Performance Comparison of Q-learning and DQN in Maze Solving Problem

Jinseo Choi

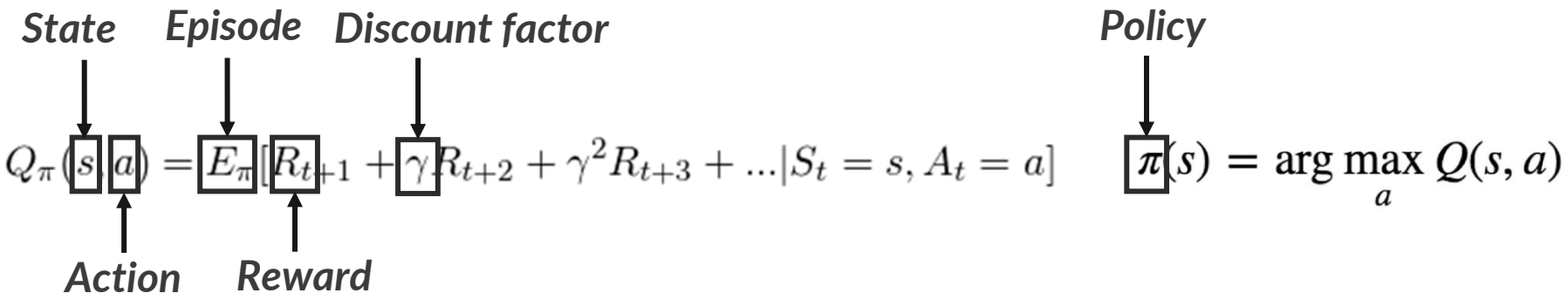
CHANGWON NATIONAL UNIVERSITY

Abstract

- 특정문제에서 강화학습 알고리즘의 성능을 비교하고자 함
  - 해결시간, 성능
- 5x5 Grid에서 장애물을 피해 목표지점에 도달하는 환경을 구현
  - Problem solving 알고리즘으로 강화학습 적용
- 비교대상 강화학습 알고리즘 구현
  - Q-learning
  - DQN(Deep Q Network)
- 성능 측정

Introduction

- Q-learning
  - 학습모델이 환경에 대한 정보 없이 여러 번의 시행(Episode)을 거쳐 점차적으로 학습
  - 현재 상태로부터 시작하여 모든 연속적인 단계들을 거쳐 전체 보상의 예측값을 극대화
  - Q: 현재상태(State)에서 에이전트가 취한 행동(Action)으로 얻어지는 보상(Reward)
  - Q-value: 현재상태에서 에피소드가 종료될 때 까지 행동을 취했을 때의 보상 기댓값
  - Q-learning 정책(Policy): 곧 현재상태에서의 Q-value가 최댓값을 가지는 행동을 선택



Introduction

- Q-learning algorithm
  - 초기상태에서 Q-value를 어떻게 구할 것인가?
  - 다음상태의 Q-value가 존재한다고 가정
  - $Q(s,a) \leftarrow R + \max(Q(\acute{s},\acute{a}))$
  - Learning rate 적용 시

    $$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a'))$$

  - 초기상태의 모든 Q-value를 0으로 초기화
  - 여러 번의 에피소드를 시행하면서 Q-value가 업데이트 되면서 Action이 최적화

Algorithm 3: Q-learning Algorithm

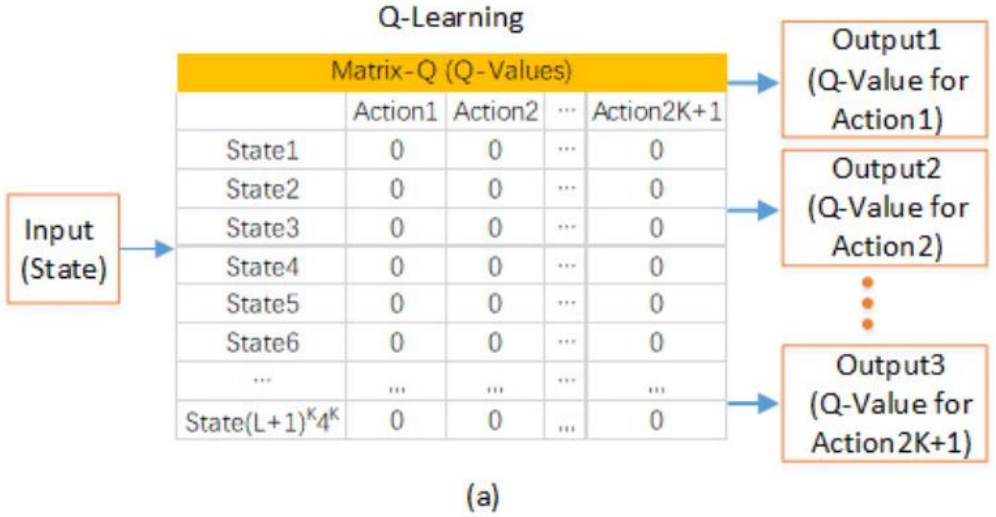
Procedure `simulate()`
- $s \sim \rho(s_0)$ `// Sample start state`
- **for** $t = 1$ *to* T **do**
  - $a \leftarrow \begin{cases} argmax_a Q(s,a), & if\ p > \epsilon \\ a \sim U(1,|A|), & otherwise \end{cases}$ `// Epsilon greedy sampling`
  - $s' \leftarrow T(s,a)$ `// Simulate action`
  - $r = R(s,a,s')$ `// Observe reward`
  - $D \leftarrow D \cup < s,a,r,s' >$ `// Store tuple in memory buffer`
  - $s \leftarrow s'$
- **end**
- **return** D

Q-Learning
- Initialize $Q(s,a) = 0,\ \forall(s,a)$, Initialize D
- **while** *Q values are not converged* **do**
  - $D \leftarrow$`simulate()`
  - **for** $< s,a,s',r >$ *in* D **do**
    - $Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a'))$
  - **end**
- **end**
- **return** $Q(s,a) \approx Q^*(s,a)\ \forall(s,a)$

Q-learning algorithm pseudo code, Georgia tech, Bootcamp Summer 2020 Week 3 – Value Iteration and Q-learning (gatech.edu)

Introduction

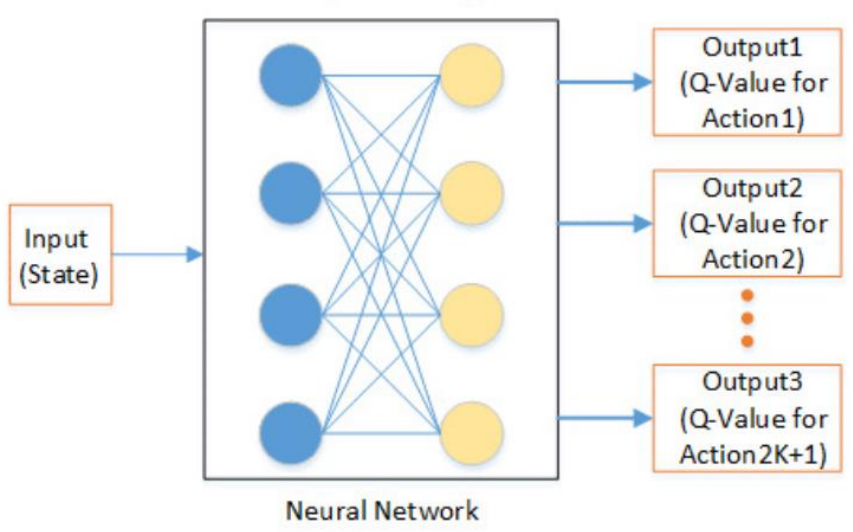
- Q-table
  - 모든 상태(State)에서 가능한 행동들에 대한 모든 Q-value를 저장
  - 에피소드 수행 시 업데이트
  - State, Action의 개수가 증가할 수록 Q-table 생성에 많은 컴퓨팅 리소스가 요구됨



Q-learning algorithm, Huang, Chong & Chen, Gaojie & GONG, Yi. (2021). Delay-Constrained Buffer-Aided Relay Selection in the Internet of Things With Decision-Assisted Reinforcement Learning. IEEE Internet of Things Journal. PP. 10.1109/JIOT.2021.3051239.

Introduction

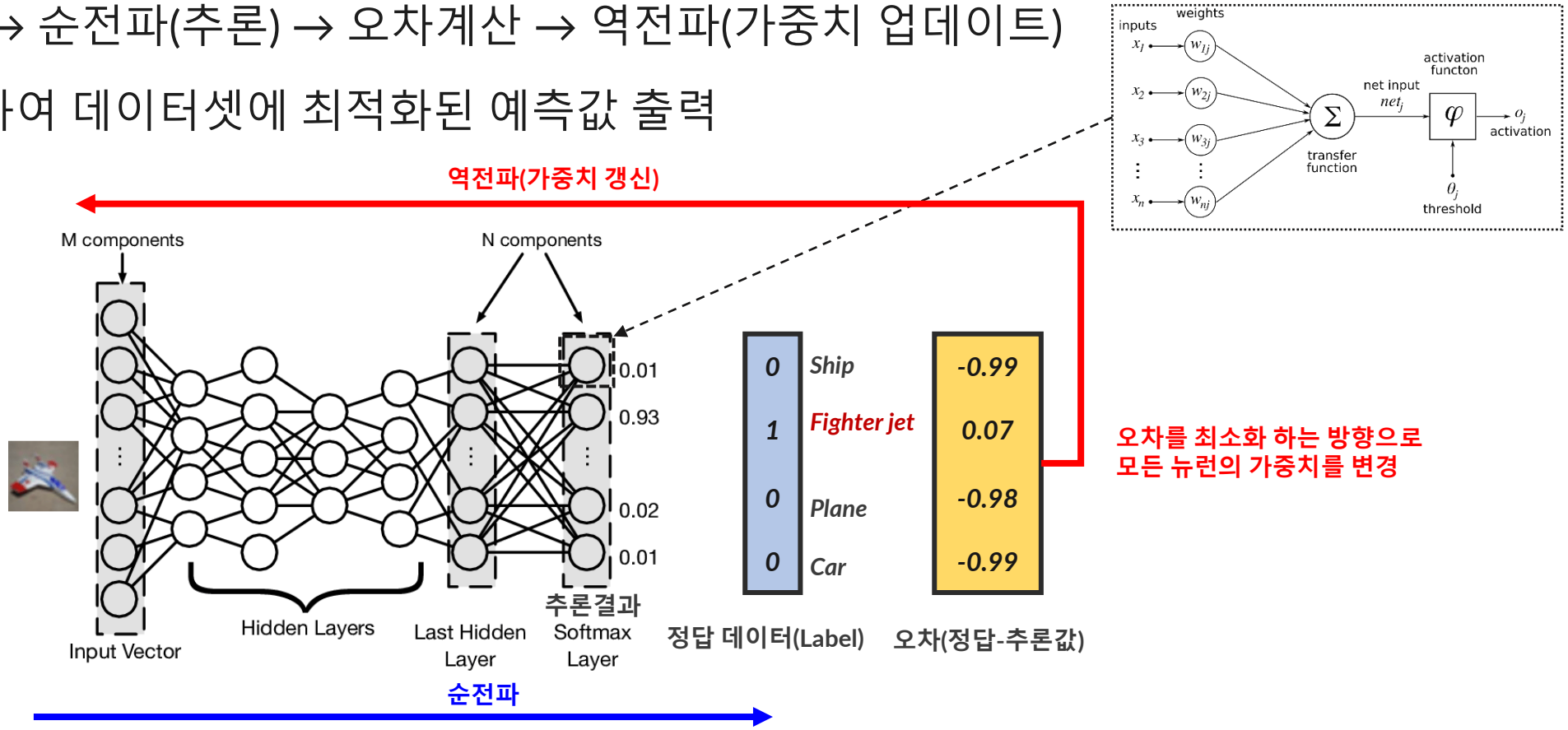
- Deep Q learning
  - Q-table의 scalability 문제의 해결방법으로 제안
  - 현재 상태에서 가능한 행동들에 대한 Q-value를 심층신경망(DNN)으로 예측
  - DNN이 예측한 Q-value 중 가장 큰 값을 행동을 다음행동으로 결정



Q-learning algorithm pseudo code, Georgia tech, Bootcamp Summer 2020 Week 3 – Value Iteration and Q-learning (gatech.edu)

Introduction

- DNN(Deep Neural Network)
  - 여러 개의 인공신경망이 연결된 구조
  - 훈련데이터 입력 → 순전파(추론) → 오차계산 → 역전파(가중치 업데이트)
  - 훈련과정을 반복하여 데이터셋에 최적화된 예측값 출력



Introduction

- Deep Q learning-Algorithm
  - 입력데이터: (상태, 행동)
  - 추론 값: 현재 상태에서 주어진 행동에 대한 Q-value
  - 가능한 행동들에 대해 모든 Q-value 추론
  - 정답데이터로 가장 큰 Q-value 값을 가진 행동을 수행한 다음 상태에서의 Max Q-value를 가짐
  - 오차 = (다음상태 Q-value 최댓값) – (현재상태 Q-value 최댓값)^2 **(Q-learning 의 Q-table을 대체)**
  - 오차값을 통해 모델 학습

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a'))$$

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot);s'\sim\mathcal{E}}\left[\left(r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) - Q(s,a;\theta_i)\right)\nabla_{\theta_i} Q(s,a;\theta_i)\right]. \quad (3)$$

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

DQN algorithm pseudo code, Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning." *arXiv preprint arXiv:1312.5602* (2013).

Introduction

- Deep Q learning-Algorithm
  - Q-value 계산을 위해 학습모델의 추론과정을 2번 수행(Computing Overhead)
  - 학습모델 훈련을 위해 현재 (상태, 행동, 보상, 다음상태) 값을 버퍼에 저장(Storage Overhead)
  - 학습모델 예측이 수렴하기까지 비교적 시간이 오래걸림

Implementation

- Environment
  - OS: Ubuntu 20.04
  - Language: Python3.8
  - Framework: pygame(grid 구현), Numpy, Tensorflow, Keras

Implementation

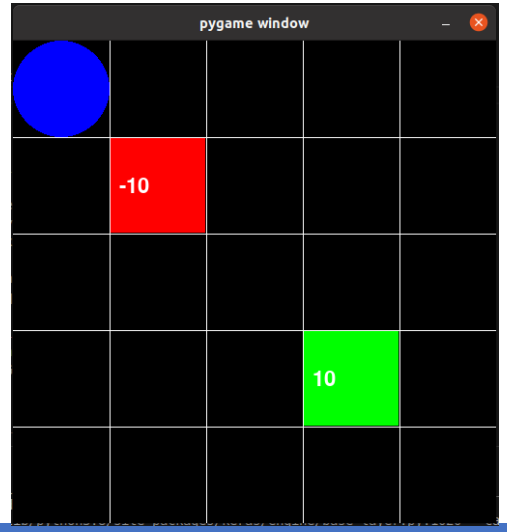
- 5x5 Grid
  - 보상정책: 목표 도달 시 보상 10, 함정 보상 -10, 이동시 보상 차감 없음, 나머지 셀 보상 0
  - 각 셀의 상태를 좌표 (x,y) 튜플으로 정의
  - 행동은 ACTION에 정의된 4방향 리스트의 인덱스 값을 사용
  - 각 셀의 보상값을 좌표:보상 {(x,y) : reward} 딕셔너리로 정의
  - 이벤트 셀(목표, 함정)을 상수로 정의

Constant.py

```
WHITE_COLOR_RGB = (255,255,255)
BLACK_COLOR_RGB = (0,0,0)
RED_COLOR_RGB = (255, 0, 0)
GREEN_COLOR_RGB = (0, 255, 0)
BLUE_COLOR_RGB = (0, 0, 255)
WIDTH = 500
NUM_ROWS = 5
NUM_EPISODES=30
EPSILON = 4 # Integer 1-10. 2 = 20% random, 3 = 30% random ...
CELL_SIZE = WIDTH/NUM_ROWS
ACTIONS = ['left', 'right', 'up', 'down']
TERMINAL_CELLS = [(2, 2)]
CELL_VALUES = [((1, 1), -10),((3, 3), 10)]
#((6, 1), -10),
#((1, 4), -10),
#((2, 7), -40),
#((2, 8), -10),
```

Implementation

- Grid 클래스 구성
  - 셀, 보상 초기화
  - 이동
  - 행동에 따른 보상 변화
  - 범위 초과, 목표 도달여부
  - 현재 좌표, 보상값 반환, 메서드 구현



Player.py

```python
import pygame

class Player(object):
    def __init__(self, pos=(0,0)):
        self.currPos = pos
        self.score = 0

    def move(self, action):
        oldX, oldY = self.currPos
        if action == 'left':
            self.currPos = (oldX - 1, oldY)
        elif action == 'right':
            self.currPos = (oldX + 1, oldY)
        elif action == 'up':
            self.currPos = (oldX, oldY + 1)
        elif action == 'down':
            self.currPos = (oldX, oldY - 1)
        elif action == 'stay':
            self.currPos = (oldX, oldY)



    def updateCurrPos(self, newCoords):
        self.currPos = newCoords
    def updateScore(self, newPoints):
        self.score += newpoints
    def getCurrCoords(self):
        return self.currPos
```

Board.py

```python
from Constants import NUM_ROWS, CELL_VALUES, TERMINAL_CELLS, ACTIONS

class Board(object):
    cellValues = {} # Maps cell (x, y) to reward r

    def __init__(self, numRows = NUM_ROWS):
        self.numRows = numRows
        self.initCellRewards()

    def initCellRewards(self):
        for xPos in range(NUM_ROWS):
            for yPos in range(NUM_ROWS):
                self.cellValues[(xPos, yPos)] = 0

    def createPenaltyCells(self):
        for cell, val in CELL_VALUES:
            self.cellValues[cell[0], cell[1]] = val

    def isTerminalCell(self, coord):
        return coord in TERMINAL_CELLS

    def isValidCell(self, coord, action):
        xCoord, yCoord = self.getCellAfterAction(coord, action)
        return (0 <= xCoord < NUM_ROWS  and 0 <= yCoord < NUM_ROWS)

    def getCellAfterAction(self, coord, action):
        xCoord, yCoord = coord
        if action == 'left':
            xCoord-=1
        elif action == 'right':
            xCoord+=1
        elif action == 'up':
            yCoord+=1
        elif action == 'down':
            yCoord-=1

        return (xCoord, yCoord)

    def getCellValue(self, coord):
        return self.cellValues[coord]
```

Implementation

- Q-learning 구현
  - Run_Qlrn.py: Q-learning 실행
  - Qlearner.py: Q-table 생성, 업데이트 로직

Implementation

- Run_Qlrn.py
  - Grid관련 객체 로드
  - 셀, Q-table 초기화(0)
  - Q-table 객체 생성
  - Q-table에서 최적경로 추출
  - 시행

```
pygame.init()
w = Window()
p = Player()
b = Board()
b.createPenaltyCells()
w.drawSurface(b, p)

while True:
    # exit call
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            exit()

    q = QLearner(b)
    qTable = q.learn()
    dirToGo = {}
    for k, v in qTable.items():
        # Q table iteration
        dirToGo[k] = max(v, key=v.get)
        print("k: {}, v: {}".format(k,v))
    #
    print("_____-")
    for k, v in dirToGo.items():
        print("k: {}, v: {}".format(k,v))

    currNode = (p.getCurrCoords())
    time.sleep(2)
    while(not b.isTerminalCell(currNode)):

        time.sleep(2)
        p.move(dirToGo[currNode])
        currNode = p.getCurrCoords()
        # print(currNode)
        w.colorCell(currNode, (0, 0, 255))
        pygame.display.update()
```

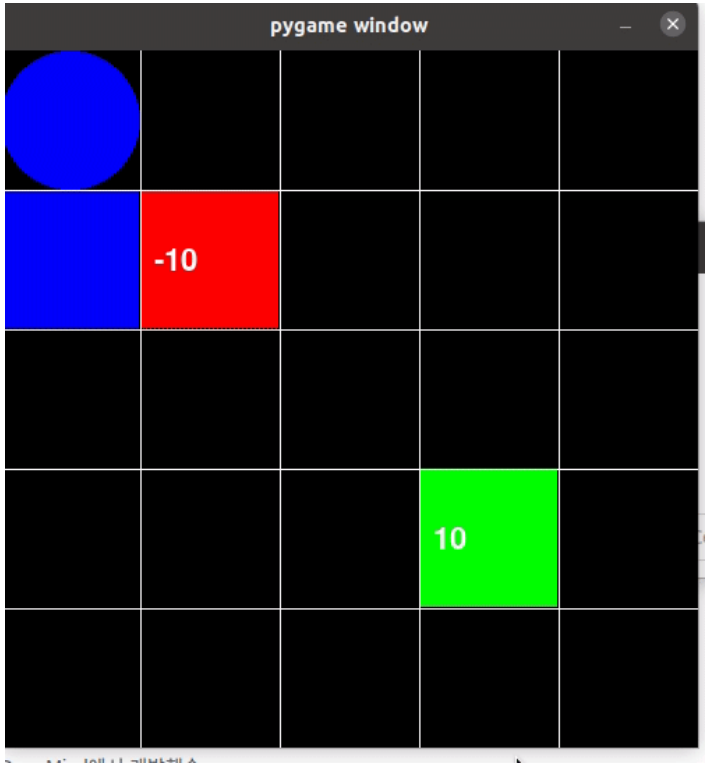
Implementation

- Qlearner.py
  - Q-table 생성
- Learn
  - 정해진 에피소드 횟수만큼 반복시행
  - EPSILON(30%) 확률로 무작위 행동결정
  - Q-value 계산
- evalQFunction
  - 현재 상태:행동의 max Q-value 계산
  - 다음 상태의 max Q-value 계산
  - Q-table 업데이트

```
class QLearner(object):
    qTable = {} # Maps cell to possible actions. Actions then map to reward
    discount = 0.9
    alpha = 0.9
    currState = (0,0)
    def learn(self):
        count = 0
        for episode in range(NUM_EPISODES):
            self.currState = (0, 0)
            count+=1
            # print(count)
            while not self.board.isTerminalCell(self.currState):
                # action value includs f
                action = self.epsilonGreedy(self.currState)
                self.evalQFunction(self.currState, action)
                self.currState = self.board.getCellAfterAction(self.currState, action)

        return self.qTable

    def epsilonGreedy(self, state):
        randInt = random.randint(1,11)
        if randInt <= EPSILON:
            #filtered avaliable action and return one action that randomly choiced
            validActions = list(filter(lambda action: self.board.isValidCell(state, action), ACTIONS))
            return random.choice(validActions)

        else:
            # Gets all qValues for specified state for all q values
            arr = {key: val for key, val in self.qTable.items() if key == state}
            # returns action that yields highest q value
            return max(arr[state], key=arr[state].get)


    def epsilon(self):
        return random.choice(ACTIONS)

    # Q(s,a)+=α·[r+γ·maxαQ(s′)−Q(s,a)]
    def evalQFunction(self,coord, action):
        nextCell = self.board.getCellAfterAction(coord, action)
        reward = self.board.getCellValue(nextCell)
        #Select next cell's max Q value in Qtable
        maxQSPrime = max([self.qTable[nextCell][action2] for action2 in ACTIONS if self.board.isValidCell(nextCell, action2)])
        #Calculate current state Q value = previous vaule
        self.qTable[coord][action] += (self.alpha * (reward + self.discount * maxQSPrime - self.qTable[coord][action]))
```

Run

- Case
  - EPISODE: 30
  - EPSILON: 30(%)
  - Terminal : 1
  - Trap : 1



Implementation

- Deep Q-learning 구현
  - Run_DQN.py
  - DQNlearner.py: 모델 훈련
  - replaybuffer.py: 학습데이터 저장, batch 단위 가공

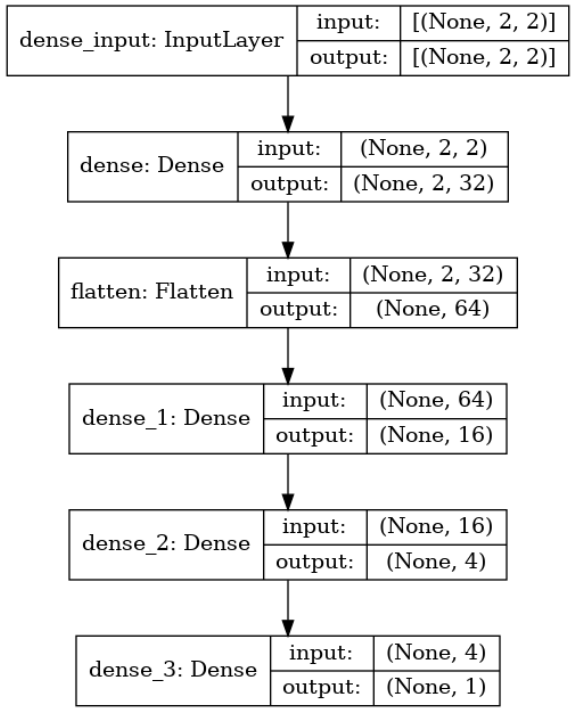
Implementation

- runDQN.py
  - Grid, 학습모델 객체 로드
  - Episode 횟수만큼 학습모델 훈련
  - 가중치 저장
  - 가중치 로드
  - 학습모델 추론 및 시행

```
p = Player()
b = Board()
b.initCellRewards()
b.createPenaltyCells()
w.drawSurface(b, p)

prev_action = "stay"
q = DQN_learner(b)
q.train(NUM_EPISODES)

print(b.getCellMap())
print(type(b.getCellMap()))
q.model.load_weights(".maze_solve_dqn.h5")
currNode = (p.getCurrCoords())
time.sleep(2)

def move(state, prev_action):
    input_state = np.reshape(tf.convert_to_tensor(state,dtype=tf.float32),(-1,2))
    prediction_act = tf.argmax(q.model.predict(input_state)).numpy()
    action = ACTIONS[prediction_act]
    if b.isValidCell(state, ACTIONS[prediction_act]):
        print("pridiction  value argmax is...",prediction_act)
        p.move(action)
        prev_action = action
    else:
        print("invalid action... stay")
        p.move(prev_action)


while(not b.isTerminalCell(currNode)):
    time.sleep(1)
    move(currNode,prev_action)

    currNode = p.getCurrCoords()
    print(currNode)
    w.colorCell(currNode, (0, 0, 255))
    pygame.display.update()
```

Implementation

- DQNlearner.py
- 모델 정의
  - 입력데이터 [좌표, 행동] (2차원 Tensor)
  - 출력값: Q-value (1차원 Tensor)
  - 가중치 역전파 옵티마이저: Adam
  - 학습률: 0.001
  - Loss 함수: Mean Squared Error
  - 활성화 함수: tanh(range: 0~1)
  - Layer 구성: 완전연결 신경망




```
def build_model(self):
    model = Sequential()
    model.add(Dense(32, input_shape=(2,2), activation='tanh'))
    model.add(Flatten())
    model.add(Dense(16, activation='tanh'))
    model.add(Dense(4, activation='tanh'))
    model.add(Dense(1, activation='linear'))
    model.compile(loss="mse", optimizer=tf.keras.optimizers.Adam(lr=self.DQN_LEARNING_RATE))
    return model
```

Implementation

- DQNlearner.py
  - Choose_best_action: 현재상태에서 가장 Q-value를 높게 예측한 행동 출력
  - epsilonGreedy: EPSILON 확률에 따라 행동 랜덤 선택

```
def choose_best_action(self, state, str_movables):
    best_actions = []
    max_act_value = -10
    movables_val = self.get_movables(state,str_movables)
    for a in movables_val:
        np_action = np.array([[state, a]])
        act_value = self.model.predict(np_action)
        if act_value > max_act_value:
            best_actions = [a,]
            which_act = np.array(state) - np.array(a)
            str_act = self.get_str_act(which_act)
            max_act_value = act_value
        elif act_value == max_act_value:
            best_actions.append(a)
            which_act = np.array(state) - np.array(a)
            str_act = self.get_str_act(which_act)
    return random.choice(best_actions), str_act
```

```
def epsilonGreedy(self, state):
    randInt = random.randint(1,11)
    validActions = list(filter(lambda action: self.board.isValidCell(state, action), ACTIONS))
    if randInt <= EPSILON:

        rnd_action = random.choice(self.get_movables(state,validActions))
        return rnd_action, random.choice(validActions)
    else: #just return max action value that regardless of valid cell
        action, str_action = self.choose_best_action(state, validActions)
        return action, str_action
```

Implementation

- replaybuffer.py
  - 훈련데이터 저장

```python
class ReplayBuffer(object):
    """
    Reply Buffer
    """
    def __init__(self, buffer_size):
        self.buffer_size = buffer_size
        self.buffer = deque()
        self.count = 0

    ## save to buffer
    def add_buffer(self, state, action, reward, next_state, done):
        transition = (state, action, reward, next_state, done)

        # check if buffer is full
        if self.count < self.buffer_size:
            self.buffer.append(transition)
            self.count += 1
        else:
            self.buffer.popleft()
            self.buffer.append(transition)

    ## sample a batch
    def sample_batch(self, batch_size):
        if self.count < batch_size:
            batch = random.sample(self.buffer, self.count)
        else:
            batch = random.sample(self.buffer, batch_size)
        # return a batch of transitions
        states = np.asarray([i[0] for i in batch])
        actions = np.asarray([i[1] for i in batch])
        rewards = np.asarray([i[2] for i in batch])
        next_states = np.asarray([i[3] for i in batch])
        dones = np.asarray([i[4] for i in batch])
        return states, actions, rewards, next_states, dones
```

Implementation

- Train
  - EPISODE 횟수만큼 훈련 반복
  - 한번의 에피소드가 종료될 때 까지epsilonGreedy()로 선택된 행동으로 시행
  - 행동 선택 시 **(현재상태, 행동, 보상, 다음상태, 행동)** 데이터를 버퍼에 저장 (훈련데이터 생성목적)

```
def train(self, max_episode_num):
        # initial transfer model weights to target model network
        count = 0
        times =  500
        for ep in range(int(max_episode_num)):
            count+=1
            time, episode_reward, is_terminal = 0, 0, False
            state = self.currState
            state = (0,0)

            for time in range(times):
                if self.board.isTerminalCell(state) or time ==(times-1):
                    break
                print("TERMINAL?",self.board.isTerminalCell(state))
                action, str_action = self.epsilonGreedy(state)# output: string
                next_state = state

                next_state = self.board.getCellAfterAction(state, str_action)# output: coordinate
                reward = tf.constant(self.board.getCellValue(state), dtype=tf.float32)
                is_terminal = self.board.isTerminalCell(next_state)# boolean
                print("is_termianae",is_terminal)
                next_movables = list(filter(lambda action: self.board.isValidCell(next_state, action), ACTIONS))
                self.buffer.add_buffer(state, action, reward, next_state,self.get_movables(state,next_movables), is_terminal)
                X = []
                Y = []
```

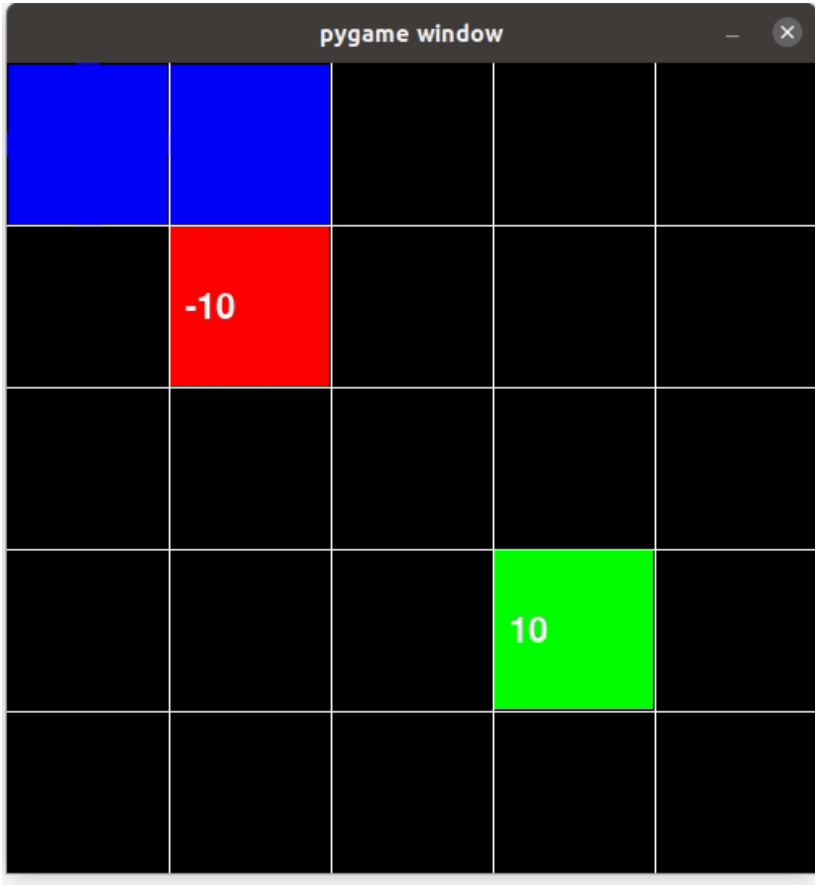
Implementation

- Train
  - X[] = 훈련데이터
  - Y[] = 정답데이터
  - 버퍼크기가 512 이상일 경우 훈련시작
  - 버퍼로부터 BATCH_SIZE(32) 만큼 로드
  - 버퍼를 순회하며 다음상태의 Q-value를 예측
  - 예측된 Q-val 중 $\max_{a'} Q(s', a'))$ 에 감가율을 곱해
  - 정답데이터(Y)에 저장
  - X에 버퍼로부터 불러온 (상태, 행동) 리스트 저장
  - X, Y를 사용하여 모델 훈련 시작

```
    X = []
    Y = []
    if self.buffer.buffer_count() > 512:  # start train after buffer has some amounts
        if self.EPSILON > self.EPSILON_MIN:
            self.EPSILON *= self.EPSILON_DECAY

        states, action, rewards, next_states, next_movables, is_terminals = self.buffer.sample_batch(self.BATCH_SIZE)
        #print("isterminals",is_terminals)
        #print("isterminals",is_terminals.shape)
        for i in range(self.BATCH_SIZE):
            input_action = [states[i], np.array(action[i])]

            if(is_terminals[i]):
                target_f = rewards[i]
            else:
                next_rewards = []
                for next_action in next_movables[i]:
                    np_next_s_a = np.array([[next_states[i],next_action]])
                  # print("dsdsdsdsdsdsds",np_next_s_a.shape)
                    next_rewards.append(self.model.predict(np_next_s_a))
                np_next_reward_max = np.amax(np.array(next_rewards))
                target_f = rewards[i] + self.GAMMA * np_next_reward_max
            X.append(input_action)
            Y.append(target_f)
        np_X = np.array(X)
        np_Y = np.array([Y]).T
        if self.EPSILON > self.EPSILON_MIN:
            self.EPSILON *= self.EPSILON_DECAY
        self.dqn_learn(np_X,np_Y)
        #self.update_target_network(self.TAU)
    # update current state
    state = next_state
    print("State....",state)
    print("Reward",reward)
    episode_reward += reward
    time += 1
## display rewards every episode
print('Episode: ', ep+1, 'Time: ', time, 'Reward: ', episode_reward)
self.save_epi_reward.append(episode_reward)
```

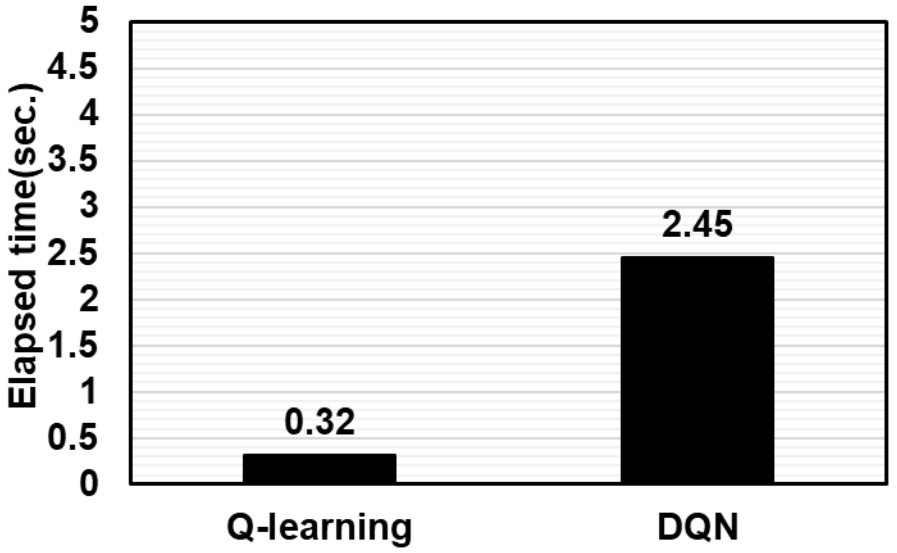
Run

- Case
  - EPISODE: 30
  - EPSILON: 30(%)
  - Terminal : 1
  - Trap : 1
  - BATCH_SIZE: 32
  - Sleep: 0.5



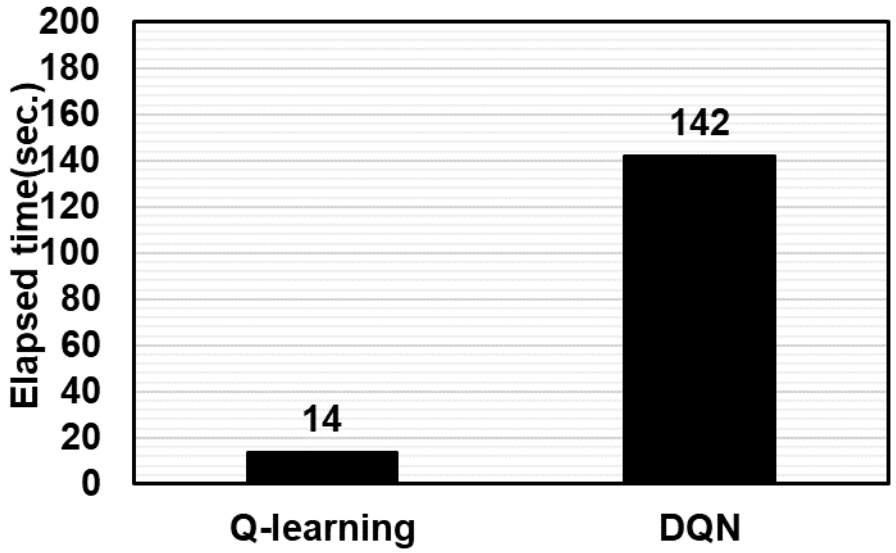
Evaluation

- 실행시간비교
  - Q-learning Deep Q-learning의 시행시간 비교측정
  - 이전 case와 동일 시행, 미리 작성된 Q-table, 훈련된 Model 사용
  - 각 3번시행 후 평균값 비교
  - DQN이 6.6배 더 오래 걸림
  - 행동 결정 시 학습모델 추론시간 > Q-table 참조시간



Evaluation

- 훈련시간비교
  - Q-learning Deep Q-learning의 훈련시간 비교
  - Q-table 생성시간, 학습모델 훈련시간 비교
  - 각 3번시행 후 평균값 비교
  - 9배 차이



향후 연구적용 방안

- 지금까지 강화학습은 현실문제를 해결하기 위해 정확도, 성능개선 위주로만 연구되어 옴
- 모델 또는 시스템의 규모가 커짐에 따라 발생하는 시스템 리소스 낭비, 성능저하에 대해서는 상대적으로 관심이 작음
- 따라서 컴퓨터 시스템 관점에서 최신 강화학습 모델들의 리소스 사용량을 분석하여 동일한 환경에서 더 높은 처리성능을 달성하기 위한 연구를 진행하고자 함

Impression

- 전공과정에서는 알 수 없는 대형선박의 건조과정에 대해 배울 수 있어 흥미로웠음
- 스마트야드 과제에서 진행중인 부분이 전체 제조과정에서 어느 단계에 해당하는지 알 수 있는 기회가 됨
- 강의가 전체적인 기틀을 잡아주어 추가적인 개선사항에 대해 고려할 요소를 생각할 수 있게 되었음