Term Project

2018324133 김태우

1. 목적

이 프로젝트이 목적은 Cart-Pole Balancing 문제를 Actor-Critic Method 방식으로 해결하는 것이다. Cart-Pole Balancing이란 카트를 움직여 카트 위에 세워져있는 막대가 쓰러지지 않도록 유지하는 것이다. Actor-Critic Learning이란 Actor와 Critic이라고 명명된 두 개의 네트워크를 Actor는 Policy를, Critic은 Value를 Estimation하도록 트레이닝하여, Action Policy를 직접 도출하면서도 평가는 Value-based로 할 수 있는 러닝 기법이다.

2. 실험 환경

우선 프로젝트를 진행하기에 앞서, Agent를 트레이닝하고 평가하기 위한 게임 환경을 설계하였다. 이 환경은 Cart-Pole 문제의 물리 모델을 수학적으로 구현하여, Cart에 줄 힘의 크기를 input으로 받고 Cart와 Pole의 물리적 상태를 결과로 제공해 줄 것이다. 또한 우리가 Agent의 성능을 쉽게 평가할 수 있도록 현재 상태에 대한 Visualize도 제공한다.

사용한 물리 모델은 다음과 같다.

$$
\begin{aligned}
\dot{\omega} &= \frac{g\sin\theta + \frac{(\mu_c \mathrm{sgn}(v) - F - ml\omega^2\sin\theta)\cos\theta}{m+m_c} - \frac{\mu_p \omega}{ml}}{l\left(\frac{4}{3} - \frac{m}{m+m_c}\cos^2\theta\right)} \\
\dot{v} &= \frac{F + ml(\omega^2\sin\theta - \dot{\omega}\cos\theta) - \mu_c \mathrm{sgn}(v)}{m+m_c} \\
\omega &= \dot{\theta} \\
v &= \dot{x}
\end{aligned}
$$

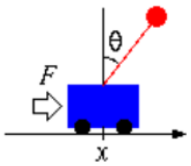


그림 1 Cart-Pole 물리 모델

구현을 위해 Python과 numpy, opencv 라이브러리를 사용하였다.

```
import numpy as np
import cv2

# Constants
timeconst = 0.025 # time difference of each frame
FMax = 20. # Maximum force Agent can apply
sigma_noise = 1. # Stddev of Random noise
mu_c = 0.01 # Friction Factor
mu_p = 0.01 # Friction Factor
```

```
G = 9.8 # Gravity
M = 1 # Mass of Cart
m = 0.1 # Mass of Pole
L = 1 # Length of Pole

# Initial Values
x = 0.
v = 0.
theta = 0.
w = 0.
F = 0.

image = np.zeros((240, 320, 3), dtype = np.float)

while True:
        # Apply Phisical Model
        sintheta = np.sin(theta)
        costheta = np.cos(theta)
        sgn_mu_c = mu_c if v > 0. else -mu_c if v < 0. else 0.
        F += np.random.normal(0., 0.02 * timeconst)

        dw = G * sintheta + ₩
                ((sgn_mu_c - F - m * L * w * w * sintheta) * costheta) / (M + m) - ₩
                (mu_p * w) / (m * L)
        dw /= L * (1.333333 - m / (m + M) * costheta * costheta)
        dv = (F + m * L * (w * w * sintheta - dw * costheta) - sgn_mu_c) / (m + M)

        # Update Values
        w += dw * timeconst
        v += dv * timeconst

        theta += w * timeconst
        x += v * timeconst

        # Drawing
        cv2.rectangle(image, (0, 0), (320, 240), (1., 1., 1.), -1)

        cv2.line(image, (0, 120), (320, 120), (0., 1., 0.), 2)
        t = int(((x / 1.) - np.floor(x / 1.)) * 40.)
        while t < 320 :
                cv2.line(image, (t, 115), (t, 125), (0., 1., 0.), 2)
                t += 40

        cv2.rectangle(image, (140, 110), (180, 130), (1., 0., 0.), -1)
        cx = int(160 + sintheta * 60);
        cy = int(120 + costheta * -60);
        cv2.line(image, (160, 120), (cx, cy), (0., 0., 0.), 5)
        cv2.circle(image, (cx, cy), 10, (0., 0., 1.), -1)
```

```
        cv2.imshow("game", image);
        key = cv2.waitKey(30);

        # Human Input
        if key == ord('z') :
                F = -FMax
        elif key == ord('x') :
                F = FMax
        elif key == ord('q') :
                break
        else:
                F = 0.
```

그리고 사람이 이 게임을 플레이하여 구현의 정확도와 이 문제의 난이도를 검증하였다.

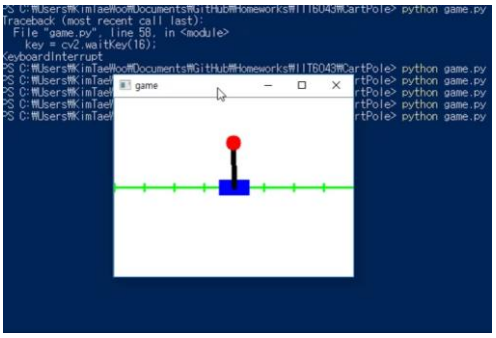


그림 2 CartPole human playing

https://boratw.github.io/Homeworks/IIT6043/CartPole/humanplaying.mp4

플레이해본 결과 일반적인 프레임 속도로 사람이 하기엔 굉장히 난이도가 있는 문제임을 알 수 있었다. 사람은 프레임 단위로 반응할 수 없기 때문이다. AI Agent는 프레임 단위로 결과를 얻을 수 있으므로 더 훌륭한 플레이를 할 것을 기대하며 이후 단계를 진행하였다.

3. 네트워크 설계

Actor-Critic Method로 문제 해결을 위해 Actor와 Critic 두 개의 뉴럴 네트워크를 설계해야 한다.

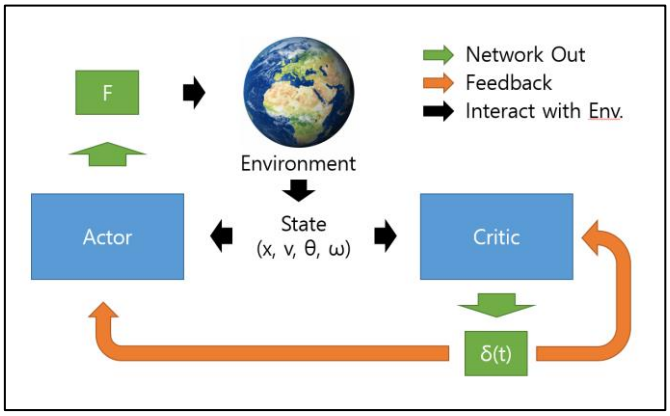


그림 3 Actor-Critic 전체 구조

Actor는 Environment로부터 State를 받아 카트에 가할 힘 (Policy)를 도출한다. Critic은 State로부터 Total Reward (Value)를 도출하여 Actor의 action이 얼마나 좋았는지에 대한 가이드를 제공하고, 자신 역시 트레이닝한다.

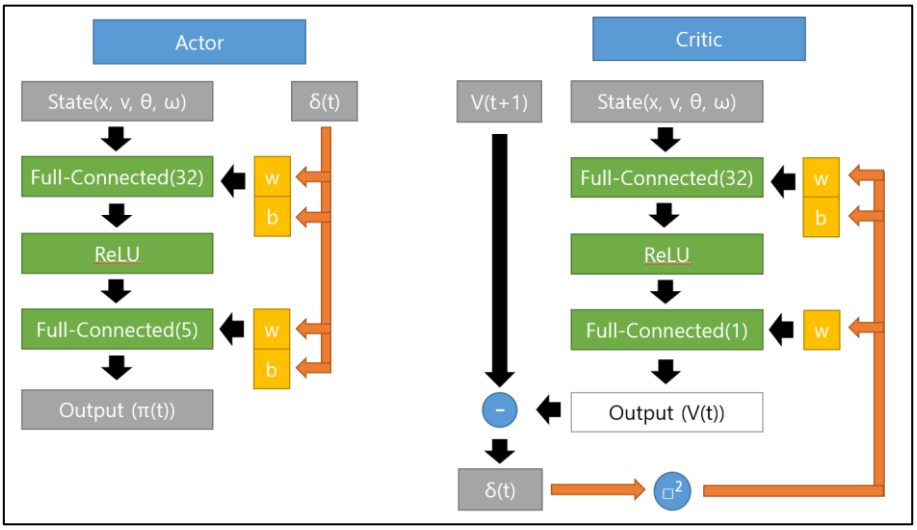


그림 4 각 네트워크의 구성도

Actor 네트워크는 내부에 32 unit의 hidden layer를 1개 가지고 있다. input으로는 현재 state를 받고, Output으로 현재 state에서의 policy를 출력한다. action space는 [-1, -0.5, 0, 0.5, 1] * maxF의 다섯 개를 사용하였다. 그리고 Critic Network로부터 TD of Value δ(t)를 받아 parameter들의 gradiant로 사용한다.

Critic 네트워크도 내부에 32 unit의 hidden layer를 1개 가지고 있다. input으로 현재 state를 받고 현재 state에서의 Value (얻을 수 있는 Reward의 총합)을 출력한다. 또한 같은 네트워크에로 Next State의 Value 역시 얻어서, TD of Value δ(t)를 구하고 그 제곱을 parameter들의 gradient로 사용한다.

Tensorflow 라이브러리를 이용한 구현은 다음과 같다.

```
# Constants
TD_GAMMA = 0.9
# Actor Network ( One Hidden layer with 32 elements )
global_step = tf.Variable(0, trainable=False)
with tf.variable_scope('Actor'):
        actor_x = tf.placeholder(tf.float32, [None, 4])
        actor_action = tf.placeholder(tf.int32, [None])
        actor_td_error = tf.placeholder(tf.float32, [None])

        actor_w1 = tf.Variable(tf.random_normal([4, 32], stddev=0.1))
        actor_b1 = tf.Variable(tf.random_normal([32], stddev=0.1))
        actor_f1 = tf.nn.relu(tf.matmul(actor_x, actor_w1) + actor_b1)
```

```
        actor_w2 = tf.Variable(tf.random_normal([32, 5], stddev=0.1))
        actor_b2 = tf.Variable(tf.random_normal([5], stddev=0.1))
        actor_y = tf.matmul(actor_f1, actor_w2) + actor_b2
        actor_act = tf.nn.softmax(actor_y)

        actor_action_hot = tf.one_hot(actor_action, 5)
        actor_log_prob = tf.log(tf.reduce_sum(actor_act * actor_action_hot) + 1e-15)
        actor_loss = -actor_log_prob * actor_td_error

        actor_train = tf.train.AdamOptimizer(tf.train.exponential_decay(0.001, global_step, 1000, 0.5))₩
                .minimize(actor_loss)
# Critic Network ( One Hidden layer with 32 elements )
with tf.variable_scope('Critic'):
        critic_x = tf.placeholder(tf.float32, [None, 4])
        critic_v_next = tf.placeholder(tf.float32, [None, 1])
        critic_r = tf.placeholder(tf.float32, [None, 1])

        critic_w1 = tf.Variable(tf.random_normal([4, 32], stddev=0.1))
        critic_b1 = tf.Variable(tf.random_normal([32], stddev=0.1))
        critic_f1 = tf.nn.relu(tf.matmul(critic_x, critic_w1) + critic_b1)

        critic_w2 = tf.Variable(tf.random_normal([32, 1], stddev=0.1))
        critic_v = tf.matmul(critic_f1, critic_w2)

        critic_error = tf.reduce_mean(   critic_r + critic_v_next * TD_GAMMA - critic_v )
        critic_train = tf.train.AdamOptimizer(tf.train.exponential_decay(0.01, global_step, 1000, 0.5))₩
                .minimize(tf.square(critic_error) )
```

Optimizer는 AdamOptimizer를 사용하였고 Learning Rate는 Actor는 0.001, Critic은 0.01부터 시작하여 1000 step마다 0.1배로 감소하도록 하였다.

4. 트레이닝 과정

Agent를 실제로 앞서 만든 Game Environment를 플레이하도록 하여 트레이닝을 진행하였다.

매 Game은 θ의 절대값이 90º 이상이 되거나 (Pole이 쓰러짐) x의 절대값이 10 이상이 되거나 (Cart가 지나치게 이동함) 1000 frame을 버티면 중단하도록 하였으며, reward는 처음엔 cos(θ)를 사용하였으나 트레이닝이 잘 되지 않아 좀 더 수직한 상태가 높은 reward를 받도록 $\cos^3(\theta)$을 사용하였다.

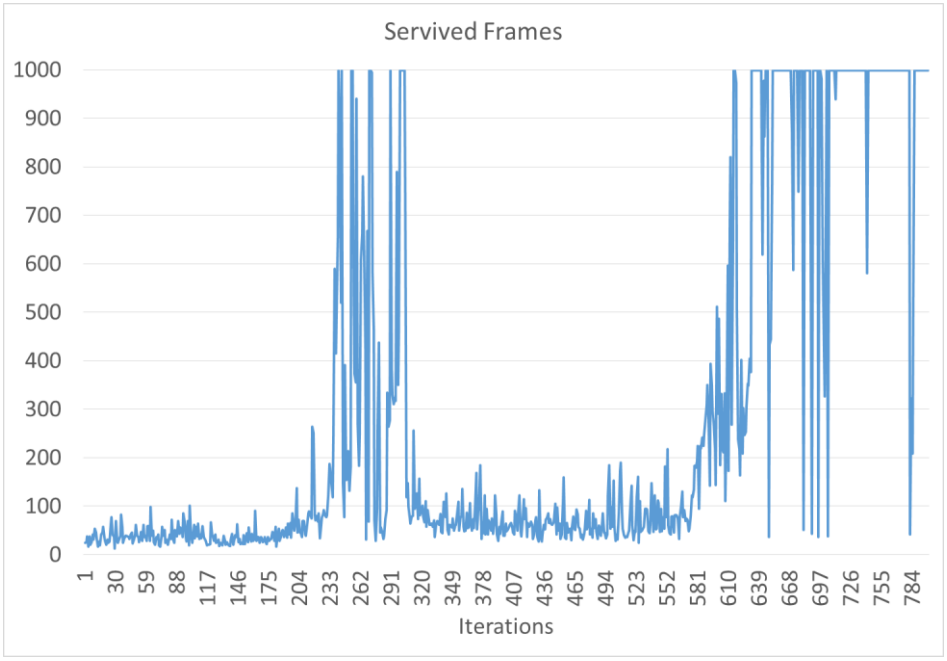
트레이닝 코드는 다음과 같다

```
for iteration in range(1, 100001) :
        # Game Start : Env Reset
        x = 0.
        v = 0.
        theta = np.random.normal(0., 0.2) # Randomize Initial theta
        w = 0.
```

```
td_error_sum = 0.
actor_loss_sum = 0.
reward_sum = 0.
for step in range(1000) :
        # Get Action
        act_prob = sess.run(actor_act, {actor_x : np.array([[x, v, theta, w]])})
        #act_prob_added = [t + 0.05 for t in act_prob[0]]
        #act_prob_added /= np.sum(act_prob_added)
        action = np.random.choice(5, p=act_prob[0])
        F = (action - 2) * 0.5 * FMax + np.random.normal(0., sigma_noise)

        ## Physics part of Environment
        ## Get next_x, next_v, next_theta, next_w

        # Learn Critic
        reward = np.cos(next_theta) ** 3
        if reward > 0. or next_x < -10. or next_x > 10.:
                next_value = sess.run(critic_v, {critic_x : np.array([[next_x, next_v, next_theta, next_w]])})
                [_, td_error] = sess.run([critic_train, critic_error], ₩
                        {global_step : iteration, critic_x : np.array([[x, v, theta, w]]), ₩
                        critic_v_next : next_value, critic_r : np.array([[reward]])})
        else :
                [_, td_error] = sess.run([critic_train, critic_error], ₩
                        {global_step : iteration, critic_x : np.array([[x, v, theta, w]]), ₩
                        critic_v_next : np.array([[0.]]), critic_r : np.array([[reward]])})

        # Learn Actor
        [_, loss] = sess.run([actor_train, actor_loss], ₩
                {global_step : iteration, actor_x : np.array([[x, v, theta, w]]), ₩
                actor_action : np.array([action]), actor_td_error : np.array([td_error])})

        # Record current Output
        td_error_sum += td_error ** 2
        actor_loss_sum += np.abs(loss[0])
        reward_sum += reward

        ## Remaining part of Environment
        ## Update State

        if reward <= 0. :
                break
# Log average outputs of current game
print(step, reward_sum / step, actor_loss_sum / step, td_error_sum / step)
f_w.write(str(step) + "₩t" + str(reward_sum / step) + "₩t" + ₩
        str(actor_loss_sum / step) + "₩t" + str(td_error_sum / step) + "₩n")
# Save network at every 100 games
if iteration % 100 == 0:
        saver.save(sess, ".₩₩model1", global_step=iteration)
```

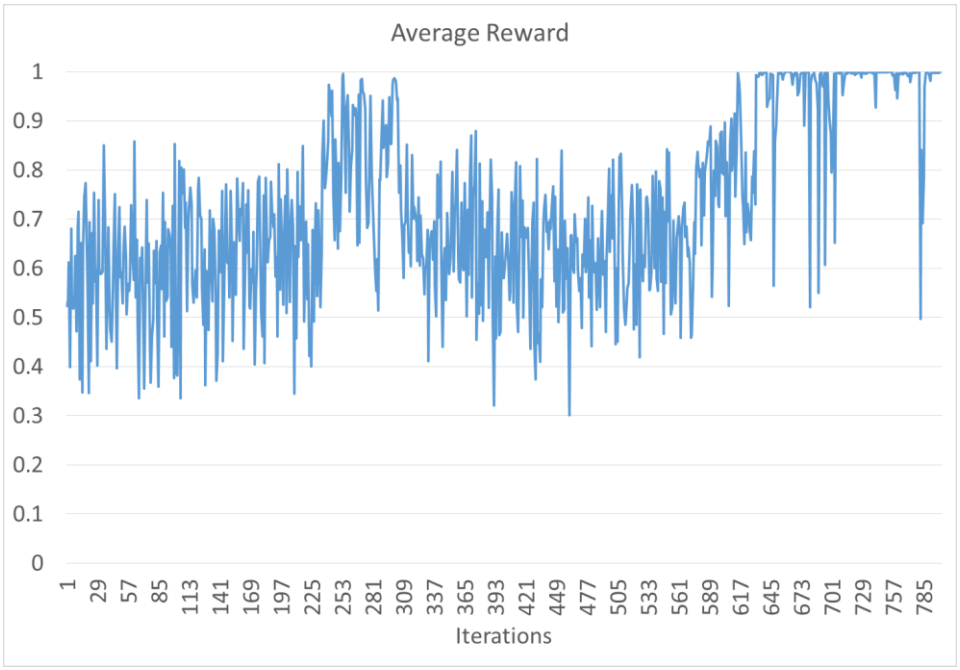
5. 결과

총 800회의 Game후 충분히 수렴했다고 판단하여 트레이닝을 종료하였다.



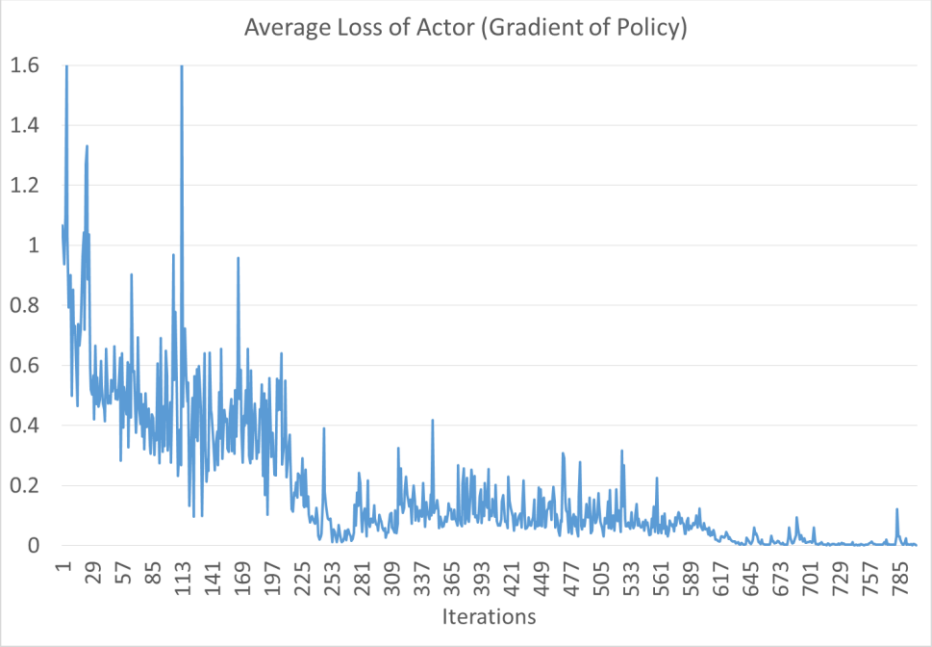
그래프 1 Pole이 쓰러지지 않고 버틴 Frame 수

가장 성능을 잘 알 수 있는 각 게임에서 버텨낸 Frame 수를 나타낸 그래프이다. 600회를 넘으면서 거의 항상 끝까지 버텼음을 확인할 수 있다.



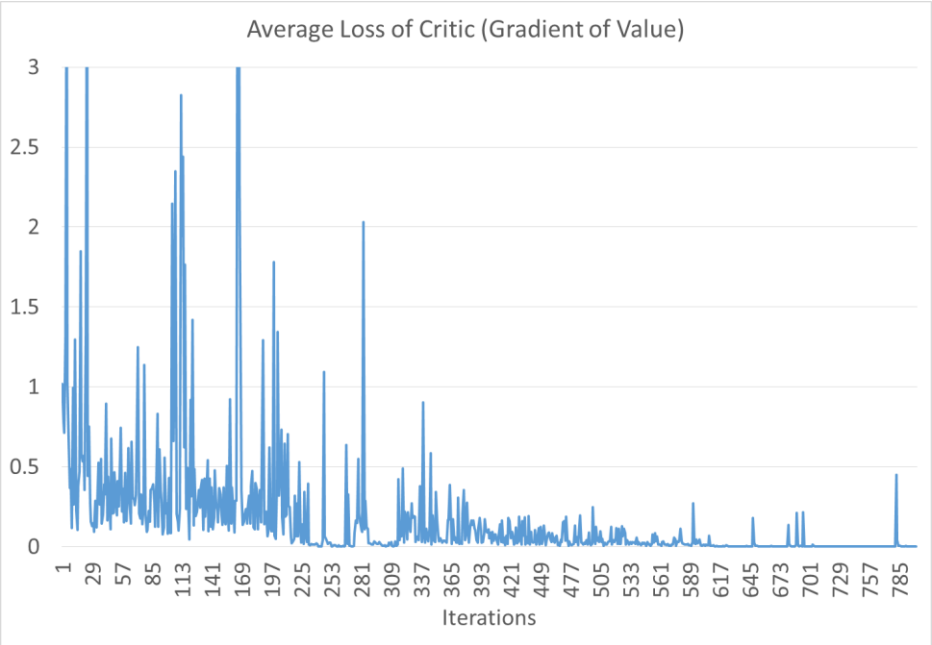
그래프 2 각 Game 동안의 평균 reward

게임을 진행하는 동한 agent가 얻은 reward의 평균 그래프이다. 1에 가깝다는 건 게임 동안 거의 항상 Pole이 수직으로 서 있었다는 것을 의미한다.



그래프 3 각 Game의 Actor Loss의 절대값의 평균

Actor 네트워크의 Loss인 -log(π(a|s)) * δ(t)의 절대값의 평균이다. 700회를 넘으며 Actor가 잘 수렴하였음을 알 수 있다.



그래프 4 각 Game의 Critic Loss의 절대값의 평균

Critic 네트워크의 Loss인 $\delta(t)^2$의 평균이다. Critic 역시 잘 수렴하였음을 알 수 있다.

Training 과정에서의 테스트 영상 및 최종 결과 영상은 다음 링크에서 확인할 수 있다.

100회 : https://boratw.github.io/Homeworks/IIT6043/CartPole/train100.mp4

300회 : https://boratw.github.io/Homeworks/IIT6043/CartPole/train300.mp4

600회 : https://boratw.github.io/Homeworks/IIT6043/CartPole/train600.mp4

800회 (최종) : https://boratw.github.io/Homeworks/IIT6043/CartPole/train800.mp4

Lots of Noise : https://boratw.github.io/Homeworks/IIT6043/CartPole/train800-noise.mp4

마지막은 카트에 부여되는 Random Noise의 Stddev를 Fmax와 같게 만들고 찍은 영상이다. 매우 악조건에서도 잘 버티는 것을 확인할 수 있다.

6. 결론

Actor-Critic Method 방식으로 Cart-Pole 문제를 풀 수 있는 Agent를 만들고 실제로 트레이닝을 시켜 보았다. 이번 프로젝트를 통해 문제를 해결하기 위해 Envorionmet를 설계하고, 문제를 잘 해결할 수 있는 Network를 구성하여 실제로 Reinforced Learning을 이용해 네트워크를 트레이닝 하는 것 까지 익힐 수 있었다.