머신러닝 심화 - 기말과제

인공지능소프트웨어과
2301110336
유승호

목 차

1. 목적

Unity 와 Python 통신을 이용하여 강화학습 구현방법 조사

2. 강화학습의 개요

- 강화학습의 주요 개념

- Agent가 Environment과 상호작용을 통해 보상을 최대화하는 행동 방식을 학습하는 인공지능 방식
- Agent는 Action을 수행하는 주체, 학습을 통해 최적의 행동을 찾음
- Environment는 Agent가 상호작용하며 State와 Reward가 변화
- State는 Agent가 관찰할 수 있는 정보
- Action은 Agent가 현재 상태에서 취할 수 있는 행동
- Reward는 행동의 결과로 보상이 클수록 Agent가 해당 행동을 선호하도록 학습

- 강화학습의 학습 원리

- 탐험과 이용을 반복해 최적의 행동을 학습
- 탐험은 새로운 행동을 시도해 보상을 확인
- 이용은 현재까지 배운 지식을 바탕으로 최적의 행동을 선택

- DQN 알고리즘

- DQN 알고리즘은 논문 “Playing Atari with Deep Reinforcement Learning”에서 처음 소개됨
- 기존 Q-Learning의 State-Action 값을 신경망으로 근사화
- Replay Buffer를 통해 이전에 수행한 행동을 저장, 샘플링해 Agent의 편향된 학습 억제
- Target Network를 통해 일정한 주기마다 학습 중인 네트워크의 가중치를 복사해 Target Network 사용, Q값 업데이트의 안정성을 높임

3. Unity와 Python을 연동하여 강화학습을 구현하는 방법

1) 구현 방법

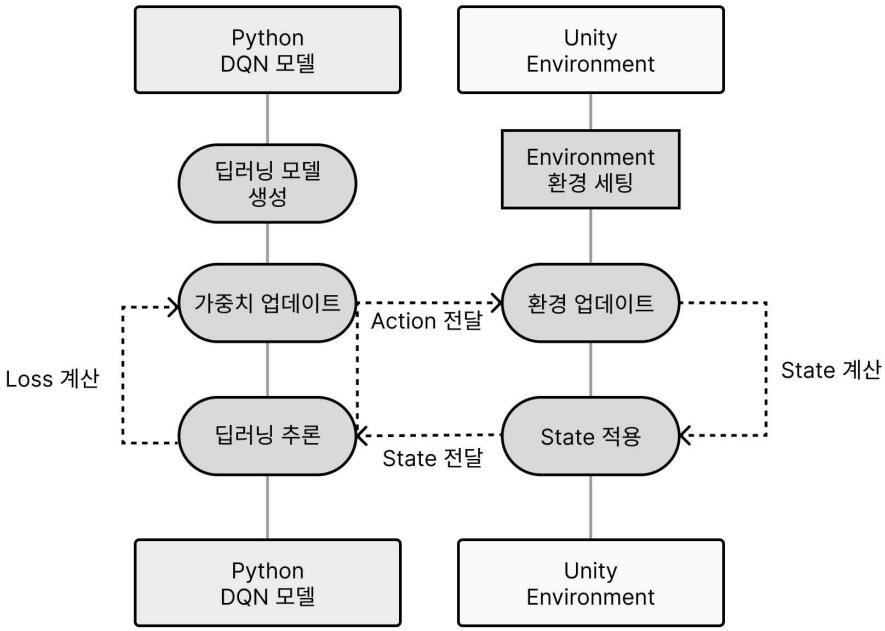
- Unity를 통한 환경, Action, Reword, Agent 설정

- Unity를 이용해 Environment를 제작
- 빨간색 구체를 생성해 Agent로 사용
- 물리 엔진을 이용해 Environment환경 설정
- 환경의 최대 보상은 빨간색 정육면체에 Agent가 충돌하는 것을 목표
- 빨간색 정육면체와 Agent의 거리가 가까워질수록 높은 Reword 설정
- 빨간색 정육면체와 Agent가 충돌하면 가장 높은 Reword 설정
- 환경에 Forward, Backward, Right, Left Action 추가
- TCP 통신으로 Agent의 다음 Action을 받아 State를 업데이트

- **Python을 이용한 딥러닝 강화학습 설정**

  - Tensorflow로 DQN 알고리즘 설계
  - 2개의 좌표(x,z)를 입력받아 다음 행동 추론
  - Action 수행을 위해 출력을 4개로 조정
  - Epsilon 값을 통해 탐험할 비율 조정
  - TCP 통신을 이용해 State, Reword를 전달받음
  - 학습이 진행되는 동안 Action을 서버로 전달해 변경된 State 변경

- **Flow 차트**



2) 장점

- State와 Action을 분리해 코드 가독성이 높음
- 실제 Agent가 이동함으로써 State의 변화를 실시간 확인 가능
- 컴퓨터 자원이 부족하다면 Environment과 DQN을 분리해 각기 다른 컴퓨터에서 구동 가능

3) 단점

- 통신을 활용하기 때문에, delay가 어느 정도 발생할 수 있음
- Reword 계산식이 게임 내 규칙에 존재하기 때문에 기능적 분리가 아쉬움

4. 학습된 모델을 실행하는 방법

1) 학습

- DQN.py에 다음 코드 추가

Episode, Loss, Epsilon, Total Reward 기록과 Best Loss를 보여주는 가중치를 저장하기 위해 다음 코드를 추가

```
def train():
    bestloss = 1.0

    agent = DQNAgent()
    episodes = 5000  # 학습할 에피소드 수  123,456
    for e in range(episodes):
        conn.sendall("NewStatus\n".encode())
        data = conn.recv(1024)  # 1024 바이트 크기
        print(data)
        state = data.decode().split(',')  # 통신으로 받아와야 함.
        state = [float(state[0]), float(state[1])]
        total_reward = 0
        done = False

        while not done:
            action = agent.act(state)  # 액션 선택
            #통신으로 액션 전달
            strAction = "{0}\n".format(action)
            conn.sendall(strAction.encode())
            #next_state, reward 받아야 함.
            data = conn.recv(1024)  # 1024 바이트 크기
            data = data.decode().split(',')  # 통신으로 받아와야 함.
            print(data)
            next_state = [float(data[0]), float(data[1])]
            reward = float(data[2])
            done = data[3].replace("\r\n", "")
            if done == "0":
                done = False
            else:
                done = True
            agent.memory.add((state, action, reward, next_state, done))  # 경험 리플레이에 저장
            state = next_state
            total_reward += float(reward)

            loss = agent.learn()  # 모델 학습
            print("loss =",loss)
            #conn.sendall("Status\n".encode())
            #data = conn.recv(1024)  # 1024 바이트 크기
            #state = data.decode().split(',')  # 통신으로 받아와야 함.

        if loss is not None and not tf.math.is_nan(loss) and loss < bestloss:
            bestloss = loss
            with open("training_best_log.csv", "a", newline="") as file:
                writer = csv.writer(file)
                writer.writerow([e + 1, loss, total_reward, agent.epsilon])
            agent.model.save('dqn_model_best', save_format='tf')

        with open("training_all_log.csv", "a", newline="") as file:
            writer = csv.writer(file)
            writer.writerow([e + 1, loss, total_reward, agent.epsilon])
        print(f"Episode: {e+1}/{episodes}, Total Reward: {total_reward}, Epsilon: {agent.epsilon}")

    agent.model.save('dqn_model', save_format='tf')
```

- 매 episodes의 loss가 낮아질 때마다 가중치가 dqn_model_best에 저장되도록 설정
- 훈련 로그와 저장될 때마다 로그가 각각 저장되도록 설정

2) 실행

- DQNTest.py 파일 생성

```
file_name = 'validation(lastmodel).csv'
with open(file_name, mode='w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow(['Prediction Count', 'Clear Status', 'total_reward'])
class DQNAgent:
    def __init__(self):
        self.model = load_model('dqn_model_best')
        self.epsilon = 0.0
    def act(self, state):
        if np.random.rand() <= self.epsilon:
            return random.randrange(action_size)  # 랜덤 액션 선택 (탐사)
        q_values = self.model(np.array([state]))
        return np.argmax(q_values)  # 최대 Q-value를 가진 액션 선택 (활용)


agent = DQNAgent()
episodes = 500  # 학습할 에피소드 수  123,456

for e in range(episodes):
        conn.sendall("NewStatus\n".encode())
        data = conn.recv(1024)  # 1024 바이트 크기
        state = data.decode().split(',')  # 통신으로 받아와야 함.
        state = [float(state[0]), float(state[1])]
        total_reward = 0
        done = False
        prediction_count = 0  # 예측 횟수 초기화
        clear = False


        while not done:
            action = agent.act(state)  # 액션 선택
            prediction_count += 1  # 예측 횟수 증가
            #통신으로 액션 전달
            strAction = "{0}\n".format(action)
            conn.sendall(strAction.encode())
            #next_state, reward 받아야 함.
            data = conn.recv(1024)  # 1024 바이트 크기
            data = data.decode().split(',')  # 통신으로 받아와야 함.
            next_state = [float(data[0]), float(data[1])]
            reward = float(data[2])
            done = data[3].replace("\r\n", "")
            if done == "0":
                done = False
            else:
                done = True

            state = next_state
            total_reward += float(reward)
            if reward >= 1:
                clear = True

        print(prediction_count)
        print("clear: ", clear)

        with open(file_name, mode='a', newline='') as file:
            writer = csv.writer(file)
            writer.writerow([prediction_count, clear, total_reward])
```

- 학습 결과 실행에는 이용만 해야 하므로 epsilon의 값을 0.0으로 조정
- 500번의 게임을 통해 강화학습을 평가
- 성공 시 몇 번의 움직임을 전달했는지 저장하는 변수를 추가
- validation.csv에 log 기록

- Random_Positions.csv 생성

- 모든 모델은 같은 Environment에서 평가해야 함
- Environment의 변인 중 하나인 빨간색 큐브의 위치 500개를 사전에 csv로 만들어 모두 같게 함
- GPT를 이용해 -35, 35 사이의 X, Y 좌표 500쌍 생성

Random Positions

	X	Z
1	-1.0	3.3
2	0.4	1.1
3	-1.7	0.6
4	-1.0	2.2
5	2.7	1.9
6	-2.1	2.2

500개의 랜덤 위치(X, Z)가 생성되었습니다. 필요에 따라 추가적인 조정이 필요하면 알려주세요.

- Unity의 Resources/Random_Positions.csv에 저장

- MyCheck.cs 수정

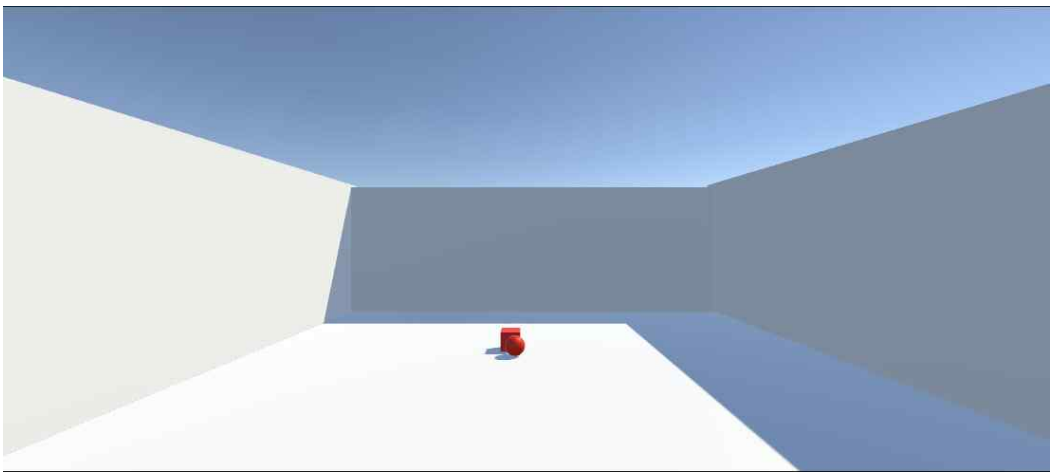
■ Environment를 적용하기 위해 코드 수정

```
private void LoadCSV()
{
    positions = new List<Vector3>();

    TextAsset csvData = Resources.Load<TextAsset>("Random_Positions");

    string[] rows = csvData.text.Split('\n');

    for (int i = 1; i < rows.Length; i++)
    {
        if (!string.IsNullOrEmpty(rows[i]))
        {
            string[] values = rows[i].Split(',');

            // X, Z 위치값을 float로 변환합니다.
            float x = float.Parse(values[0]);
            float z = float.Parse(values[1]);

            positions.Add(new Vector3(x, 0.5f, z));
        }
    }
}
void Start()
{
    LoadCSV(); // 🔥 CSV 파일 불러오기
    ConnectToServer();
    initPoint = transform.position - agent.transform.position;
}

private Vector3 GetNextPositionFromCSV()
{
    Vector3 nextPosition = positions[currentIndex];

    currentIndex++;
    if (currentIndex >= positions.Count)
    {
        currentIndex = 0; // 500개를 전부 돌았으면 다시 처음으로
    }

    UnityEngine.Debug.Log($"다음 위치로 이동: {nextPosition}");
    return nextPosition;
}
```

```
if (done == 1)
{
    done = 0;
    step = 0;

    Rigidbody target = agent.gameObject.GetComponent<Rigidbody>();
    Vector3 zero = Vector3.zero;
    target.velocity = zero;
    target.angularVelocity = zero;
    agent.gameObject.transform.position = new Vector3(0, 0.5f, -3.5f);

    Vector3 nextPosition = GetNextPositionFromCSV();

    Rigidbody target2 = gameObject.GetComponent<Rigidbody>();
    target2.velocity = zero;
    target2.angularVelocity = zero;
    transform.position = nextPosition; // 🔥 다음 위치로 이동

    initPoint = transform.position - agent.transform.position;
}
```

- 코드를 통해 Test할 모델이 모두 같은 Environment에서 Test 가능

- DQNTest.py 실행 및 Unity 실행



```
Prediction Count,Clear Status,total_reward
101,False,-37.34000000000001
100,False,-37.26999999999999
30,True,111.67
100,False,-39.790000000000035
100,False,-36.53999999999999
55,True,123.89
100,False,-39.430000000000035
100,False,-37.94999999999999
56,True,125.08
51,True,120.68
100,False,-36.62000000000002
41,True,117.62
46,True,119.74
39,True,116.11
100,False,-43.15999999999994
100,False,-37.98999999999994
100,False,-37.07000000000004
100,False,-40.50000000000007
100,False,-41.88000000000002
100,False,-35.609999999999914
```

■ 실제 강화학습된 모델이 게임을 진행

■ Unity의 게임 화면으로 확인해본결과 Clear에 도달하지 못했던 Agent는 빨간색 큐브 앞에서 전진과 후진을 반복하며 500번의 Action을 진행

5. DQN모델의 Reward 정책 분석

1) 정책 분석

- 수식 분석

- 현재 DQN 모델의 Reward는 벨만 최적 방정식을 적용

$$target = r + \gamma \max a' Q^*(s', a')$$

- 벨만 최적 방정식은 Q값의 목표값을 도출
- r은 현재 상태의 즉시보상
- gamma는 감가율, 미래의 보상을 신뢰하는 피라미터
- max를 통해 다음 상태 s'에서 가능한 행동중 최대 Q값 선택
- 현재 DQN 모델의 Reward는 벨만 최적 방정식을 적용

- 코드 분석

- 기존 벨만 최적 방정식에 에피소드 상태 추가
- 에피소드가 끝나는 dones가 1이 되면 현재 보상인 rewards만 반영
- 다음상태의 Q값 next_q_values중 최대 Q값을 선택

```
target = rewards + gamma * np.max(next_q_values, axis=1) * (1 - dones)
```

- Loss 반영

```
target = rewards + gamma * np.max(next_q_values, axis=1) * (1 - dones)
one_hot_actions = tf.one_hot(actions, action_size)
q_value = tf.reduce_sum(q_values * one_hot_actions, axis=1)

loss = tf.reduce_mean(tf.square(target - q_value))  # MSE Loss
```

- 벨만 최적 방정식을 이용해 Target Q_value를 도출
- One-Hot encoding을 통해 현재 Agent가 취한 행동을 One-Hot Vector로 변환
- q_values와 One-Hot Vector를 원소별로 곱해 Vector의 합을 구함
- 곱해진 Vector의 모든 원소를 더해 q_value 생성
- Target Q_value와 q_value의 MSE를 계산해 학습에 적용

6. Reward 정책 개선 아이디어 및 학습결과 예상

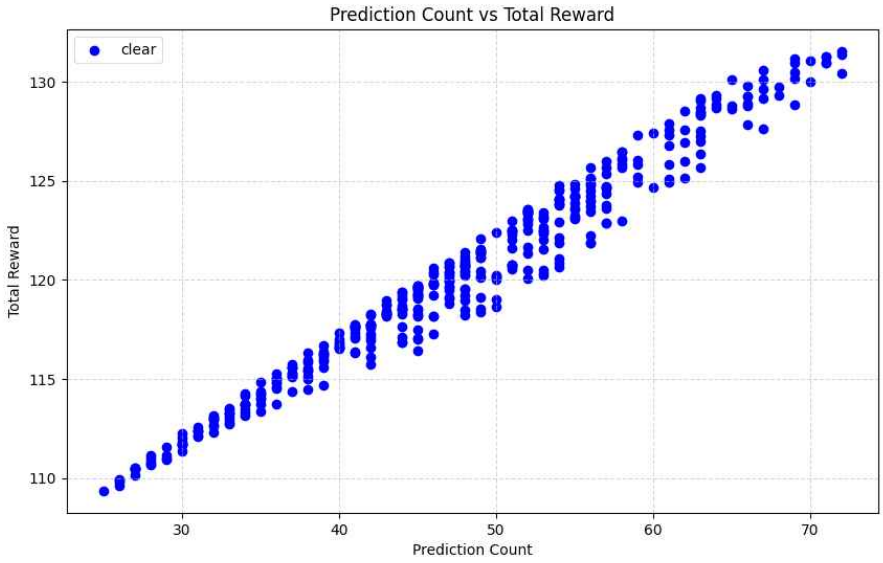
1) 정책개선 아이디어

■ 현재 Reward 정책으로 학습된 모델은 Agent가 빨간색 Cube 앞에서 앞뒤로 움직이며 Clear 하지 못하는 문제를 일으켰음

```
Prediction Count,Clear Status,total_reward
101,False,-37.34000000000001
100,False,-37.26999999999999
30,True,111.67
100,False,-39.790000000000035
100,False,-36.53999999999999
55,True,123.89
100,False,-39.430000000000035
100,False,-37.94999999999999
```

■ 이는 행동에 대한 페널티가 없어 목표에 가까워질 때 게임을 Clear 해 최대 Reward를 얻으려는 것 보다 Agent가 즉시 보상을 얻기 위해 앞뒤로 움직이며 Reward를 최대화 하려는 것으로 보임

■ 실제로 게임을 Clear 한 Episode에서 Action(Prediction Count)이 커질수록, 전체 보상(Total Reward) 이 커지는 것을 시각화 할 수 있음



■ 기존 Reward에 페널티를 추가해 Action이 늘어날 때 페널티가 부여되도록 Reward 정책 수정

2) 학습결과 예상

- 기존 모델과 동일하게 목표에 가까워질수록 Reward가 커져 Agent가 빨간색 큐브로 접근
- Action이 늘어날수록 Reward가 페널티로 인해 줄어들기 때문에 적은 Action으로 빨간색 큐브에 도달하도록 목적 변경

3) 학습결과

- MyCheck.cs에 페널티 코드 추가

- step마다 페널티가 선형으로 증가하도록 설정
- 큐브에 접근하게 되면 점수가 올라가지만, Action이 많아지면 페널티가 커지므로 최소한의 Action을 선호하도록 조정

```
// next state
state = transform.position - agent.transform.position;
float reward = (Vector3.Magnitude(initPoint) - Vector3.Magnitude(state)) / Vector3.Magnitude(initPoint);
if (done == 1)
{
    reward = 100;
}


reward += -0.01f * step;

if (step > 100)
{
    done = 1;
    reward = -100;
}
SendMessageToServer(string.Format("{0:F2},{1:F2},{2:F2},{3}", state.x, state.z, reward, done));
```

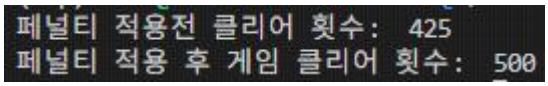
- DQNTest.py 실행

- 새로운 모델의 실제 게임결과를 저장

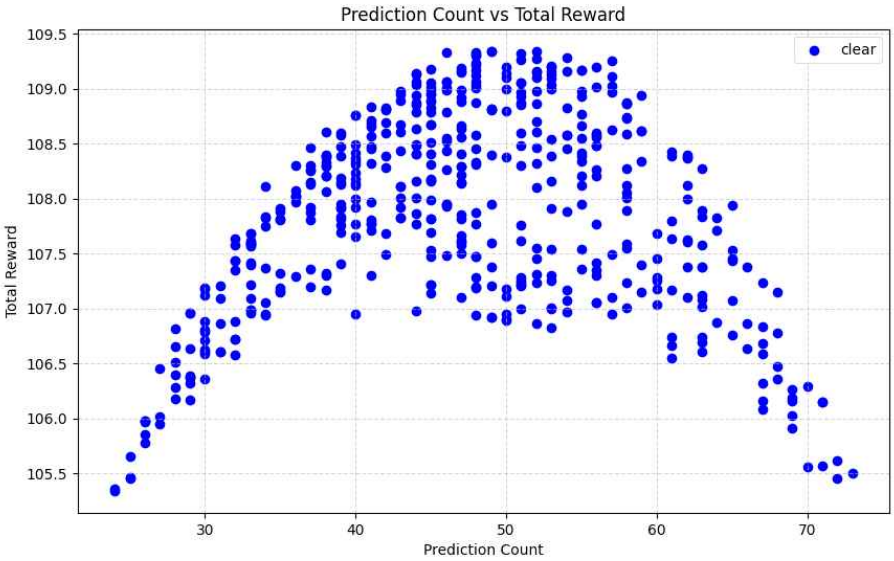
```
Prediction Count,Clear Status,total_reward
58,True,108.59
51,True,108.60000000000001
30,True,106.71
35,True,107.19
42,True,108.28
52,True,106.86
51,True,108.30000000000001
46,True,108.52
55,True,108.39
50,True,106.89999999999999
```

- 이전 결과와의 비교

- 페널티 없이 학습했던 모델은 425개의 게임을 Clear
- 페널티 적용 후에는 500개의 게임을 모두 Clear
- 유의미할 정도의 성능 향상을 보임



- Reward 역시 움직임이 많아질수록 Total Reward가 줄어드는 것으로 확인되며 Agent가 학습의 방향대로 움직이는 것을 확인할 수 있음



- 페널티 Reward를 추가해 학습한 모델이 이전 Reward 정책을 사용한 모델보다 안정적이며, 게임을 모두 Clear 하는 모습을 보임.
- 이는 정책개선 아이디어로 유의미한 성과를 얻은 것으로 생각됨