2020270811 이승수

강화학습을 사용한 주식투자

목차

- 개요
- Class diagram
- 모델 설명
- 학습 결과 설명

개요 – 문제 정의

1. 주식시장에서 하루 동안 한 주식을 사고 팔고를 반복할 때, Agent 가 수익을 낼 수 있는 선택을 하는 강화 학습

2. DQN 알고리즘 사용

개요 – 전처리 과정

Data source : (대신증권creon API)

```
"""
open / last close
high / close
low / close
close / last close
volume / last volume
close / MA5close
close / MA10 close
close / MA20 close
close / MA60 close
close / MA120 close
volume / MA5 volume
volume / MA10 volume
volume / MA20 volume
volume / MA60 volume
volume / MA120 volume
acc_buy / acc_sell * 100 (1min)
MA3 acc_ratio (3min)
"""
```

개요 – state 정의

- 최근 6분간 실시간 정보 (6, 19) 형태
- 19 컬럼 (실시간 정보 + hold여부 + 실시간 수익)

Agent class

```
def get_state(self, observation, done, reward):
    if done == -1 or done == 1: # hold after sell
        self.holds.pop(0)
        self.holds.append(0)
    elif done == 0: # hold after buy
        self.holds.pop(0)
        self.holds.append(1)
    # 실제 reward X - > 현재 가치
    self.rewards.pop(0)
    self.rewards.append(reward)

    #time = observation[3]
    data = observation[5]
    # print(observation[1], observation[2])
    state = [[datas, [holds], [rewards]] for datas, holds, rewards in zip(data, self.holds, self.rewards)]
    state = list(chain.from_iterable(state))
    state = list(chain.from_iterable(state))
    state = np.reshape(state, [6, self.state_size])
    # np.array [list([1], hold, reward) list([2]) list([3])
    # list([4]) list([5]) list([6]) ]
    return state
```

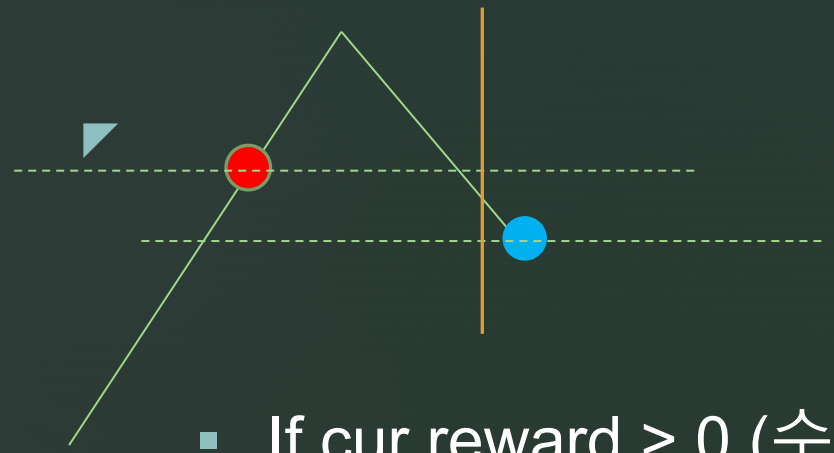
Agent class

```
def get_action_list(self, action, buy_hold_cnt, sell_hold_cnt):
    if action == 2:
        if self.action_point == 0:
            if self.over_3_or_not(buy_hold_cnt):
                return [1, 2, 4]
            else:
                return [1, 2]
        elif self.action_point == 1 or self.action_point == -1:
            if self.over_3_or_not(sell_hold_cnt):
                return [0, 2, 3]
            else:
                return [0, 2]
    elif action == 0:
        self.action_point = 0
        return [1, 2]
    elif action == 1:
        self.action_point = 1
        return [0, 2]
    elif action == 3:
        self.action_point = 0
        return [1, 2]
    elif action == 4:
        self.action_point = 1
        return [0, 2]


# 0 buy -> 1 sell / 2 hold
# 1 sell -> 0 buy / 2 hold
# 2 hold -> 0 buy / 1 sell
# 3 con_buy
# 4 con_sell

# action point
# action_point == 1 -> sell ~
# action_point == 0 -> buy ~
```

개요 – action 정의

- Action rule : buy – hold – sell (set)

- 공격적 매수 (hold after sell < 4)
- 보수적 매수 (hold after sell >=4)
- 공격적 매도 (hold after buy < 4)
- 보수적 매도 (hold after buy >= 4)
- 아무것도 안함



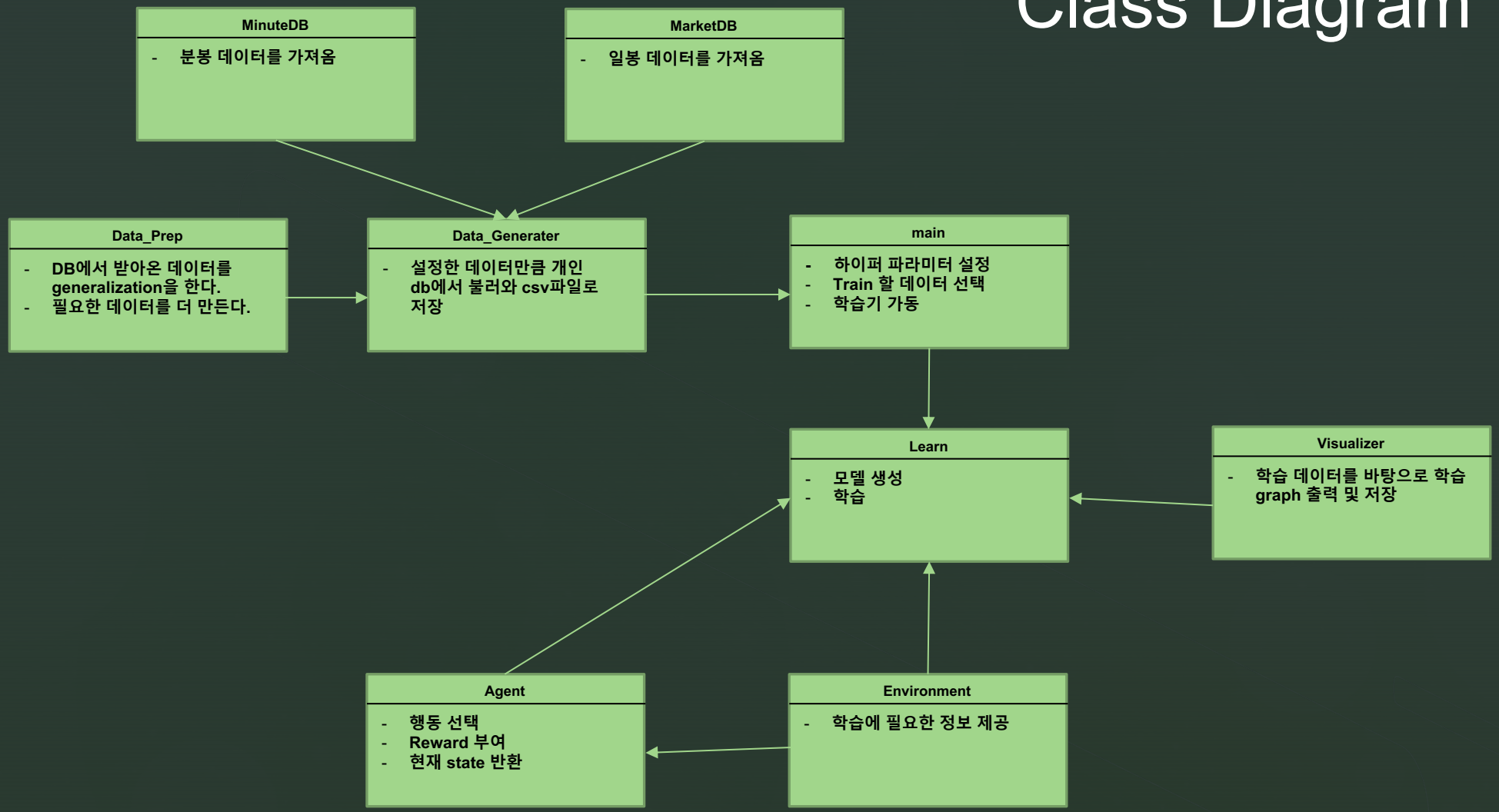
개요 – reward 정의

- If cur reward > 0 (수익 범위 안)
  - If Cur reward > fin reward (현재 가격 > 매도 가격)
    - Fin reward – cur reward (매도 가격 – 현재 가격) [음수=penalty]
  - If Cur reward < fin reward (매도 가격 > 현재 가격)
    - Cur reward (현재 가격) [양수=reward]
- Else if cur reward < 0 (손실 범위 안)
  - Cur reward (현재 가격) [음수=penalty]

Learn class

```
def append_sample(self, action_p, sample_list):
    if action_p == 1: # <buy - hold - sell> sell ~
        # agent 정보를 reset한다.
        self.agent.reset_agent()
        # reward 수정
        #print(sample_list)
        sample_list = np.array(sample_list, dtype="object") # dtype="object"
        #print(sample_list)
        states = sample_list[:,0]
        actions = sample_list[:,1]
        rewards = sample_list[:,2]
        next_states = sample_list[:,3]
        length_list = len(sample_list)
        #print(states, actions, rewards, next_states)

        fin_reward = rewards[length_list - 1]
        for i in range(length_list):
            cur_reward = rewards[i]
            if cur_reward > 0:
                if cur_reward > fin_reward:
                    ret_reward = fin_reward - cur_reward
                elif cur_reward <= fin_reward:
                    ret_reward = cur_reward
            elif cur_reward <= 0:
                ret_reward = cur_reward
            # 수정된 sample memory에 append
            self.memory.append((states[i], actions[i], ret_reward, next_states[i]))
        return True

    else:
        return False
```

Class Diagram



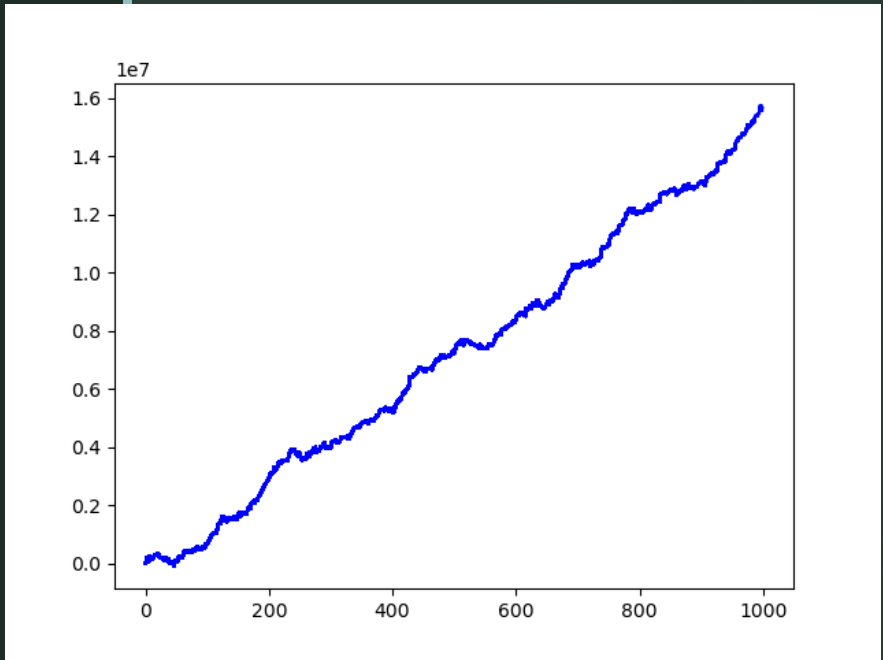
모델 설명(기본)

Learn class

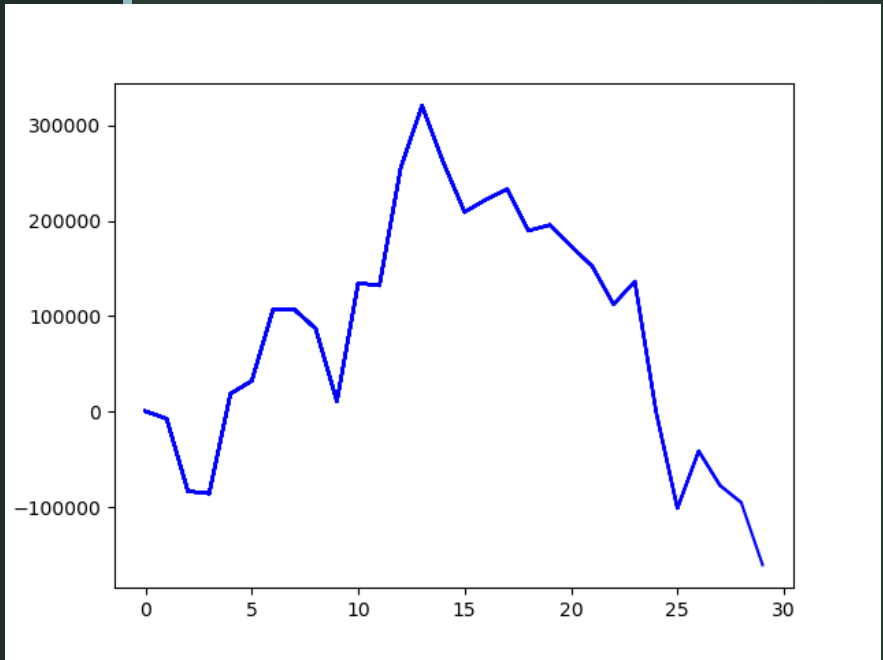
```
def build_model(self):
    model = Sequential()
    # np.array [list([1], hold reward) list([2]) list([3]) list([4]) list([5]) list([6]) ]
    model.add(Dense(100, input_shape=(6, 19), activation='relu',
                    kernel_initializer='he_uniform'))
    model.add(Dense(100, activation='relu',
                    kernel_initializer='he_uniform'))
    model.add(Dense(100, activation='relu',
                    kernel_initializer='he_uniform'))
    model.add(Dense(self.action_size, activation='linear',
                    kernel_initializer='he_uniform'))
    model.summary()
    model.compile(loss='mse', optimizer=Adam(lr=self.learning_rate))
    return model
```

5 layer DNN
Input layer : (6, 19) shape
Hidden layer (3) : 100개의 뉴런
Output layer : action_size 크기

학습결과(1)



Train graph

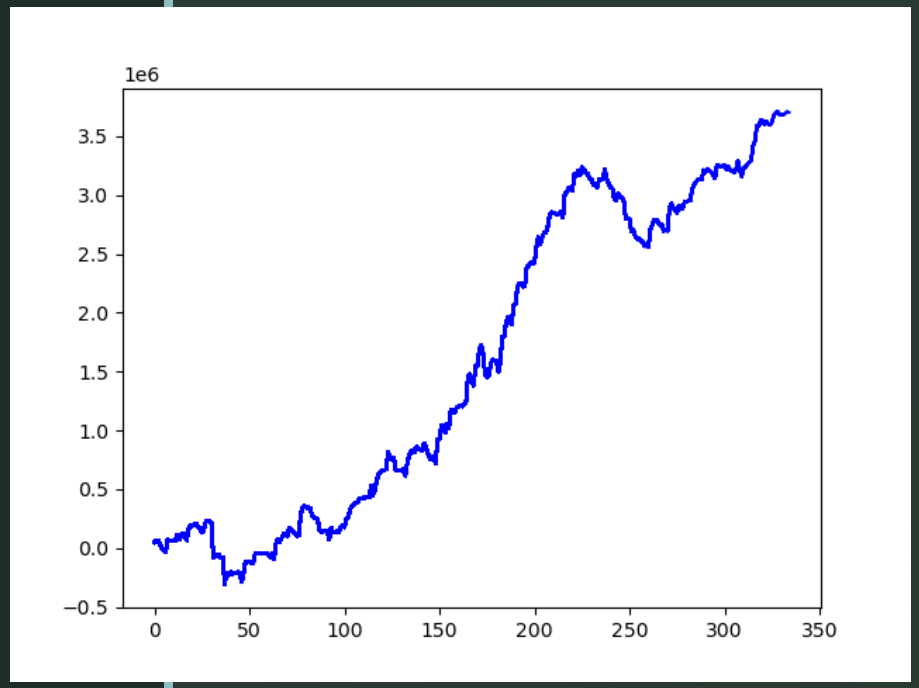


Test graph

```
<약 1000종목 학습>
trade money(한번 거래 할때 금액) : 1,000,000원
discounting_factor : 0.99
learning_rate : 0.001
Epsilon : 1.0
epsilon_decay : 0.999
epsilon_min : 0.01
batch_size : 64
train_start : 100,000
max_memory : 200,000
```

Weight경로 : train_models/[result1]stock_dqn.h5

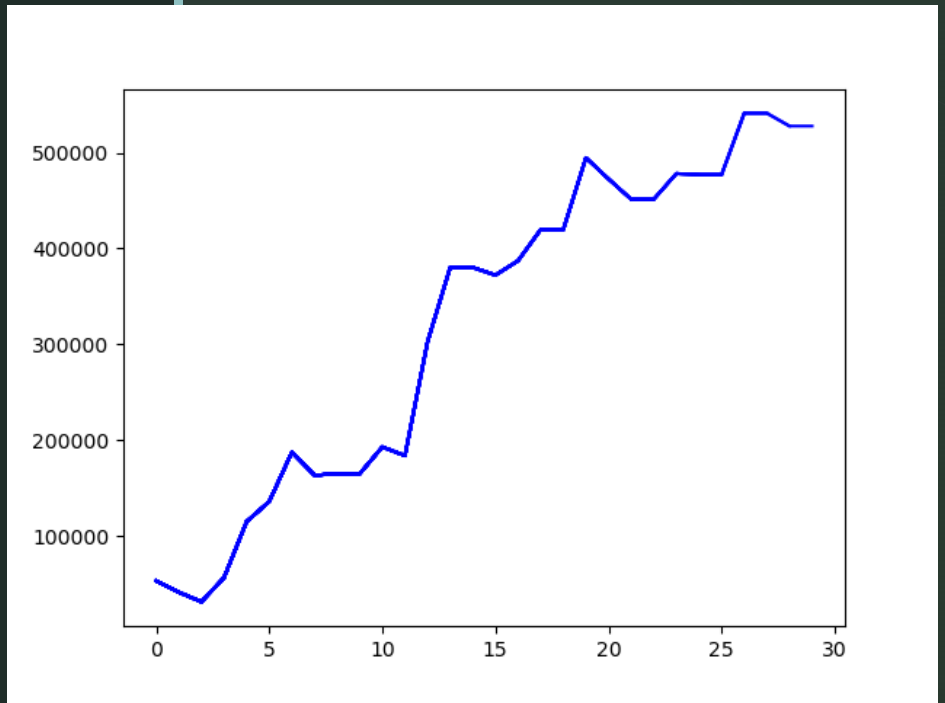
```
def build_model(self):
    model = Sequential()
    # np.array [list([1], hold reward) list([2]) list([3]) list([4]) list([5]) list([6]) ]
    model.add(Dense(100, input_shape=(6, 19), activation='relu',
                    kernel_initializer='he_uniform'))
    model.add(Dense(100, activation='relu',
                    kernel_initializer='he_uniform'))
    model.add(Dense(100, activation='relu',
                    kernel_initializer='he_uniform'))
    model.add(Dense(self.action_size, activation='linear',
                    kernel_initializer='he_uniform'))
    model.summary()
    model.compile(loss='mse', optimizer=Adam(lr=self.learning_rate))
    return model
```



Train graph

학습결과(2)

<약 1000종목 학습>
Epoch : 330 (330종목으로 제한 overfitting 방지)
trade money(한번 거래 할때 금액) : 1,000,000원
discounting_factor : 0.99
learning_rate : 0.001
Epsilon : 1.0
epsilon_decay : 0.999
epsilon_min : 0.01
batch_size : 2,000
train_start : 50,000
max_memory : 100,000

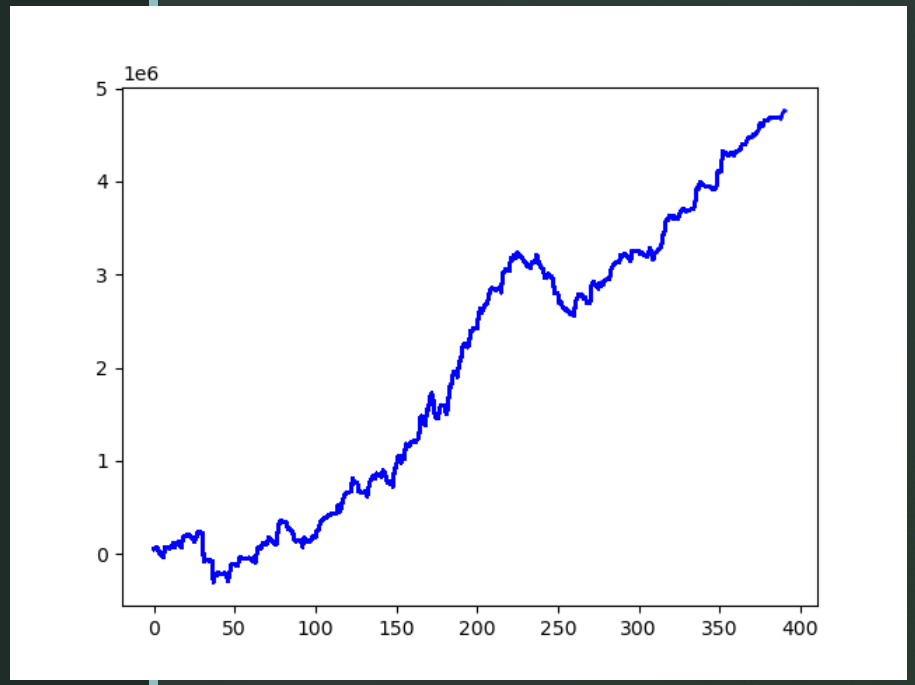


Test graph

Weight경로 : train_models/[result2]stock_dqn.h5

```
def build_model():
    model = Sequential()
    model.add(Dense(200, input_shape=(6, 19), activation='relu',
                    kernel_initializer='he_uniform'))
    model.add(Dense(200, activation='relu',
                    kernel_initializer='he_uniform'))
    model.add(Dense(200, activation='relu',
                    kernel_initializer='he_uniform'))
    model.add(Dense(action_size, activation='linear',
                    kernel_initializer='he_uniform'))
    model.summary()
    model.compile(loss='mse', optimizer=Adam(lr=learning_rate))
    return model
```

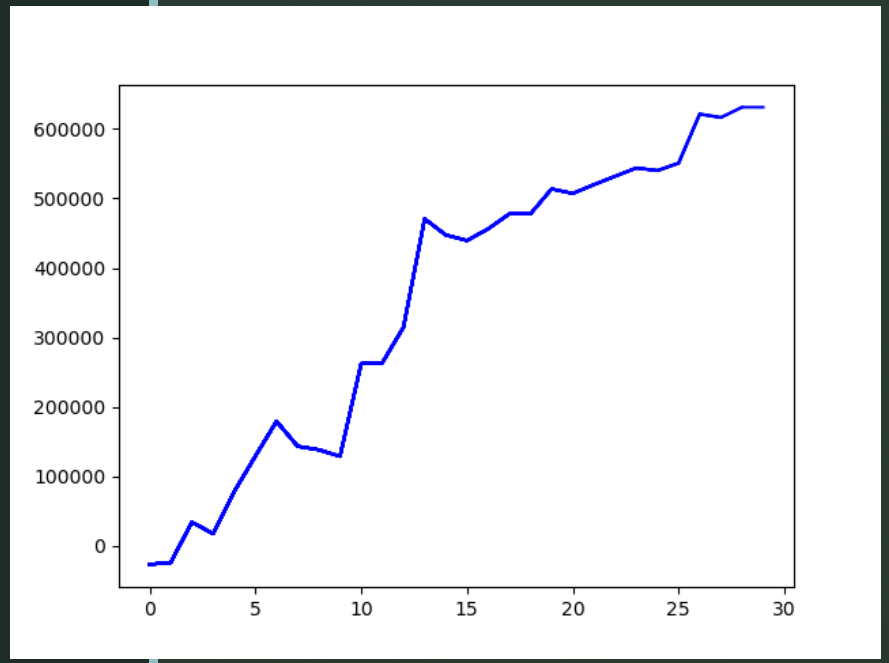
Train graph



학습결과(3)

<약 1000종목 학습>
Epoch 400: (400종목으로 제한 overfitting 방지)
trade money(한번 거래 할때 금액) : 1,000,000원
discounting_factor : 0.99
learning_rate : 0.001
Epsilon : 1.0
epsilon_decay : 0.999
epsilon_min : 0.01
batch_size : 2,000
train_start : 50,000
max_memory : 100,000

Test graph

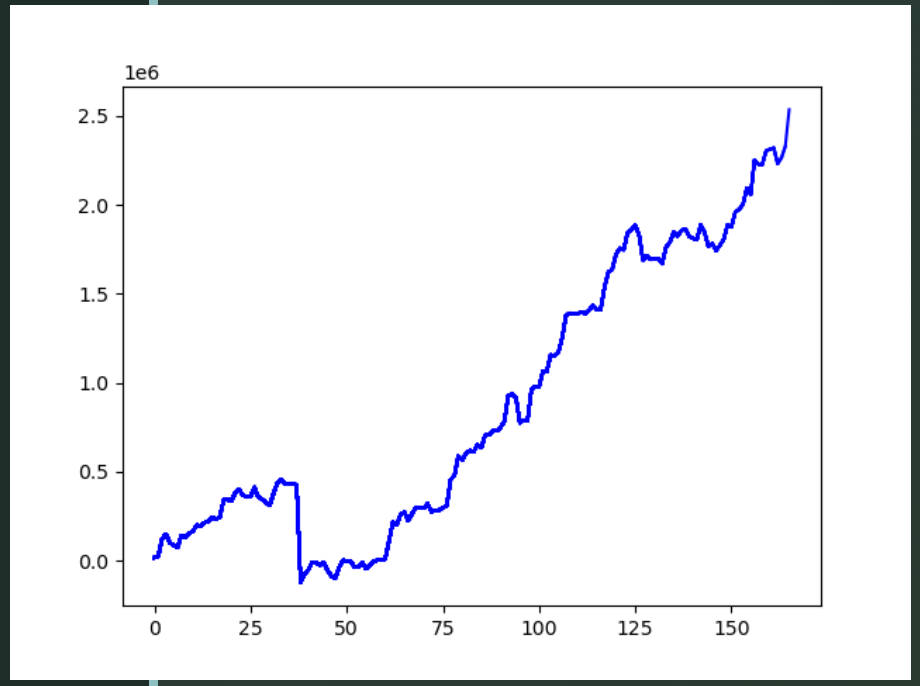


Weight경로 : train_models/[result3]stock_dqn.h5

```
def build_model():
    model = Sequential()
    model.add(Dense(200, input_shape=(6, 19), activation='relu',
                    kernel_initializer='he_uniform'))
    model.add(Dense(200, activation='relu',
                    kernel_initializer='he_uniform'))
    model.add(Dense(200, activation='relu',
                    kernel_initializer='he_uniform'))
    model.add(Dense(action_size, activation='linear',
                    kernel_initializer='he_uniform'))
    model.summary()
    model.compile(loss='mse', optimizer=Adam(lr=learning_rate))
    return model
```

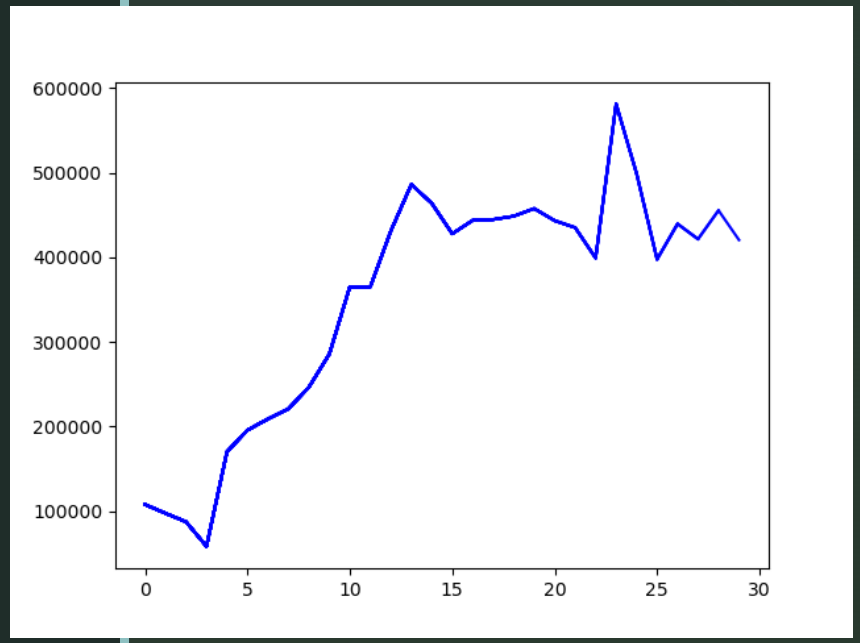
학습결과(4)

Train graph




```
<약 1000종목 학습>
Epoch : 160 (160종목으로 제한 overfitting 방지)
trade money(한번 거래 할때 금액) : 1,000,000원
discounting_factor : 0.99
learning_rate : 0.001
Epsilon : 1.0
epsilon_decay : 0.999
epsilon_min : 0.01
batch_size : 5,000
train_start : 20,000
max_memory : 40,000
```

Test graph

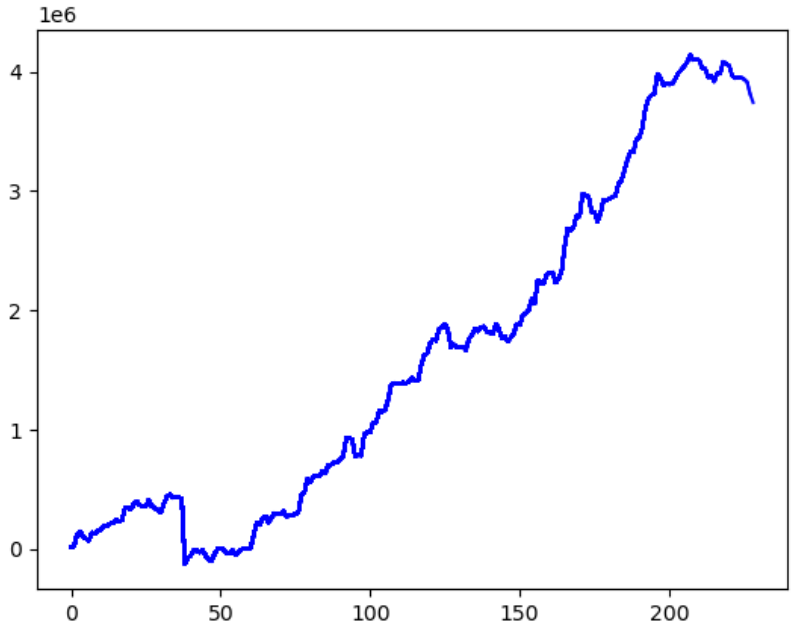


Weight경로 : train_models/[result4]stock_dqn.h5

```
def build_model():
    model = Sequential()
    model.add(Dense(200, input_shape=(6, 19), activation='relu',
                    kernel_initializer='he_uniform'))
    model.add(Dense(200, activation='relu',
                    kernel_initializer='he_uniform'))
    model.add(Dense(200, activation='relu',
                    kernel_initializer='he_uniform'))
    model.add(Dense(action_size, activation='linear',
                    kernel_initializer='he_uniform'))
    model.summary()
    model.compile(loss='mse', optimizer=Adam(lr=learning_rate))
    return model
```

학습결과(5)

Train graph



Test graph


```
<약 1000종목 학습>
Epoch 230: (230종목으로 제한 overfitting 방지)
trade money(한번 거래 할때 금액) : 1,000,000원
discounting_factor : 0.99
learning_rate : 0.001
Epsilon : 1.0
epsilon_decay : 0.999
epsilon_min : 0.01
batch_size : 5,000
train_start : 20,000
max_memory : 40,000
```

Weight경로 : train_models/[result5]stock_dqn.h5

```python
def build_model():
    model = Sequential()
    model.add(Dense(200, input_shape=(6, 19), activation='relu',
                    kernel_initializer='he_uniform'))
    model.add(Dense(200, activation='relu',
                    kernel_initializer='he_uniform'))
    model.add(Dense(200, activation='relu',
                    kernel_initializer='he_uniform'))
    model.add(Dense(action_size, activation='linear',
                    kernel_initializer='he_uniform'))
    model.summary()
    model.compile(loss='mse', optimizer=Adam(lr=learning_rate))
    return model
```

고찰

- 하이퍼 파라미터
  - Batch_size, train_start, neurons, hidden layers
- Action 설정과 reward 설정

감사합니다.

질문 있으시면 메일로 연락주십시오