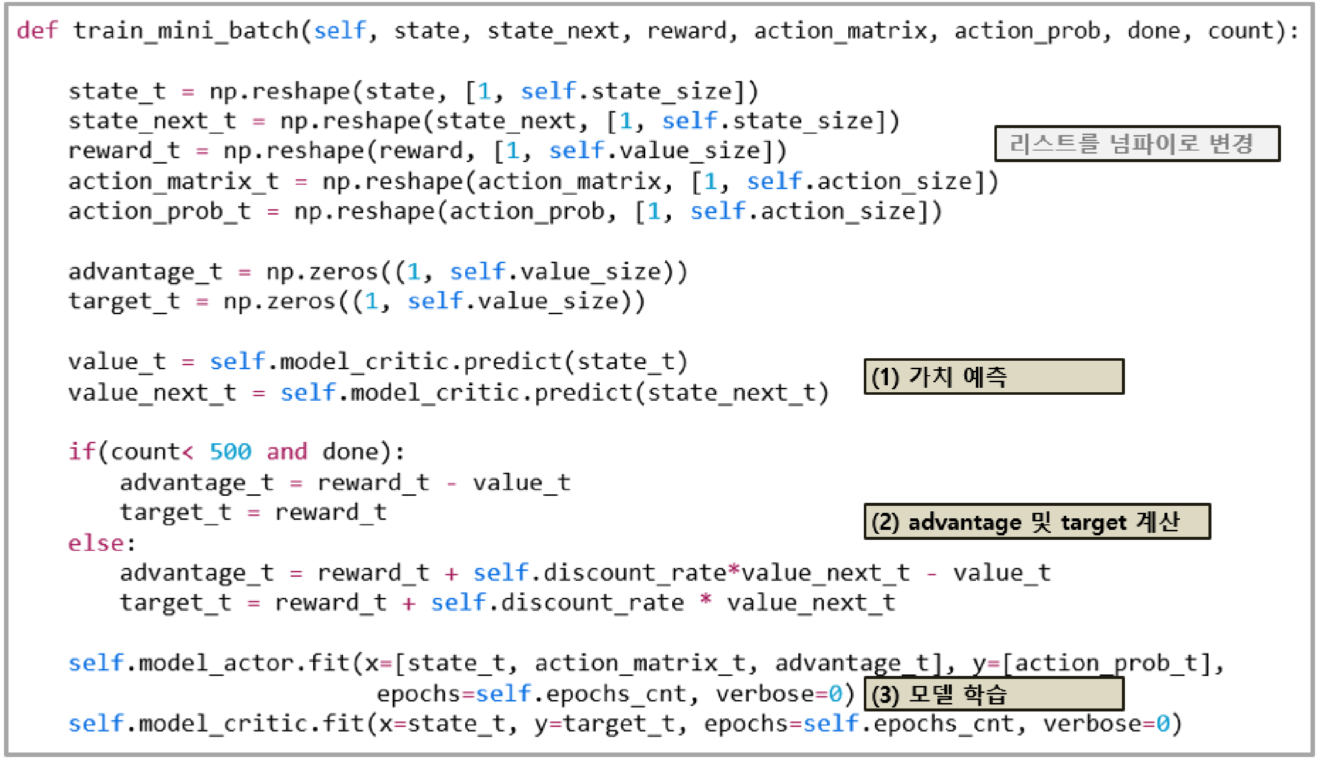
매트릭 생성

(2) 모델 학습

```python
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,self.state_size])
        action_matrix_t = np.reshape(action_matrix, [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]

        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.train_mini_batch(state, state_next, reward, action_matrix, action_prob,
                              done, count)

        state = state_next
        reward_tot += reward

    return count, reward_tot
```

A2C 프로그래밍 코드분석

train_mini_batch 함수



```
def train_mini_batch(self, state, state_next, reward, action_matrix, action_prob, done, count):

    state_t = np.reshape(state, [1, self.state_size])
    state_next_t = np.reshape(state_next, [1, self.state_size])
    reward_t = np.reshape(reward, [1, self.value_size])
    action_matrix_t = np.reshape(action_matrix, [1, self.action_size])
    action_prob_t = np.reshape(action_prob, [1, self.action_size])

    advantage_t = np.zeros((1, self.value_size))
    target_t = np.zeros((1, self.value_size))

    value_t = self.model_critic.predict(state_t)
    value_next_t = self.model_critic.predict(state_next_t)

    if(count< 500 and done):
        advantage_t = reward_t - value_t
        target_t = reward_t
    else:
        advantage_t = reward_t + self.discount_rate*value_next_t - value_t
        target_t = reward_t + self.discount_rate * value_next_t

    self.model_actor.fit(x=[state_t, action_matrix_t, advantage_t], y=[action_prob_t],
                         epochs=self.epochs_cnt, verbose=0)
    self.model_critic.fit(x=state_t, y=target_t, epochs=self.epochs_cnt, verbose=0)
```

리스트를 넘파이로 변경

(1) 가치 예측

(2) advantage 및 target 계산

(3) 모델 학습