# 강화학습

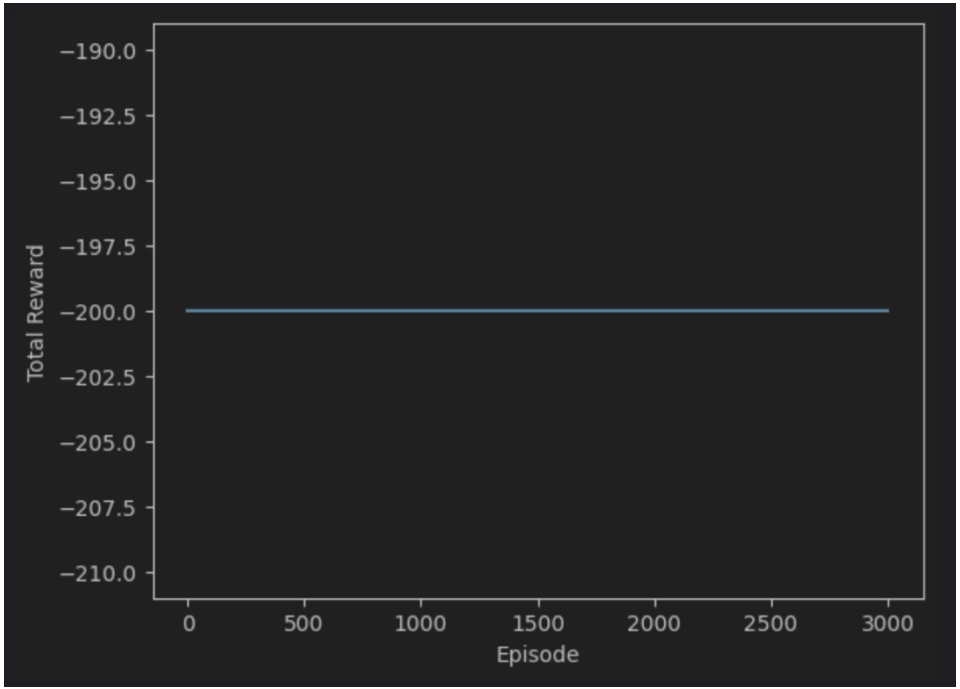
## Quiz-Policy Gradient Method

학과 : 산업인공지능학과
학번 : 2024254022
이름 : 정현일

2025.05.20.

충북대학교 산업인공지능학과

Actor-Critic 을 Mountain Car 문제에 적용하되, Hyper-parameter 를 변경하여 최대의 total reward 를 갖는 policy 를 결정하라.




| Hyper-parameter | Value |
|---|---|
| gamma | 0.98 |
| lr_pi | 0.0002 |
| lr_v | 0.0005 |
| episodes | 3000 |

# 문제점 및 해결책

1. 기존 보상의 문제점

•대부분의 에피소드에서 200 스텝 동안 목표에 도달하지 못함
•평균 보상이 거의 -200임
•기본 보상(-1)은 학습에 충분한 정보를 제공하지 못함

❖ 해결책

•위치와 속도를 활용한 보상 함수 재설계

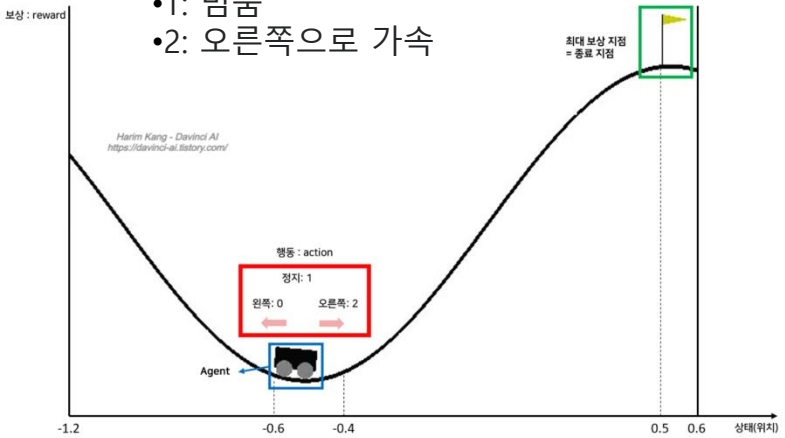
```
# 보상 설계
position = next_state[0]
velocity = next_state[1]

# 보상 설계 (위치와 속도 고려)
designed_reward = reward
if velocity > 0:    # 속도가 있고 오른쪽을 가면 보상 추가
    designed_reward = ((position + 1.2) / 1.8) ** 2

# 목표 도달 시 추가 보상
if position >= 0.5:
    designed_reward = 10

episode_reward += reward
```

➢ 2차원 연속 공간 [위치, 속도]

•위치(Position): -1.2 ~ 0.6 범위
•속도(Velocity): -0.07 ~ 0.07 범위

➢ 행동 공간(Action Space)

•0: 왼쪽으로 가속
•1: 멈춤
•2: 오른쪽으로 가속



# dezero 프레임워크이용 Actor-Critic 모델 구현.

```python
import numpy as np
import gym
from dezero import Model
from dezero import optimizers
import dezero.functions as F
import dezero.layers as L
import imageio
from matplotlib import pyplot as plt


class PolicyNet(Model):  # 정책 신경망
    def __init__(self, action_size=2):
        super().__init__()
        self.l1 = L.Linear(128)
        self.l2 = L.Linear(64)
        self.l3 = L.Linear(action_size)

    def forward(self, x):
        x = F.relu(self.l1(x))
        x = F.relu(self.l2(x))
        x = self.l3(x)
        x = F.softmax(x)  # 확률 출력
        return x


class ValueNet(Model):  # 가치 함수 신경망
    def __init__(self):
        super().__init__()
        self.l1 = L.Linear(128)
        self.l2 = L.Linear(1)
        self.l3 = L.Linear(64)
        self.l4 = L.Linear(1)

    def forward(self, x):
        x = F.relu(self.l1(x))
        x = self.l2(x)
        x = F.relu(self.l3(x))
        x = self.l4(x)
        return x
```

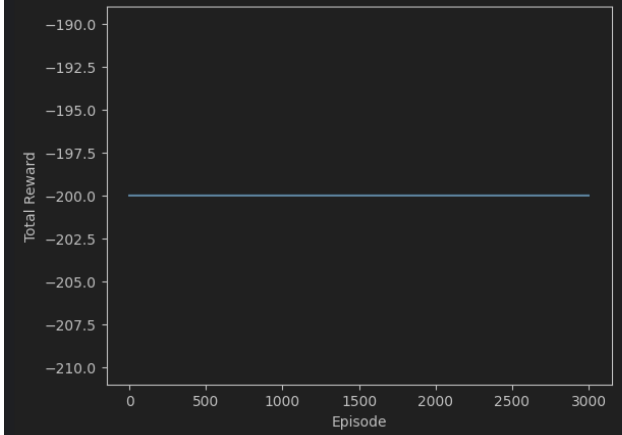
```python
class Agent:
    def __init__(self):
        self.gamma = 0.98
        self.lr_pi = 0.0001
        self.lr_v = 0.0001
        self.action_size = 2

        self.pi = PolicyNet()
        self.v = ValueNet()
        self.optimizer_pi = optimizers.Adam(self.lr_pi).setup(self.pi)
        self.optimizer_v = optimizers.Adam(self.lr_v).setup(self.v)

    def get_action(self, state):
        state = state[np.newaxis, :]  # 배치 처리용 축 추가
        probs = self.pi(state)
        probs = probs[0]
        action = np.random.choice(len(probs), p=probs.data)
        return action, probs[action]  # 선택된 행동과 해당 행동의 확률 반환

    def update(self, state, action_prob, reward, next_state, done):
        # 배치 처리용 축 추가
        state = state[np.newaxis, :]
        next_state = next_state[np.newaxis, :]


        # 가치 함수(self.v)의 손실 계산
        target = reward + self.gamma * 0.98 * self.v(next_state) * (1 - done)  # TD 목표
        target.unchain()
        v = self.v(state)  # 현재 상태의 가치 함수
        loss_v = F.mean_squared_error(v, target)  # 두 값의 평균 제곱 오차

        # 정책(self.pi)의 손실 계산
        delta = target - v
        delta.unchain()
        loss_pi = -F.log(action_prob) * delta

        # 신경망 학습
        self.v.cleargrads()
        self.pi.cleargrads()
        loss_v.backward()
        loss_pi.backward()
        self.optimizer_v.update()
        self.optimizer_pi.update()
```

```python
episodes = 3000
env = gym.make("MountainCar-v0", render_mode="rgb_array")
agent = Agent()
reward_history = []
best_reward = -float('inf')
best_frames = []

for episode in range(episodes):
    state = env.reset()[0]
    done = False
    total_reward = 0
    frames = []

    while not done:
        frame = env.render()
        frames.append(frame)
        action, prob = agent.get_action(state)
        next_state, reward, terminated, truncated, info = env.step(action)
        done = terminated | truncated

        # 보상 설계
        position = next_state[0]  #위치
        velocity = next_state[1]  #속도

        # 보상 설계 (위치와 속도 고려)
        designed_reward = reward
        if velocity > 0:  # 속도가 있고 오른쪽을 가면 보상 추가
            designed_reward = ((position + 1.2) / 1.8) **2

        # 목표 도달 시 추가 보상
        if position >= 0.5:
            designed_reward = 10

        total_reward += reward
        agent.update(state, prob, designed_reward, next_state, done)
        state = next_state
# 보상 기록
    reward_history.append(total_reward)
    if total_reward > best_reward:
        best_reward = total_reward
        best_frames = frames

    if episode % 100 == 0:
        print("episode :{}, total reward : {:.1f}".format(episode, total_reward))

from common.utils import plot_total_reward
plot_total_reward(reward_history)
```

# dezero 프레임워크이용 Actor-Critic 모델 구현.

•Dezero로 구현된 모델은 평균보상은 -200으로 학습불가

```
episode :0, total reward : -200.0
episode :100, total reward : -200.0
episode :200, total reward : -200.0
episode :300, total reward : -200.0
episode :400, total reward : -200.0
episode :500, total reward : -200.0
episode :600, total reward : -200.0
episode :700, total reward : -200.0
episode :800, total reward : -200.0
episode :900, total reward : -200.0
episode :1000, total reward : -200.0
episode :1100, total reward : -200.0
episode :1200, total reward : -200.0
```




```
Best episode reward: -200.0Best episode reward: -200.0
Best episode saved as best_mountaincar.gif
```

# Keras 활용 Actor-Critic 모델 구현 - 소스코드

## 1. 필요 라이브러리

```
import gym
import numpy as np
import tensorflow as tf
from tensorflow import keras
from sklearn.preprocessing import StandardScaler
import imageio
from matplotlib import pyplot as plt
import warnings

warnings.filterwarnings("ignore")


# 환경 생성
env = gym.make("MountainCar-v0", render_mode="rgb_array")
state_size = env.observation_space.shape[0]
action_size = env.action_space.n

# 상태 정규화를 위한 스케일러 준비
state_samples = np.array([env.observation_space.sample() for _ in range(10000)])
scaler = StandardScaler()
scaler.fit(state_samples)
```

## 2. Actor-Critic 모델 생성

```
# Actor-Critic 모델 생성
def build_actor_critic_model(state_size, action_size):
    # 공유 레이어
    input_layer = keras.layers.Input(shape=(state_size,))
    dense1 = keras.layers.Dense(64, activation='tanh')(input_layer)
    dense2 = keras.layers.Dense(32, activation='tanh')(dense1)

    # Actor 출력 (정책)
    policy = keras.layers.Dense(action_size, activation='softmax')(dense2)

    # Critic 출력 (가치)
    value = keras.layers.Dense(1)(dense2)

    # 모델 생성
    model = keras.models.Model(inputs=input_layer, outputs=[policy, value])

    return model
```

## 3. 모델 및 옵티마이저 설정

```
# 모델 및 옵티마이저 설정
model = build_actor_critic_model(state_size, action_size)
optimizer = keras.optimizers.Adam(learning_rate=0.001, clipnorm=1.0)

# 하이퍼파라미터
discount_factor = 0.99
episodes = 1000
```

## 4. 학습 함수 생성

```
def train_step(state, action, reward, next_state, done):
    with tf.GradientTape() as tape:
        # 현재 상태와 다음 상태의 정책과 가치 구하기
        policy, value = model(state)
        _, next_value = model(next_state)

        reward = tf.cast(reward, tf.float32)
        done = tf.cast(done, tf.float32)

        # 타겟 계산
        target = reward + (1 - done) * discount_factor * next_value[0]

        # Actor 손실 계산
        action_one_hot = tf.one_hot(action, action_size)
        log_prob = tf.math.log(tf.reduce_sum(action_one_hot * policy, axis=1) + 1e-10)
        advantage = target - value[0]
        actor_loss = -log_prob * tf.stop_gradient(advantage)

        # Critic 손실 계산
        critic_loss = tf.square(advantage)

        # 전체 손실
        loss = actor_loss + 0.5 * critic_loss

    # 그래디언트 계산 및 적용
    grads = tape.gradient(loss, model.trainable_variables)
    optimizer.apply_gradients(zip(grads, model.trainable_variables))

    return loss
```

가치 함수 손실 계산

정책 함수 손실 계산

그래디언트 계산 및 적용

# 소스코드

## 5. 학습 루틴

```python
for episode in range(episodes):
    state = env.reset()[0]
    episode_reward = 0
    done = False
    frames = []

    while not done:
        frame = env.render()
        frames.append(frame)
        state_tensor = tf.convert_to_tensor(state.reshape(1, -1), dtype=tf.float32)
        policy, _ = model(state_tensor)
        action = np.random.choice(action_size, p=policy[0].numpy())

        # 환경에 행동 적용
        next_state, reward, terminated, truncated, info = env.step(action)
        done = terminated | truncated

        # 보상 설계
        position = next_state[0]
        velocity = next_state[1]

        # 보상 설계 (위치와 속도 고려)
        designed_reward = reward
        if velocity > 0:   # 속도가 있고 오른쪽을 가면 보상 추가
            designed_reward = ((position + 1.2) / 1.8) ** 2

        # 목표 도달 시 추가 보상
        if position >= 0.5:
            designed_reward = 10

        episode_reward += reward
        next_state = scaler.transform([next_state])[0]

        # 학습 단계
        next_state_tensor = tf.convert_to_tensor(next_state.reshape(1, -1), dtype=tf.float32)
        train_step(state_tensor, action, designed_reward, next_state_tensor, done)

        state = next_state
```

산악차 프레임 저장

Agent 행동 선택

보상 설계 수정사항

학습에 반영

## 6. 상태 저장 및 보상값 plot출력

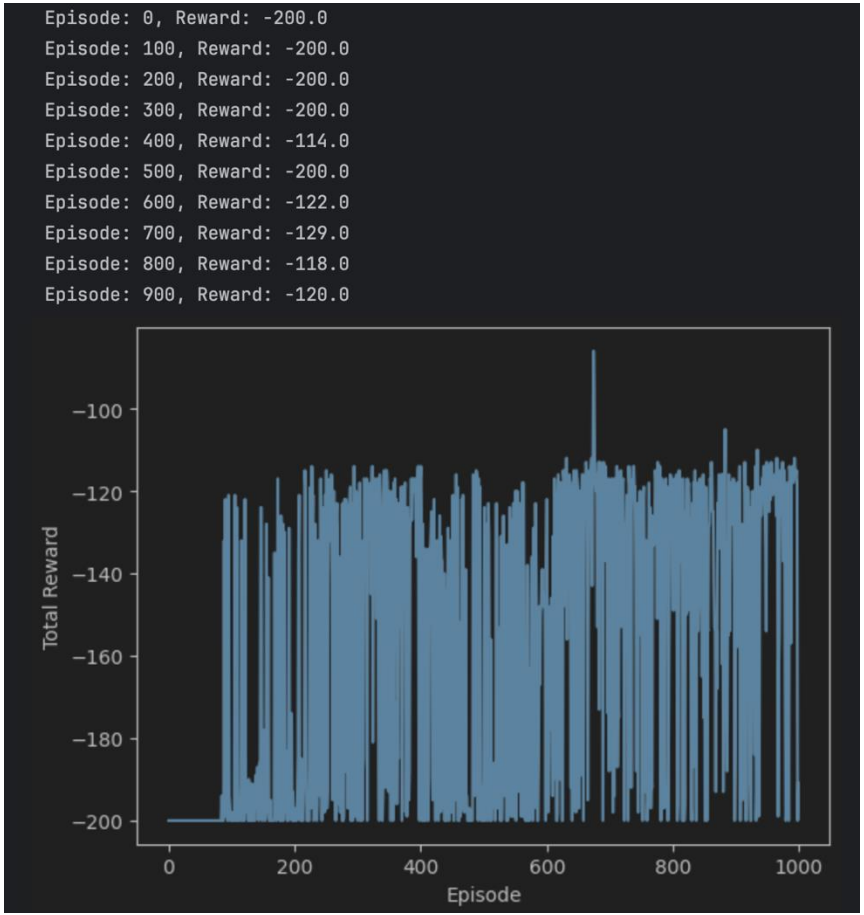
```python
    reward_history.append(episode_reward)
    if episode_reward > best_reward:
        best_reward = episode_reward
        best_frames = frames

    # 에피소드 결과 출력
    if episode % 100 == 0:
        print(f"Episode: {episode}, Reward: {episode_reward}")


from common.utils import plot_total_reward
plot_total_reward(reward_history)

env.close()
print(f"Best episode reward: {best_reward}")
```

## 7. 이미지 저장및 산악차 첫 장면 시각화

```python
# 이미지로 저장
output_filename = "best_mountaincar2.gif"
imageio.mimsave(output_filename, best_frames, fps=30)
print(f"Best episode saved as {output_filename}")

# 첫 장면만 시각화
plt.imshow(best_frames[0])
plt.title("First Frame of Best Episode")
plt.axis('off')
plt.show()
```

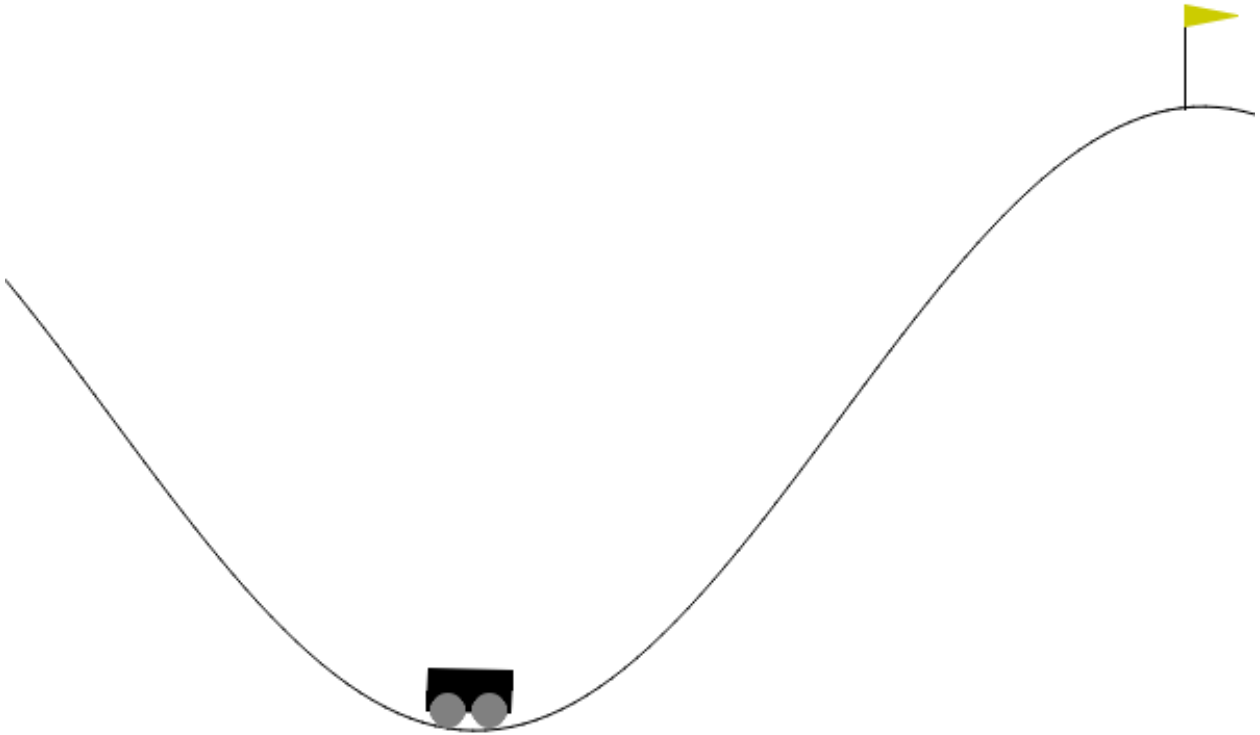
# Keras 활용 Actor-Critic 모델 구현 - 결과.

```
Episode: 0, Reward: -200.0
Episode: 100, Reward: -200.0
Episode: 200, Reward: -200.0
Episode: 300, Reward: -200.0
Episode: 400, Reward: -114.0
Episode: 500, Reward: -200.0
Episode: 600, Reward: -122.0
Episode: 700, Reward: -129.0
Episode: 800, Reward: -118.0
Episode: 900, Reward: -120.0
```




| Hyper-parameter | Value |
|---|---|
| gamma | 0.99 |
| lr | 0.001 |
| episodes | 1000 |

# Mountain car 최적의 보상 동영상



※ F5 발표자 모드에서 PLAY이 됩니다.