AI+X 프로젝트 블로그 초안
강화학습을 이용한 물류창고 로봇 경로 최적화

데이터사이언스학부 데이터사이언스전공 2023062760 정병윤
데이터사이언스학부 심리뇌과학전공 2023066735 전용현

1. Proposal

1-1. Motivation

물류창고 내에서의 로봇 이동 효율성은 작업 생산성에 큰 영향을 미친다. 기존 수작업 동선 설계가 아닌, 강화학습 기반 접근을 통해 로봇이 스스로 최적의 경로를 학습하도록 하고자 한다.

1-2. 목표

강화학습 알고리즘(Q-learning, DQN)을 통해 창고 내 픽업 → 드랍 → 리턴 작업을 수행할 수 있는 최적 경로 학습, 학습된 모델의 성능을 시각화하여 비교 분석

2. Dataset

실세계 데이터를 사용하는 대신, Python 코드 기반의 시뮬레이션 환경 구축
5x5 또는 10x10 격자(Grid) 환경 내에서 로봇의 위치, 상태, 목표 지점을 구성
픽업(Pickup), 드랍(Drop), 리턴(Return) 위치를 고정하여 실험 일관성 확보

3. Methodology

3-1. 적용 알고리즘

Q-learning (mlpg-Q.py)
상태-행동 값을 테이블로 저장하며 업데이트, 작은 상태 공간에서 빠르게 수렴
DQN (Deep Q-Network, dqn-p.py)
상태 공간 확장 대응을 위한 신경망 기반 Q-function 근사
경험 재사용 (Experience Replay), 타깃 네트워크 적용

3-2. 상태 및 행동

상태: 에이전트의 위치(x, y) + 아이템 보유 여부(0/1)
행동: 상, 하, 좌, 우 (4가지)
• 보상 설계
유효 이동: -1
벽 충돌: -5
픽업 성공: +10
드랍 성공: +20
리턴 도달: +5

4. Conclusion (결론 및 역할 분담)

4-1. 중간 결론

Q-learning은 구조는 단순하나 확장성에 한계가 있음
DQN은 복잡한 구조지만 더 나은 정책을 학습함

4-2. 이후 계획

장애물 추가 및 다양한 맵 구성 실험
DQN 하이퍼파라미터 조정
블로그 내용 보강 및 최종 영상 제작

+α 구현한 코드

1. mlpg-q.py

```
import pygame
import numpy as np
import random
import time

# ----- 환경 설정 -----
GRID_SIZE = 5
CELL_SIZE = 100
WINDOW_SIZE = GRID_SIZE * CELL_SIZE

PICKUP_LOC = (0, 0)
DROP_LOC = (4, 4)
RETURN_LOC = (2, 2)

actions = ['up', 'down', 'left', 'right']
action_dict = {'up': (-1, 0), 'down': (1, 0), 'left': (0, -1), 'right': (0, 1)}

# ----- 환경 클래스 -----
class WarehouseEnv:
    def __init__(self):
        self.reset()

    def reset(self):
        self.agent_pos = (random.randint(0, GRID_SIZE-1), random.randint(0, GRID_SIZE-1))
        self.has_item = False
        self.done = False
        return self.get_state()

    def get_state(self):
        return (*self.agent_pos, int(self.has_item))

    def step(self, action):
        dx, dy = action_dict[action]
        x, y = self.agent_pos
        nx, ny = x + dx, y + dy

        if 0 <= nx < GRID_SIZE and 0 <= ny < GRID_SIZE:
```

```
            self.agent_pos = (nx, ny)

        reward = -1
        if self.agent_pos == PICKUP_LOC and not self.has_item:
            self.has_item = True
            reward = 10
        elif self.agent_pos == DROP_LOC and self.has_item:
            self.has_item = False
            reward = 20
        ##elif self.agent_pos == RETURN_LOC and not self.has_item:
        ##    self.done = True
        ##    reward = 5

        return self.get_state(), reward, self.done

# ----- Q-learning -----
Q_table = {}
alpha = 0.1
gamma = 0.9
epsilon = 0.1

env = WarehouseEnv()
for episode in range(1000):
    state = env.reset()
    for _ in range(50):
        if random.random() < epsilon:
            action = random.choice(actions)
        else:
            q_values = [Q_table.get((state, a), 0) for a in actions]
            action = actions[np.argmax(q_values)]

        next_state, reward, done = env.step(action)
        best_next_q = max([Q_table.get((next_state, a), 0) for a in actions])
        Q_table[(state, action)] = Q_table.get((state, action), 0) + alpha * (
            reward + gamma * best_next_q - Q_table.get((state, action), 0)
        )
        state = next_state
        if done:
            break

print("✅ 학습 완료")

# ----- Pygame GUI -----
pygame.init()
screen = pygame.display.set_mode((WINDOW_SIZE, WINDOW_SIZE))
pygame.display.set_caption("Warehouse Robot RL")
clock = pygame.time.Clock()

def draw_grid(agent_pos, has_item):
    screen.fill((255, 255, 255))
```

```
    for x in range(GRID_SIZE):
        for y in range(GRID_SIZE):
            rect = pygame.Rect(y*CELL_SIZE, x*CELL_SIZE, CELL_SIZE, CELL_SIZE)
            pygame.draw.rect(screen, (200, 200, 200), rect, 1)

            if (x, y) == PICKUP_LOC:
                pygame.draw.rect(screen, (255, 255, 0), rect)  # 노랑
            elif (x, y) == DROP_LOC:
                pygame.draw.rect(screen, (255, 165, 0), rect)  # 주황
            elif (x, y) == RETURN_LOC:
                pygame.draw.rect(screen, (0, 255, 0), rect)    # 초록

    ax, ay = agent_pos
    agent_rect = pygame.Rect(ay*CELL_SIZE+20, ax*CELL_SIZE+20, 60, 60)
    pygame.draw.ellipse(screen, (255, 0, 0), agent_rect)  # 빨강 에이전트

    if has_item:
        pygame.draw.circle(screen, (0, 0, 255), agent_rect.center, 10)  # 파랑 박스

    pygame.display.flip()

# ----- GUI 루프 -----
state = env.reset()
running = True
step_count = 0

while running and step_count < 3000:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    q_values = [Q_table.get((state, a), 0) for a in actions]
    action = actions[np.argmax(q_values)]
    next_state, reward, done = env.step(action)

    draw_grid(env.agent_pos, env.has_item)
    print(f"{step_count} 위치: {env.agent_pos}, 액션: {action}, 보상: {reward}")
    state = next_state
    step_count += 1

    if done:
        time.sleep(1)
        running = False

    clock.tick(2)

pygame.quit()
```

2. dqn-p.py

```
import numpy as np
import random
import torch
import torch.nn as nn
import torch.optim as optim
import pygame
import time
from collections import deque

# ----- 환경 설정 -----
GRID_SIZE = 5
CELL_SIZE = 100
WINDOW_SIZE = GRID_SIZE * CELL_SIZE

PICKUP_LOC = (0, 0)
DROP_LOC = (4, 4)
RETURN_LOC = (2, 2)

actions = ['up', 'down', 'left', 'right']
action_dict = {'up': (-1, 0), 'down': (1, 0), 'left': (0, -1), 'right': (0, 1)}

# ----- 환경 클래스 -----
class WarehouseEnv:
    def __init__(self):
        self.reset()

    def reset(self):
        self.agent_pos = (random.randint(0, GRID_SIZE-1), random.randint(0, GRID_SIZE-1))
        self.has_item = False
        self.delivered = False
        self.done = False
        return self.get_state()

    def get_state(self):
        return (*self.agent_pos, int(self.has_item))

    def step(self, action):  # ← 반드시 같은 수준 들여쓰기
        dx, dy = action_dict[action]
        x, y = self.agent_pos
        nx, ny = x + dx, y + dy

        valid_move = 0 <= nx < GRID_SIZE and 0 <= ny < GRID_SIZE
        if valid_move:
            self.agent_pos = (nx, ny)
            reward = -1
        else:
            reward = -5

        if not self.has_item:
            goal = PICKUP_LOC
```

```
        elif self.has_item and not self.delivered:
            goal = DROP_LOC
        else:
            goal = RETURN_LOC

        distance = abs(self.agent_pos[0] - goal[0]) + abs(self.agent_pos[1] - goal[1])
        shaping_reward = -0.1 * distance
        reward += shaping_reward if valid_move else 0

        if self.agent_pos == PICKUP_LOC and not self.has_item:
            self.has_item = True
            reward = 10
        elif self.agent_pos == DROP_LOC and self.has_item:
            self.has_item = False
            self.delivered = True
            reward = 20
        elif self.agent_pos == RETURN_LOC and self.delivered:
            self.done = True
            reward = 5

        return self.get_state(), reward, self.done


# ----- DQN 구성 -----
class QNetwork(nn.Module):
    def __init__(self, state_size, action_size):
        super(QNetwork, self).__init__()
        self.fc1 = nn.Linear(state_size, 64)
        self.fc2 = nn.Linear(64, 64)
        self.fc3 = nn.Linear(64, action_size)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        return self.fc3(x)

state_size = 3
action_size = 4

gamma = 0.99
epsilon = 1.0
epsilon_min = 0.1
epsilon_decay = 0.995
batch_size = 32
buffer_size = 10000
target_update_freq = 10
lr = 0.001

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
policy_net = QNetwork(state_size, action_size).to(device)
target_net = QNetwork(state_size, action_size).to(device)
```

```
target_net.load_state_dict(policy_net.state_dict())
target_net.eval()

optimizer = optim.Adam(policy_net.parameters(), lr=lr)
memory = deque(maxlen=buffer_size)

env = WarehouseEnv()

# ----- 학습 -----
for episode in range(500):
    state = np.array(env.reset(), dtype=np.float32)
    for t in range(50):
        if random.random() < epsilon:
            action_idx = random.randint(0, action_size - 1)
        else:
            with torch.no_grad():
                state_tensor = torch.FloatTensor(state).unsqueeze(0).to(device)
                q_values = policy_net(state_tensor)
                action_idx = torch.argmax(q_values).item()

        next_state, reward, done = env.step(actions[action_idx])
        next_state = np.array(next_state, dtype=np.float32)
        memory.append((state, action_idx, reward, next_state, done))
        state = next_state

        if len(memory) >= batch_size:
            batch = random.sample(memory, batch_size)
            states, actions_, rewards, next_states, dones = zip(*batch)

            states = torch.FloatTensor(states).to(device)
            actions_ = torch.LongTensor(actions_).unsqueeze(1).to(device)
            rewards = torch.FloatTensor(rewards).unsqueeze(1).to(device)
            next_states = torch.FloatTensor(next_states).to(device)
            dones = torch.BoolTensor(dones).unsqueeze(1).to(device)

            q_values = policy_net(states).gather(1, actions_)
            with torch.no_grad():
                next_q = target_net(next_states).max(1, keepdim=True)[0]
                target_q = rewards + gamma * next_q * (~dones)

            loss = nn.MSELoss()(q_values, target_q)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

        if done:
            break

    if episode % target_update_freq == 0:
        target_net.load_state_dict(policy_net.state_dict())
```

```
    if epsilon > epsilon_min:
        epsilon *= epsilon_decay

print("\n✅ DQN 학습 완료")
# --- 정책 저장 ---
torch.save(policy_net.state_dict(), "dqn_policy.pth")
print("💾 정책이 dqn_policy.pth 파일로 저장되었습니다.")

# ----- Pygame GUI -----
pygame.init()
screen = pygame.display.set_mode((WINDOW_SIZE, WINDOW_SIZE))
pygame.display.set_caption("Warehouse Robot DQN")
clock = pygame.time.Clock()


def draw_grid(agent_pos, has_item):
    screen.fill((255, 255, 255))
    for x in range(GRID_SIZE):
        for y in range(GRID_SIZE):
            rect = pygame.Rect(y*CELL_SIZE, x*CELL_SIZE, CELL_SIZE, CELL_SIZE)
            pygame.draw.rect(screen, (200, 200, 200), rect, 1)
            if (x, y) == PICKUP_LOC:
                pygame.draw.rect(screen, (255, 255, 0), rect)  # 노랑
            elif (x, y) == DROP_LOC:
                pygame.draw.rect(screen, (255, 165, 0), rect)  # 주황
            elif (x, y) == RETURN_LOC:
                pygame.draw.rect(screen, (0, 255, 0), rect)    # 초록

    ax, ay = agent_pos
    agent_rect = pygame.Rect(ay*CELL_SIZE+20, ax*CELL_SIZE+20, 60, 60)
    pygame.draw.ellipse(screen, (255, 0, 0), agent_rect)  # 빨강 에이전트
    if has_item:
        pygame.draw.circle(screen, (0, 0, 255), agent_rect.center, 10)  # 파랑 박스
    pygame.display.flip()

# ----- 실행 시각화 -----
state = np.array(env.reset(), dtype=np.float32)
running = True
step_count = 0

while running and step_count < 50:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    with torch.no_grad():
        state_tensor = torch.FloatTensor(state).unsqueeze(0).to(device)
        q_values = policy_net(state_tensor)
        action_idx = torch.argmax(q_values).item()

    next_state, reward, done = env.step(actions[action_idx])
```

```
    draw_grid(env.agent_pos, env.has_item)
    print(f"Step {step_count} 위치: {env.agent_pos}, 액션: {actions[action_idx]}, 보상: {reward}")
    state = np.array(next_state, dtype=np.float32)
    step_count += 1

    if done:
        time.sleep(1)
        running = False

    clock.tick(2)

pygame.quit()
```

3. dqn-1010.py

```
import numpy as np
import torch
import torch.nn as nn
import pygame
import time
import random

# ----- 환경 설정 -----
GRID_SIZE = 10
CELL_SIZE = 60
WINDOW_SIZE = GRID_SIZE * CELL_SIZE

PICKUP_LOC = (0, 0)
DROP_LOC = (9, 9)
RETURN_LOC = (5, 5)

actions = ['up', 'down', 'left', 'right']
action_dict = {'up': (-1, 0), 'down': (1, 0), 'left': (0, -1), 'right': (0, 1)}

# ----- 환경 클래스 -----
class WarehouseEnv:
    def __init__(self):
        self.reset()

    def reset(self):
        self.agent_pos = (random.randint(0, GRID_SIZE-1), random.randint(0, GRID_SIZE-1))
        self.has_item = False
        self.delivered = False
        self.done = False
        return self.get_state()

    def get_state(self):
        return (*self.agent_pos, int(self.has_item))

    def step(self, action):
        dx, dy = action_dict[action]
```

```
        x, y = self.agent_pos
        nx, ny = x + dx, y + dy

        if 0 <= nx < GRID_SIZE and 0 <= ny < GRID_SIZE:
            self.agent_pos = (nx, ny)

        reward = -1
        if self.agent_pos == PICKUP_LOC and not self.has_item:
            self.has_item = True
            reward = 10
        elif self.agent_pos == DROP_LOC and self.has_item:
            self.has_item = False
            self.delivered = True
            reward = 20
        elif self.agent_pos == RETURN_LOC and self.delivered:
            self.done = True
            reward = 5

        return self.get_state(), reward, self.done

# ----- DQN 네트워크 정의 (저장된 것과 동일해야 함) -----
class QNetwork(nn.Module):
    def __init__(self, state_size, action_size):
        super(QNetwork, self).__init__()
        self.fc1 = nn.Linear(state_size, 64)
        self.fc2 = nn.Linear(64, 64)
        self.fc3 = nn.Linear(64, action_size)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        return self.fc3(x)

# ----- 모델 로드 -----
state_size = 3
action_size = 4

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
policy_net = QNetwork(state_size, action_size).to(device)
policy_net.load_state_dict(torch.load("dqn_policy.pth", map_location=device))
policy_net.eval()

# ----- 시각화 -----
pygame.init()
screen = pygame.display.set_mode((WINDOW_SIZE, WINDOW_SIZE))
pygame.display.set_caption("Warehouse Robot DQN (10x10 Test)")
clock = pygame.time.Clock()

def draw_grid(agent_pos, has_item):
    screen.fill((255, 255, 255))
    for x in range(GRID_SIZE):
```

```
            for y in range(GRID_SIZE):
                rect = pygame.Rect(y*CELL_SIZE, x*CELL_SIZE, CELL_SIZE, CELL_SIZE)
                pygame.draw.rect(screen, (200, 200, 200), rect, 1)
                if (x, y) == PICKUP_LOC:
                    pygame.draw.rect(screen, (255, 255, 0), rect)  # 노랑
                elif (x, y) == DROP_LOC:
                    pygame.draw.rect(screen, (255, 165, 0), rect)  # 주황
                elif (x, y) == RETURN_LOC:
                    pygame.draw.rect(screen, (0, 255, 0), rect)    # 초록

        ax, ay = agent_pos
        agent_rect = pygame.Rect(ay*CELL_SIZE+10, ax*CELL_SIZE+10, 40, 40)
        pygame.draw.ellipse(screen, (255, 0, 0), agent_rect)  # 빨강 에이전트
        if has_item:
            pygame.draw.circle(screen, (0, 0, 255), agent_rect.center, 8)  # 파랑 박스
        pygame.display.flip()

    # ----- 시뮬레이션 -----
    env = WarehouseEnv()
    state = np.array(env.reset(), dtype=np.float32)
    running = True
    step_count = 0

    while running and step_count < 100:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False

        with torch.no_grad():
            state_tensor = torch.FloatTensor(state).unsqueeze(0).to(device)
            q_values = policy_net(state_tensor)
            action_idx = torch.argmax(q_values).item()

        next_state, reward, done = env.step(actions[action_idx])
        draw_grid(env.agent_pos, env.has_item)
        print(f"Step {step_count} 위치: {env.agent_pos}, 액션: {actions[action_idx]}, 보상: {reward}")
        state = np.array(next_state, dtype=np.float32)
        step_count += 1

        if done:
            time.sleep(1)
            running = False

        clock.tick(4)

    pygame.quit()
```