

2023 전기 중간보고서

심층 강화학습을 사용한 주식투자 전략 개발



제출일	2023. 08. 01	전공	정보컴퓨터공학부
팀명	파라솔	담당교수	유영환 교수님
학번	201824586	이름	정희영
	201824469		박동진
	201824514		신재환

목차

1. 과제 목표 및 요구 조건
 - 1-1. 과제 목표
 - 1-2. 요구 조건
2. 과제 설계
 - 2-1. 서비스 흐름도
 - 2-2. 데이터 수집
 - 2-2-1. 수집할 종목
 - 2-2-2. 수집할 데이터
 - 2-3. 강화학습
 - 2-3-1. 강화학습 환경
 - 2-3-2. 강화학습 에이전트
 - 2-3-3. 강화학습 알고리즘
 - 2-4. 서비스
 - 2-4-1. Front-end
 - 2-4-2. Back-end
3. 구성원별 진척도 및 과제 추진 계획
 - 3-1. 구성원별 과제 진척도
 - 3-2. 과제 추진 계획
4. 과제 수행 내용 및 중간 결과
 - 4-1. 데이터 수집
 - 4-2. 강화학습
 - 4-3. 환경 개발
 - 4-4. 서비스 개발

1. 과제 목표 및 요구 조건

1-1. 과제 목표

주식투자 강화학습 모델을 만들고, 사용자에게 강화학습 모델의 추천도를 제공한다.

1-2. 요구 조건

과제를 위해 달성해야 하는 요구 조건은 아래와 같다.

1. 주식투자에 대한 의견을 제시할 수 있는 강화학습 모델 개발

1. 강화학습을 진행할 환경 개발
2. 강화학습을 수행하는 Agent 선택
3. 적절한 Hyperparameter 와 Feature 를 통한 모델 개발
4. 학습을 진행하기 위한 데이터 수집

2. 사용자에게 서비스를 제공하기 위한 Front-end 구현

3. Front-end의 요청에 대한 응답을 제공하고 모델의 예측결과와 학습데이터를 저장할 Back-end 개발

2. 과제 설계

서비스 흐름도

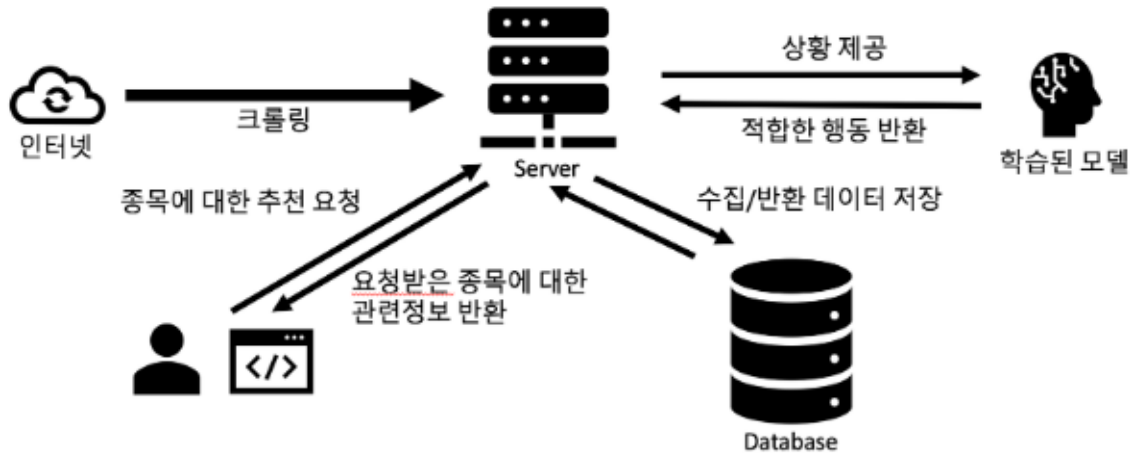
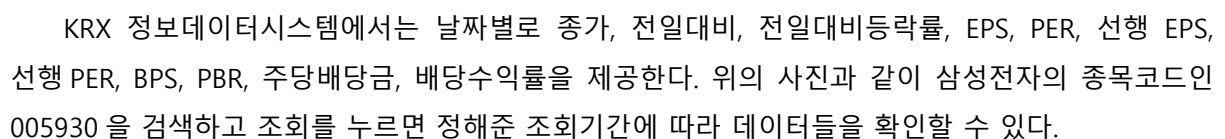
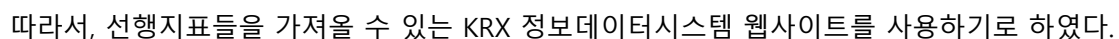


그림 1. 서비스 흐름도

완성된 과제의 형태는 위와 같다.

1. 업데이트 시각이 되면 서버에서 크롤링을 통해 학습에 필요한 데이터를 수집/저장한다.
2. 획득한 데이터를 학습된 모델에 제공한다.
3. 학습된 모델은 제공된 데이터를 기반으로 투자 의견을 제시한다.
4. 제시된 투자 의견을 데이터베이스에 저장한다.
5. 사용자로부터 종목에 대한 추천요청을 받으면, 학습된 모델로부터 도출된 주식추천과, 관련 데이터를 사용자에게 제공한다.

기존의 착수보고서에서는 키움증권에서 제공하는 OpenAPI를 사용하고자 하였다. 하지만 키움증권의 OpenAPI는 다음과 같이 우리가 AI에게 학습시킬 때 필요한 과거의 PER, PBR, ROE 등의 데이터를 가져올 수가 없었다.



2-1-1. 수집할 종목

먼저, 수집할 종목을 정하는 데 있어서 가장 중요한 것은 '학습시킬 수 있는 데이터가 충분한가?'라고 판단하였다. 학습시킬 수 있는 데이터의 양이 많을수록 정확도도 높아지기 때문에 선행 데이터들이 많으며, 작전에 의해 주가의 변동성이 크지 않은 시가 총액이 높은 종목을 선정해야 했다. 또한, 주식 종목의 경우 각 섹터에 따른 주가 변동성도 크기 때문에 학습시킬 때에 참고할 수 있는 ETF가 있는 종목이고, 우리가 결과를 도출하기 위해 사용할 5가지의 종목의 섹터들이 최대한 겹치지 않도록 선정하였다. 5가지 종목은 삼성전자, SK 하이닉스, 현대건설, POSCO 홀딩스, 엔씨소프트이다. 삼성전자의 경우에는 'KODEX 삼성그룹', SK 하이닉스는 'KODEX 반도체', 현대건설은 'KODEX 건설', POSCO 홀딩스는 'KODEX 철강', 엔씨소프트는 'KBSTAR 게임테마'의 ETF를 참고하여 학습시키고자 한다.

2-1-2. 수집할 데이터

• 주당순이익(Earning Per Share, EPS)

EPS는 기업의 순이익을 발행한 주식 수로 나눈 값을 의미한다. PER과 함께 가치투자의 대표적인 지표이다. EPS가 주가보다 낮다고 가정하면, EPS 값이 높아질수록 주가에 수렴하기 때문에 고평가되었던 종목이 저평가 단계로 다가가기 때문에 투자가치가 높아 가치투자자 입장에서는 매력적으로 볼 수 있다. EPS가 높다는 것은 그만큼 경영실적이 양호하다는 뜻이고, 배당 여력도 많으므로 주가에 긍정적인 영향을 미친다.

$$\text{EPS (주당순이익)} = \frac{\text{주가}}{\text{PER (주가수익비율)}} = \frac{\text{주가}}{\text{발행주식수}}$$

그림 2. EPS 계산 공식

• 주가순자산비율(Price Book-value Ratio, PBR)

PBR은 자산 대비 주가의 비율이다. PBR이 낮을수록 회사의 가치 대비 주가가 낮다는 것이므로 해당 종목이 저평가돼 있을 가능성이 있다.

$$\text{PBR (주가순자산비율)} = \frac{\text{주가}}{\text{BPS (주당순자산)}} = \frac{\text{주가}}{\frac{\text{총자산-총부채}}{\text{발행주식수}}}$$

그림 3. PBR 계산 공식

• 주가수익비율(Price Earning Ratio, PER)

PER 은 주가와 회사의 순이익의 비율이다. PER 이 낮을수록 회사의 순이익에 비해 주가가 저평가돼 있다고 볼 수 있다. 그러나 업종마다 평균 PER 의 차이가 크기 때문에 PER 이 높다, 낮다라고 절대적으로 판단할 기준은 없다.

$$\text{PER (주가수익비율)} = \frac{\text{주가}}{\text{EPS (주당순이익)}} = \frac{\text{주가}}{\frac{\text{당기순이익}}{\text{주식수}}}$$

그림 4. PER 계산 공식

2-2. 강화학습

강화학습 알고리즘을 쉽게 구현하고 테스트하기 위한 대표적인 라이브러리로는 Stable-Baselines3 이 있고, 강화학습 환경을 쉽고 편리하게 구현하기 위한 라이브러리로는 OpenAI 에서 제공하는 OpenAI Gym 이 있다. Gym 라이브러리를 이용하여 강화학습 환경을 개발하면, Stable-Baselines3 에서 제공하는 학습 알고리즘을 사용하여 학습을 진행할 수 있다. 또한 제시된 Env 를 상속받아 환경을 개발하기 때문에, 환경 설계에 대한 시행착오를 줄일 수 있다. 이러한 이유들로 환경을 개발하는 데에는 Open AI Gym 을, 학습을 진행하는 데에는 Stable-Baselines3 를 사용하기로 했다.

2-2-1. 강화학습 환경

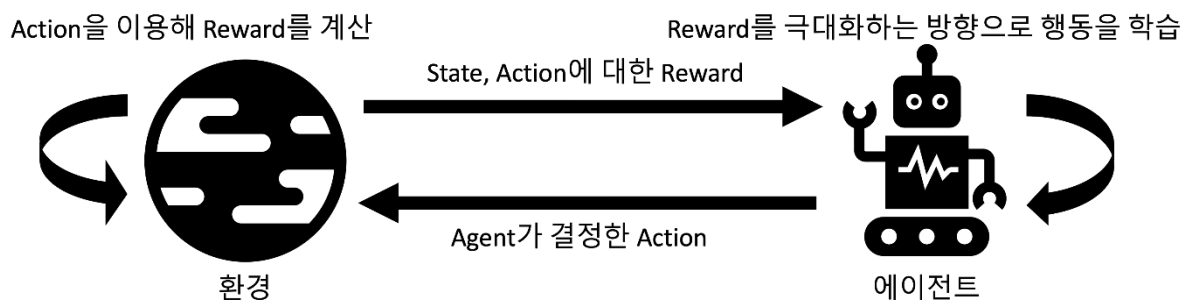


그림 5. 강화학습의 이해를 돕기 위한 이미지

강화학습에서 환경은 Agent 에게 현재 상태를 State 로서 제공하고, Agent 로부터 Action 을 입력받으면 해당 Action 을 처리하여 적절한 Reward 를 Agent 에게 반환한다. Agent 는 환경이 제공하는 State 와 Reward 를 이용해서 Action 을 결정하기 때문에, 환경은 강화학습의 결과에 큰 영향을 준다.

환경 설계

강화학습을 위해서는 Agent 의 행동에 따라 주식투자를 진행하고 그에 따른 손익을 reward 로 제공할 수 있도록 환경을 구성해야 한다. 전체적인 환경은 기본적인 주식투자 환경과 동일하게 설계하였다. 종목에 대한 데이터, 주변 요소들의 데이터를 보고 Action(매수, 매도, 관망)이 결정되면, 그에 따른 행동이 이루어지고, 다음 날의 주가변동에 따라서 포트폴리오 가치가 변화는 식이다. 모든 매매는 종가를 기준으로 이루어지게 하였다.

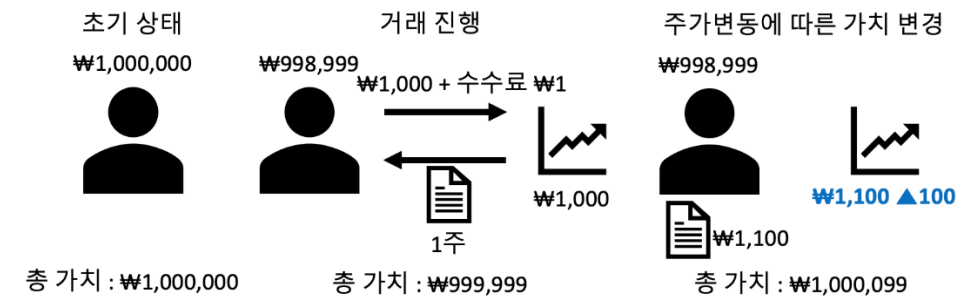


그림 6. 주식 투자 환경에 대한 이미지

Open AI Gym 을 이용하여 환경을 개발하려면, Agent 가 선택할 수 있는 Action 과 Agent 에 제공하는 State 의 형태를 정의하고, 학습 알고리즘에서 사용하는 2 개의 메서드를 필수적으로 개발해야 한다.

필수 Property

`self.action_space`

Agent 가 선택할 수 있는 Action 의 범위이다. 매수, 매도, 관망, 세 가지로 정의하였다.

`self.observation_space`

Agent 가 관측할 수 있는 환경의 상태이다. 최근의 주가데이터(`stock_info`), 현재 보유 비율(`holdings`), 평균단가(`avg_buy_price`)로 일반적으로 투자를 할 때, 투자자가 참고할 만한 내용들로 구성하였다.

`stock_info` 는 환경의 parameter 로 들어가는 `window_size` 만큼 최근 데이터를 제공한다. `window_size==10` 일 경우 최근 10 개 거래일간의 데이터를 제공하는 식이다.

필수 Method

`self.reset(self) => return ObsType`

학습을 시작하기 전에 환경을 초기화하는 메서드이다. 초기화한 환경에서 관측할 수 있는 state 를 반환한다. 날짜를 초기상태로 되돌리고 자본을 초기화한다.


```
self.step(self, ActType) => return ObsType, reward, isDone, info
```

Agent 가 action 을 도출하여 환경에 전달하면 해당 action 을 환경에 적용하고, 그에 따른 결과를 Agent 에게 전달하는 메서드이다.

보상

학습 agent 는 reward 를 기준으로 하여 학습을 진행하므로, reward 는 강화학습에서 가장 중요한 요소들 중 하나다. 주식투자전략을 학습시키기 위해서는 에이전트 포트폴리오 가치의 증감율을 reward 로서 주는 것이 가장 적합하다고 판단했다. 수익을 내면 +보상을 받고, 손실을 보면 -보상을 받는 식으로 구성하기로 했다.

보상지연

미세한 차트의 변화에 agent 가 민감하게 반응하는 것을 방지할 필요가 있다. 주식투자에서 너무 잦은 매매는 바람직하지 않으며, 너무 잦은 판단 반복은 사용자에게 혼란을 줄 수도 있기 때문이다. 이를 반영하기 위해 특정 수치(`reward_threshold`) 이상의 손익이 발생했을 경우에 reward 를 제공하도록 설계했다. Agent 의 투자로 포트폴리오 가치의 변화가 생기고, 가치변화가 특정 임계점을 넘으면 보상이 주어지는 식이다.

2-2-2. 강화학습 에이전트

강화학습 에이전트는 주어진 환경과 상호작용하면서 행동을 결정하는 의사 결정을 수행하는 주체라고 볼 수 있다. 에이전트는 현재 자신의 상태 및 주변 상태를 파악하고 정책(policy)을 통해 자신이 취할 최적의 행동을 선택한다.

2-2-3. 강화학습 알고리즘

우리의 프로젝트에서는 DQN(Deep Q-Network)과 A2C(Advantage Actor-Critic) 알고리즘을 사용한다.

DQN(Deep-Q-Network)

DQN 알고리즘은 기존의 Q-learning 과 딥러닝을 결합시킨 것이다. 기존의 Q-learning 은 state-action 에 해당하는 Q-value 를 Q-table 에 테이블 형식으로 저장하고 에이전트가 해당 state 에서 action 을 선택할 때 Q-table 을 참고해 Q-value 가 가장 큰 행동을 선택하는 것이지만 이는 state space 와 action space 가 커지게 되면 Q-table 에 저장할 값들도 많아지기 때문에 많은 메모리와 탐색 시간이 필요하게 된다. 따라서 딥러닝을 이용해서 Q-value 를 따로 저장하지 않고 Q-function 을 근사시켜 값을 구할 수 있다. DQN 알고리즘의 큰 특징으로는 experience replay 와 target network 를 꼽을 수 있다.

A2C(Advantage Actor-Critic)

A2C 알고리즘은 Actor-Critic 알고리즘을 기반으로 한다. Actor-Critic 알고리즘은 Actor network 와 Critic network 이렇게 2 개의 네트워크를 사용해서 학습을 진행하는데 Actor 는 정책(policy)을 학습하고 Critic 은 주어진 정책의 가치를 평가하는 역할을 한다. A2C 알고리즘은 Actor-Critic 알고리즘에서 Actor 의 기대 출력을 Advantage 를 활용해 계산하는 것입니다. Advantage 는 현재 상태-행동 쌍의 가치와 평균 가치를 비교해 얻어지는 것으로 양의 Advantage 는 좋은 행동을 의미하고, 음의 Advantage 는 나쁜 행동을 의미합니다. 이렇게 A2C 알고리즘은 Advantage 를 사용해서 Actor 가 더 높은 확률로 더 높은 Advantage 를 가지는 행동을 선택하게 한다.

2-3. 서비스

2-3-1. Front-end

사용자와 상호작용하는 웹 페이지는 React.js 를 이용하여 개발할 것이다. http 를 통해 server 와 통신하고, server 의 데이터를 받아와 사용자에게 제공하는 형식이다. 다양한 플랫폼에서 사용하기 쉽도록 반응형 웹으로 구성할 것이다.

2-3-2. Back-end

server 에는 선행지표들에 따라 매일매일 다르게 학습되어 각 종목에 대해 인공지능이 추천해주는 주식투자전략(매수, 매도, 관망)이 저장된다. 한국장 마감시간(18:00) 이후에 강화학습을 통하여 도출된 결과값이 22 시마다 server 에 동기화 되며, 데이터베이스에 저장된 데이터는 웹과 연동되어 사용자들은 매일 업데이트 된 내용을 받아볼 수 있다.

3. 구성원별 진척도 및 과제 추진 계획

3-1. 구성원별 과제 진척도

정희영	주어진 환경에서 에이전트에게 강화학습 적용 (완료) 학습에 필요한 하이퍼파라미터 최적화하기 (진행중) 사용자에게 제공할 지표 수치화하기 (진행중)
박동진	전체적인 프로젝트 구상 (완료) 강화학습 환경 개발 (요구사항에 맞게 수정중) 프론트엔트 개발 (진행중)
신재환	강화학습을 위한 데이터 수집 (완료) 수집한 데이터를 저장할 DB 구축 (진행중)

3-2. 과제 추진 계획

이름	날짜	담당자	상태
수집할 데이터 선정	23/06/01 ~ 23/06/13	신재환	완료
수집할 종목 선정	23/06/19 ~ 23/06/25	신재환	완료
환경 개발	23/06/23 ~ 23/07/07	박동진	완료
키움 OpenAPI 이용 및 데이터 수집	23/06/26 ~ 23/07/12	신재환	진행중
강화학습 공부 및 적용	23/07/04 ~ 23/07/14	정희영	완료
KRX 정보데이터시스템 크롤링 프로그램 개발	23/07/11 ~ 23/07/28	신재환	진행중
하이퍼파라미터 튜닝	23/07/15 ~ 23/08/15	정희영	진행중
모델 성능 테스트	23/07/15 ~ 23/08/15	정희영	진행중
프론트엔드 개발	23/07/24 ~ 23/08/25	박동진	진행중
환경 수정	23/07/28 ~ 23/08/07	박동진	진행중
프로그램 요구사항 수정	23/07/30 ~ 23/08/07		진행중
서비스 테스트	23/09/01 ~ 23/09/22		예정
최종보고서	23/09/22 ~ 23/09/29		예정

4. 과제 수행 내용 및 중간 결과

4-1. 데이터 수집

KRX 정보데이터시스템에서 데이터를 받아오기 위해서 웹크롤링 프로그램을 개발하였다. 소스코드는 다음과 같다.

```

1 from time import sleep
2 from bs4 import BeautifulSoup
3 from selenium import webdriver
4 from selenium.webdriver.chrome.service import Service
5 from webdriver_manager.chrome import ChromeDriverManager
6 from selenium.webdriver.common.by import By
7 import pandas as pd
8 from selenium.webdriver.common.keys import Keys
9 from selenium.webdriver.support.ui import WebDriverWait
10 from selenium.webdriver.support import expected_conditions as EC
11 from selenium.webdriver.common.by import By
12 from selenium.webdriver.common.action_chains import ActionChains
13
14 import requests
15 import pandas as pd
16
17 webdriver_service = Service(ChromeDriverManager().install())
18 driver = webdriver.Chrome(service=webdriver_service)
19
20 usage
21 def getCodeList():
22     response = requests.get("https://finance.naver.com/sise/sise_market_sum.nhn")
23     elements = BeautifulSoup(response.text, 'html.parser').find_all(class_="title")
24     #codes = [element.get('href').split('=')[1] for element in elements]
25     #mandatory_codes = ['005930', '000660', '000720', '005490', '036570']
26     #for code in mandatory_codes:
27     #    if code not in codes:
28     #        codes.append(code)
29     codes = ['005930', '000660', '000720', '005490', '036570']
30     return codes
31
32 HEADER = ['일자', '종가', '대버', '등락률', 'EPS', 'PER', '선형 EPS', '선형 PER', 'BPS', 'PBR', '주당배당금', '배당수익률']
33 START_YEAR = 2014
34 END_YEAR = 2023

```

```

35 def get_data_rows(tbody):
36     rows = tbody.find_elements(By.TAG_NAME, "tr")
37     data = []
38     for row in rows:
39         cols = row.find_elements(By.TAG_NAME, "td")
40         row_data = []
41         for ele in cols:
42             data_name = ele.get_attribute('data-name')
43             if data_name in ['CMPPREVDD_PRC', 'FLUC_RT']:
44                 span = ele.find_elements(By.XPATH, ".*//span")
45                 if span:
46                     if data_name == 'CMPPREVDD_PRC' and "CI-GRID-UPDOWN-DOWN" in span[0].get_attribute('class'):
47                         value = "-" + span[0].get_attribute('innerHTML')
48                     else:
49                         value = span[0].get_attribute('innerHTML')
50                 else:
51                     value = ele.get_attribute('innerHTML')
52             else:
53                 value = ele.get_attribute('innerHTML')
54             row_data.append(value)
55         data.append(row_data)
56     return data
57
58 try:
59     driver.get("http://data.krx.co.kr/contents/MDC/MDI/mdiloder/index.cmd?menuId=MDC0201020502")
60     driver.execute_cdp_cmd("Page.addScriptToEvaluateOnNewDocument", {"source": ""})
61     searchType = WebDriverWait(driver, 10).until(EC.element_to_be_clickable((By.ID, 'searchType_0_1')))
62     driver.execute_script("arguments[0].click();", searchType)
63     finder = driver.find_element(By.ID, 'tbodyisuCd_finder_stkisu0_0')
64     startCalendar = driver.find_element(By.ID, 'startCalendar')
65     endCalendar = driver.find_element(By.ID, 'endCalendar')
66     searchButton = driver.find_element(By.ID, 'jsSearchButton')
67
68     for code in getCodeList():
69         finder.clear()
70         finder.send_keys(code)
71         finder.send_keys(Keys.ENTER)
72         while '/' not in str(finder.get_attribute('value')):
73             sleep(1)
74         dfSum = pd.DataFrame()
75         for year in range(END_YEAR, START_YEAR - 1, -1):
76             for half in range(2):
77                 startCalendar.clear()
78                 endCalendar.clear()
79                 startCalendar.send_keys('{0}{1}01'.format(year, '07' if half == 0 else '01'))
80                 endCalendar.send_keys('{0}{1}{2}'.format(year, '12' if half == 0 else '06', '31' if half == 0 else '30'))
81                 searchButton.click()
82                 sleep(1)
83                 tbody = driver.find_elements(By.CLASS_NAME, 'CI-GRID-BODY-TABLE-TBODY')[1]
84                 data = get_data_rows(tbody)
85                 df = pd.DataFrame(data)
86                 dfSum = pd.concat([dfSum, df], ignore_index=True)
87                 sleep(2)
88             dfSum.to_csv('{0}.csv'.format(code))
89 finally:
90     driver.quit()

```

그림 7. 데이터 수집 코드

23 line 에서 27 line 까지는 시총 상위 50 개 종목에 대한 데이터를 받아오는 코드이다. 우리는 우선적으로 삼성전자, SK 하이닉스, 현대건설, POSCO 홀딩스, 엔씨소프트의 5 가지 종목만 학습시키기로 하였고, 추후에 추가적으로 시총 상위 50 개 종목에 대해서도 학습시켜보고자 한다. 77 line 을 보면 for half in range(2)라고 되어있는데, 이는 1 년 단위로 데이터를 수집하려고 하니, KRX 정보데이터시스템은 주식 정보를 활용한 프로그램 개발을 위해 만들어진 시스템이 아니다보니 서버 자체에서 IP 를 차단하였다. 따라서, 반기 단위로 데이터를 수집하도록 코드를 수정하였다.

또한, 10 년 이상의 많은 양의 데이터 수집해야하기에 크롤링 속도를 올렸더니 서버 자체에서 제재를 하여서 일정 속도로 조정 적용하여 문제를 해결하였다.

코드를 실행시키면 자동으로 KRX 정보데이터시스템 웹사이트 창이 생성되며, 먼저 검색한 종목코드의 데이터를 수집해온다. 수집해온 데이터는 아래의 이미지와 같이 csv 파일로 저장되고 저장된 csv 파일은 강화학습을 진행하는 데에 사용된다.

```
1 0,1,2,3,4,5,6,7,8,9,10,11
2 0,2023/07/21,"70,300",-700,-0.99,"8,057",8.73,"3,505",20.06,"57,822",1.22,"1,444",2.05
3 1,2023/07/20,"71,000",-700,-0.98,"8,057",8.81,"3,505",20.26,"57,822",1.23,"1,444",2.03
4 2,2023/07/19,"71,700",-300,-0.42,"8,057",8.90,"3,505",20.46,"57,822",1.24,"1,444",2.01
5 3,2023/07/18,"72,000",-1,300,-1.77,"8,057",8.94,"3,505",20.54,"57,822",1.25,"1,444",2.01
6 4,2023/07/17,"73,300",-100,-0.14,"8,057",9.10,"3,505",20.91,"57,822",1.27,"1,444",1.97
7 5,2023/07/14,"73,400",1,500,+2.09,"8,057",9.11,"3,505",20.94,"57,822",1.27,"1,444",1.97
8 6,2023/07/13,"71,900",0,0.00,"8,057",8.92,"3,505",20.51,"57,822",1.24,"1,444",2.01
9 7,2023/07/12,"71,900",400,+0.56,"8,057",8.92,"3,505",20.51,"57,822",1.24,"1,444",2.01
10 8,2023/07/11,"71,500",2,000,+2.88,"8,057",8.87,"3,505",20.40,"57,822",1.24,"1,444",2.02
11 9,2023/07/10,"69,500",-400,-0.57,"8,057",8.63,"3,384",20.54,"57,822",1.20,"1,444",2.08
12 10,2023/07/07,"69,900",-1,700,-2.37,"8,057",8.68,"3,384",20.65,"57,822",1.21,"1,444",2.07
13 11,2023/07/06,"71,600",-400,-0.56,"8,057",8.89,"3,384",21.16,"57,822",1.24,"1,444",2.02
14 12,2023/07/05,"72,000",-1,000,-1.37,"8,057",8.94,"3,384",21.28,"57,822",1.25,"1,444",2.01
15 13,2023/07/04,"73,000",0,0.00,"8,057",9.06,"3,370",21.66,"57,822",1.26,"1,444",1.98
16 14,2023/07/03,"73,000",800,+1.11,"8,057",9.06,"3,106",23.50,"57,822",1.26,"1,444",1.98
17 15,2023/06/30,"72,200",-200,-0.28,"8,057",8.96,"3,106",23.24,"57,822",1.25,"1,444",2.00
18 16,2023/06/29,"72,400",-300,-0.41,"8,057",8.99,"3,120",23.21,"57,822",1.25,"1,444",1.99
19 17,2023/06/28,"72,700",100,+0.14,"8,057",9.02,"3,147",23.10,"57,822",1.26,"1,444",1.99
20 18,2023/06/27,"72,600",200,+0.28,"8,057",9.01,"3,162",22.96,"57,822",1.26,"1,444",1.99
21 19,2023/06/26,"72,400",800,+1.12,"8,057",8.99,"3,162",22.90,"57,822",1.25,"1,444",1.99
22 20,2023/06/23,"71,600",300,+0.42,"8,057",8.89,"3,162",22.64,"57,822",1.24,"1,444",2.02
23 21,2023/06/22,"71,300",800,+1.13,"8,057",8.85,"3,162",22.55,"57,822",1.23,"1,444",2.03
24 22,2023/06/21,"70,500",-900,-1.26,"8,057",8.75,"3,162",22.29,"57,822",1.22,"1,444",2.05
25 23,2023/06/20,"71,400",200,+0.28,"8,057",8.86,"3,172",22.51,"57,822",1.23,"1,444",2.02
26 24,2023/06/19,"71,200",-600,-0.84,"8,057",8.84,"3,172",22.44,"57,822",1.23,"1,444",2.03
27 25,2023/06/16,"71,800",300,+0.42,"8,057",8.91,"3,172",22.63,"57,822",1.24,"1,444",2.01
28 26,2023/06/15,"71,500",-400,-0.56,"8,057",8.87,"3,144",22.74,"57,822",1.24,"1,444",2.02
29 27,2023/06/14,"71,900",-100,-0.14,"8,057",8.92,"3,144",22.87,"57,822",1.24,"1,444",2.01
30 28,2023/06/13,"72,000",1,000,+1.41,"8,057",8.94,"3,125",23.04,"57,822",1.25,"1,444",2.01
31 29,2023/06/12,"71,000",-1,000,-1.39,"8,057",8.81,"3,125",22.72,"57,822",1.23,"1,444",2.01
32 30,2023/06/09,"72,000",1,100,+1.55,"8,057",8.94,"3,125",23.04,"57,822",1.25,"1,444",2.01
33 31,2023/06/08,"70,900",-100,-0.14,"8,057",8.80,"3,125",22.69,"57,822",1.23,"1,444",2.04
34 32,2023/06/07,"71,000",-700,-0.98,"8,057",8.81,"3,125",22.72,"57,822",1.23,"1,444",2.03
35 33,2023/06/05,"71,700",-500,-0.69,"8,057",8.90,"3,125",22.94,"57,822",1.24,"1,444",2.01
36 34,2023/06/02,"72,200",1,300,+1.83,"8,057",8.96,"3,125",23.10,"57,822",1.25,"1,444",2.00
37 35,2023/06/01,"70,900",-500,-0.70,"8,057",8.80,"2,852",24.86,"57,822",1.23,"1,444",2.04
38 36,2023/05/31,"71,400",-900,-1.24,"8,057",8.86,"2,808",25.42,"57,822",1.23,"1,444",2.02
39 37,2023/05/30,"72,300",2,000,+2.84,"8,057",8.97,"2,825",25.59,"57,822",1.25,"1,444",2.00
40 38,2023/05/26,"70,300",1,500,+2.18,"8,057",8.73,"2,825",24.88,"57,822",1.22,"1,444",2.05
41 39,2023/05/25,"68,800",300,+0.44,"8,057",8.54,"2,825",24.35,"57,822",1.19,"1,444",2.10
42 40,2023/05/24,"68,500",100,+0.15,"8,057",8.50,"2,825",24.24,"57,822",1.18,"1,444",2.11
43 41,2023/05/23,"68,400",-100,-0.15,"8,057",8.49,"2,823",24.23,"57,822",1.18,"1,444",2.11
44 42,2023/05/22,"68,500",100,+0.15,"8,057",8.50,"2,823",24.26,"57,822",1.18,"1,444",2.11
45 43,2023/05/19,"68,400",2,200,+3.32,"8,057",8.49,"2,823",24.23,"57,822",1.18,"1,444",2.11
46 44,2023/05/18,"66,200",1,200,+1.85,"8,057",8.22,"2,823",23.45,"57,822",1.14,"1,444",2.18
47 45,2023/05/17,"65,000",-400,-0.61,"8,057",8.07,"2,815",23.09,"57,822",1.12,"1,444",2.22
48 46,2023/05/16,"65,400",900,+1.40,"8,057",8.12,"2,815",23.24,"57,822",1.13,"1,444",2.21
49 47,2023/05/15,"64,500",400,+0.62,"8,057",8.01,"2,815",22.92,"57,822",1.12,"1,444",2.24
```

그림 8. 그림 7 코드의 실행으로 얻은 csv 파일의 내용

4-2. 강화학습

우선 기본적으로 stable-baselines3 라이브러리에 구현된 DQN, A2C 메서드와 우리가 구현한 환경을 이용해서 모델을 학습시켰다.

아래의 코드는 stable-baselines3 에서 제공하는 DQN 알고리즘의 일부이다.

```
class stable_baselines3.dqn.DQN(policy, env, learning_rate=0.0001, buffer_size=1000000,
learning_starts=50000, batch_size=32, tau=1.0, gamma=0.99, train_freq=4, gradient_steps=1,
replay_buffer_class=None, replay_buffer_kwargs=None, optimize_memory_usage=False,
target_update_interval=10000, exploration_fraction=0.1, exploration_initial_eps=1.0,
exploration_final_eps=0.05, max_grad_norm=10, stats_window_size=100, tensorboard_log=None,
policy_kwargs=None, verbose=0, seed=None, device='auto', _init_setup_model=True) [source]
```

그림 9. Stable baseliens3 공식 문서의 DQN 알고리즘

아래는 모델 성능에 유의미한 차이를 주는 hyperparameter 에 대한 설명이다.

- **learning_rate** (Union[float, Callable[[float], float]]) – The learning rate, it can be a function of the current progress remaining (from 1 to 0)
- **buffer_size** (int) – size of the replay buffer
- **batch_size** (int) – Minibatch size for each gradient update
- **exploration_fraction** (float) – fraction of entire training period over which the exploration rate is reduced
- **exploration_final_eps** (float) – final value of random action probability

아래의 코드는 stable-baselines3 에서 제공하는 A2C 알고리즘의 일부이다.

```
class stable_baselines3.a2c.A2C(policy, env, learning_rate=0.0007, n_steps=5, gamma=0.99,
gae_lambda=1.0, ent_coef=0.0, vf_coef=0.5, max_grad_norm=0.5, rms_prop_eps=1e-05, use_rms_prop=True,
use_sde=False, sde_sample_freq=-1, normalize_advantage=False, stats_window_size=100,
tensorboard_log=None, policy_kwargs=None, verbose=0, seed=None, device='auto', _init_setup_model=True)
```

그림 10. Stable baselines3 공식 문서의 A2C 알고리즘

아래는 모델 성능에 유의미한 차이를 주는 hyperparameter 에 대한 설명이다.

- **learning_rate** (Union[float, Callable[[float], float]]) – The learning rate, it can be a function of the current progress remaining (from 1 to 0)
- **n_steps** (int) – The number of steps to run for each environment per update (i.e. batch size is n_steps * n_env where n_env is number of environment copies running in parallel)
- **gae_lambda** (float) – Factor for trade-off of bias vs variance for Generalized Advantage Estimator. Equivalent to classic advantage when set to 1.
- **ent_coef** (float) – Entropy coefficient for the loss calculation
- **vf_coef** (float) – Value function coefficient for the loss calculation

우선 hyperparameter 들을 알맞게 조정하며 여러 차례 학습을 시키면서 성능을 평가하고 최적의 hyperparameter 들을 찾고 있다. 성능을 평가하는 기준은 얼마나 안정적으로 최대한 높은 수익률을 기록하는지를 평가하였다.

학습과 테스트에는 약 10 년치 주식 데이터를 사용했다. 학습시에는 약 8 년치의 주식 데이터를, 테스트시에는 약 2 년치의 주식 데이터를 이용하였다.

아래의 코드는 stable-baselines3 라이브러리를 이용해서 DQN 알고리즘을 이용해 에이전트를 학습시키고 tensorboard log 와 모델을 별도의 폴더로 저장하는 코드이다. tensorboard log 는 모델이 timesteps 를 거치면서 학습 시간과 리워드의 변화를 확인할 수 있는 로그이다.

```
import src.env.env_bs as env
import src.utils.utils as utils
import pandas as pd
from stable_baselines3 import DQN

urls = './data/sk_hynix_train.csv'

DEFAULT_OPTION = {
    'initial_balance_coef': 10, # 초기 자본금 계수, 초기 자본금 = 최대종가 * 비율정밀도 * 계수, 1일 때, 최대가격을 기준으로 모든 비율의 주식을 구매할 수 있도록 함
    'start_index': 0, # 학습 시작 인덱스
    'end_index': 500, # 학습 종료 인덱스
    'window_size': 20, # 학습에 사용할 데이터 수, 최근 수치에 따라 얼마나 많은 데이터를 사용할지 결정
    'proportion_precision': 4, # 비율 정밀도 (결정할 수 있는 비율의 수) 20이면 0.05 단위로 결정
    'commission': .0003, # 수수료
    'selling_tax': .00015, # 매도세
    'reward_threshold': 0.03, # 보상 임계값 : 수익률이 이 값을 넘으면 보상을 1로 설정
}

stock_info = utils.load_data(urls)

env = env.MyEnv(stock_info, option=DEFAULT_OPTION)

model = DQN('MultiInputPolicy', env, verbose=1, tensorboard_log='./logs/DQN/')

model.learn(total_timesteps=100000, log_interval=1, progress_bar=True)

model.save("./models/DQN_hynix_model.zip")
```

그림 11. Stable baseliens3 라이브러리를 사용한 강화학습 코드

아래의 두 이미지는 위의 코드를 실행하여 얻은 tensorboard log 를 확인한 결과이다. 첫번째 이미지는 timesteps 이 지남에 따른 학습 시간을 나타내는 그래프이다. 두번째 이미지는 timesteps 이 지남에 따른 리워드 평균을 나타내는 그래프이다. timesteps 에 따라 리워드 값이 증가하고 감소하기를 반복하기 때문에 해당 그래프를 보고 적절한 timesteps 로 모델을 학습시키는 것도 중요하다.



그림 12. 학습 시간 평균 graph

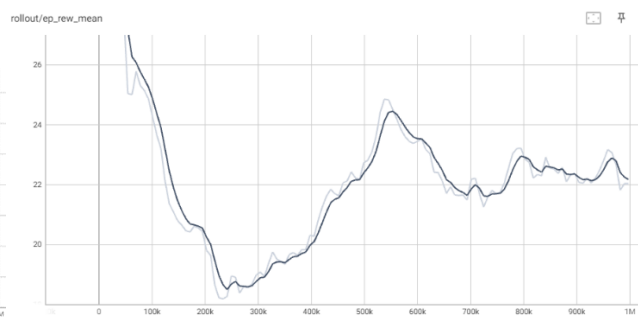


그림 13. Reward 평균 그래프

아래의 코드는 stable-baselines3 라이브러리를 이용해서 저장된 모델을 불러와 테스트 환경에서 실제로 투자를 하면서 얼마의 수익률을 기록했는지를 그래프로 나타내고 해당 state 에서 어떤 행동을 선택하여 매도, 매수, 관망을 몇 번씩 수행했는지를 확인할 수 있는 코드이다.

```
import src.env.env_bs as env
import src.utils.utils as utils
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from stable_baselines3 import DQN

urls = './data/sk_hynix_test.csv'

DEFAULT_OPTION = {
    'initial_balance_coef': 10,
    'start_index': 0,
    'end_index': 200,
    'window_size': 20,
    'proportion_precision': 4,
    'commission': .0003,
    'selling_tax': .00015,
    'reward_threshold': 0.03,
}

stock_info = utils.load_data(urls)

env = env.MyEnv(stock_info, option=DEFAULT_OPTION)

model = DQN.load("models/DQN_0.0001_1000000_32_0.1_0.05.zip", env)
ITER_LIMIT = len(stock_info)

obs = env.reset()

profits = []
sell = 0
buy = 0
hold = 0
```



```

for i in range(ITER_LIMIT):
    action, _ = model.predict(obs, deterministic=False)

    if action == 0:
        if obs['holding'] == 4:
            hold += 1
        else:
            buy += 1
    elif action == 1:
        if obs['holding'] == 0:
            hold += 1
        else:
            sell += 1
    else:
        hold += 1

    print(i + 1, "번째 action : ", action)

    obs, reward, done, info = env.step(action)

    print(i + 1, "번째 reward : ", reward)
    print(i + 1, "번째 info : ", info, obs['holding'])

    profits.append(info['profit'])

if i > ITER_LIMIT:
    done = True

if done:
    obs = env.reset()
    print("Episode done!")
    print("매도 :", sell, "\n매수 :", buy, "\n관망 :", hold)
    break

plt.plot(profits)
plt.xlabel('Episode')
plt.ylabel('Profit')
plt.title('Profit Graph')
plt.show()

```

그림 14. 학습된 모델을 불러와 테스트하는 코드

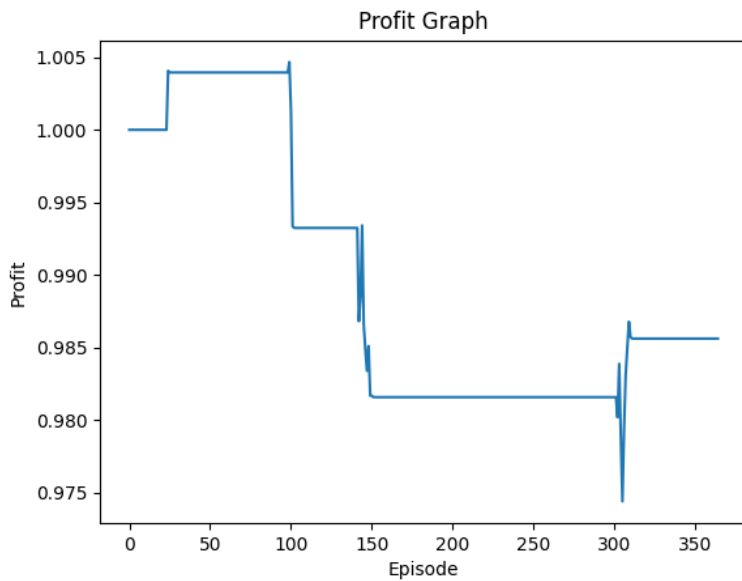
```

365 번째 action : 1
365 번째 reward : 0
365 번째 info : {'index': 383, 'total_value': 5243508.390000001, 'profit': 0.985621877819549, 'action': array(1, dtype=int64)} 0
Episode done!
매도 : 4
매수 : 4
관망 : 357

```

그림 15. 그림 14의 코드를 실행하고 얻은 출력 결과

위 이미지는 코드를 실행했을 때 얻을 수 있는 출력 결과이다.

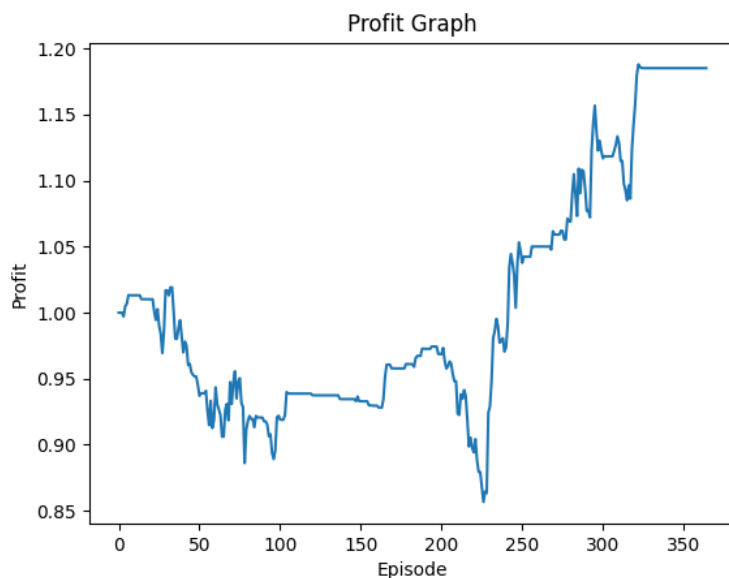


(해당 그래프는 별도로 hyperparameter를 조정하지 않은 결과이다.

learnig_rate : 0.0001
 buffer_size : 1000000
 batch_size : 32
 exploration_fraction : 0.1
 exploartion_final_eps : 0.05)

그림 16. 그림 14의 코드를 실행하고 얻은 profit 그래프

위 이미지는 코드를 실행했을 때 얻을 수 있는 수익률 그래프이다.



(해당 그래프는 별도로 hyperparameter를 조정하지 않은 결과이다.

learnig_rate : 0.0001
 buffer_size : 1000
 batch_size : 64
 exploration_fraction : 0.1
 exploartion_final_eps : 0.05)

그림 17. Hyperparameter를 조정하여 얻은 profit 그래프

위 이미지는 별도로 hyperparameter를 조정해서 얻은 결과이다.

위의 두 그래프에서 알 수 있듯이 hyperparameter에 따라 모델의 수익률에 큰 차이가 있으므로 hyperparameter tuning도 중요한 과정이다.

hyperparmeter 튜닝을 위해 여러 방법을 찾아보고 시도해보았는데 stable-baselines3 라이브러리를 이용해 학습시킨 모델에는 적용이 어려웠다. (GridSearch, optuna, etc.) 계속 다른 방법을 찾아보고 있지만 그 전까지는 brute-force 하게 다중 for 문을 이용해서 모델을 학습시키고 모델의 성능을 확인하며 hyperparameter 튜닝을 진행하고 있다.

아래는 다중 for문을 이용해 brute-force하게 hyperparameter 튜닝을 진행하는 코드이다.

```
import numpy as np
from stable_baselines3 import DQN
from stable_baselines3 import A2C
import src.env.env_bs as env
import src.utils.utils as utils

urls = './data/sk_hynix_train.csv'

DEFAULT_OPTION = {
    'start_index': 0,
    'end_index': 200,
    'window_size': 20,
    'proportion_precision': 4,
    'commission': .0003,
    'selling_tax': .00015,
    'reward_threshold': 0.03,
}

stock_info = utils.load_data(urls)

env = env.MyEnv(stock_info, option=DEFAULT_OPTION)

# dqn's hyperparameter

batch_size = [32, 64]
buffer_size = [1_000, 10_000, 100_000, 1_000_000]
dqn_learning_rate = [0.0001, 0.0003, 0.0005]
exploration_fraction = [0.05, 0.1, 0.2]
exploration_final_eps = [0.1, 0.05, 0.03]

# a2c's hyperparameter

a2c_learning_rate = [0.0005, 0.0007, 0.001]
ent_coef = [0.0, 0.05, 0.1]
vf_coef = [0.4, 0.5, 0.6]
gae_lambda = [0.9, 0.95, 1.0]
n_steps = [7, 5, 3]
```

```

for learning_rate in dqn_learning_rate:
    for buffer_size in buffer_size:
        for batch_size in batch_size:
            for exploration_fraction in exploration_fraction:
                for exploration_final_eps in exploration_final_eps:
                    model=DQN('MultiInputPolicy', env, verbose=1,
                               learning_rate=learning_rate, buffer_size=buffer_size,
                               batch_size=batch_size, exploration_fraction=exploration_fraction,
                               exploration_final_eps=exploration_final_eps,
                               tensorboard_log="./logs/FindHyperparam/DQN/")
                    model.learn(total_timesteps=1_000_000, progress_bar=True,
                                tb_log_name=f"{learning_rate}_{buffer_size}_{batch_size}"
                                f"{exploration_fraction}_{exploration_final_eps}")
                    model.save(f"./models/DQN_{learning_rate}_{buffer_size}_{batch_size}"
                               f"{exploration_fraction}_{exploration_final_eps}.zip")

for learning_rate in a2c_learning_rate:
    for n_steps in n_steps:
        for gae_lambda in gae_lambda:
            for ent_coef in ent_coef:
                for vf_coef in vf_coef:
                    model=A2C('MultiInputPolicy', env, verbose=1,
                               learning_rate=learning_rate, n_steps=n_steps,
                               gae_lambda=gae_lambda, ent_coef=ent_coef, vf_coef=vf_coef,
                               tensorboard_log="./logs/FindHyperparam/A2C/")
                    model.learn(total_timesteps=1_000_000, progress_bar=True,
                                tb_log_name=f"{learning_rate}_{n_steps}_{gae_lambda}_{ent_coef}"
                                f"{vf_coef}")
                    model.save(f"./models/A2C_{learning_rate}_{n_steps}_{gae_lambda}"
                               f"{ent_coef}_{vf_coef}.zip")

```

그림 18. 하이퍼파라미터 튜닝을 위한 코드

아래의 이미지는 위의 코드를 실행시키고 얻은 모델들의 일부분이다.

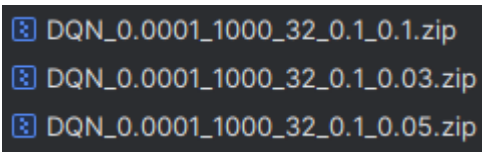


그림 19. 그림 18의 코드를 실행하고 저장된 모델의 일부

사용자에게 잘 학습된 모델이 특정 상황에서 어떤 action을 선택하는지를 확률로 나타내는지, 해당 action을 선택할 때 얻는 기대 수익을 제공하기 위해서 stable-baselines3 라이브러리를 임의로 수정하여 A2C 알고리즘을 이용해 학습된 모델의 경우 action을 선택할 확률을 반환하게 하고 DQN 알고리즘을 이용해 학습된 모델의 경우 action을 선택할 때 기대하는 q-value를 반환하게 하였다. q-value가 의미하는 바는 '해당 행동을 선택했을 때 얼마의 reward를 획득할 수 있을 것인가'이다.

(여기서의 reward는 수익률과는 다른 값이다.)

```
action q-value : tensor([[3003.1257, 3012.9211, 3012.7720]])
```

그림 20. Action 에 대한 q-value

```
action probability : tensor([0.3352, 0.3083, 0.3565])
```

그림 21. Action 에 대한 probability

위의 두 이미지가 action 에 대한 확률과 q-value 를 구한 값이다.

action 에 대한 확률은 그대로 사용자에게 제공해도 사용자가 직관적으로 어떤 의미인지 이해할 수 있지만 q-value 의 경우 사용자에게 그대로 제공하면 정확히 어떤 의미인지 알 수 없으므로 q-value 를 사용자가 이해하기 쉽고 어떤 의미인지 정확하게 하기 위해서 따로 유의미한 값으로 변환시킬 필요가 있을 것이라고 판단한다.

4-2-1. 환경 개발

환경은 action 이 제시될 경우 현재 총 자산에서 주식의 보유 비율을 조정하고, 이를 반영하기 위해 매매를 진행하는 방식으로 작성되었다. 기본적으로 주식투자에서 정석으로 여겨지는 전략인 분할매수, 분할매도를 유도하면 더 좋은 성능을 낼 것으로 기대했다. 비율을 조정하기 위해 매수/매도가 일어나고 상태가 업데이트된다. 보유 비율을 얼마나 정밀하게 할 것인지는 option 으로 제공되는 **proportion_precision** 이라는 변수를 통해 조정할 수 있다. 아래 두 이미지의 경우 **proportion_precision** 은 4 로, 25%단위로 주식의 보유 비율이 조정되는 것이다.

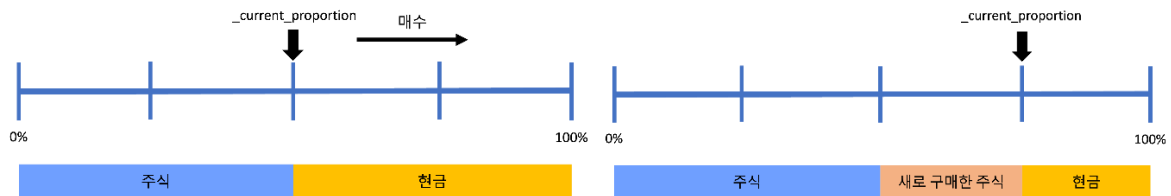


그림 22, 그림 23. Proportion_precision 의 의미 이해를 돕기 위한 이미지

완성한 환경의 구조는 아래 이미지와 같다. m 은 method 를 의미하고 f 는 field 를 의미한다. 보고서에서는 학습에 크게 영향을 끼치는 요소들만 기록하였다.

⑩ <code>__init__(self, df, option=DEFAULT_OPTION)</code>	① <code>_profit</code>
⑩ <code>reset(self)</code>	① <code>_reward_huddle</code>
⑩ <code>_observe(self)</code>	① <code>_state</code>
⑩ <code>step(self, action)</code>	① <code>_stock_data</code>
⑩ <code>_update_history(self, info)</code>	① <code>_total_value</code>
⑩ <code>_get_hold_amount(self)</code>	① <code>action_space</code>
⑩ <code>_buy(self, amount)</code>	① <code>commission</code>
⑩ <code>_sell(self, amount)</code>	① <code>df</code>
⑩ <code>_get_reward(self)</code>	① <code>end_index</code>
⑩ <code>_get_profit(self)</code>	① <code>init_balance</code>
⑩ <code>_get_total_value(self)</code>	① <code>metadata</code>
⑩ <code>_get_done(self)</code>	① <code>number_of_properties</code>
⑩ <code>print_current_state(self)</code>	① <code>observation_space</code>
① <code>_avg_buy_price</code>	① <code>option</code>
① <code>_balance</code>	① <code>price</code>
① <code>_current_index</code>	① <code>proportion_precision</code>
① <code>_current_proportion</code>	① <code>reward_threshold</code>
① <code>_done</code>	① <code>selling_tax</code>
① <code>_history</code>	① <code>shape</code>
① <code>_holdings</code>	① <code>start_index</code>
	① <code>window_size</code>

그림 24, 25. 환경의 구조

환경에 들어가는 설정

1. hyperparameter

```

DEFAULT_OPTION = {
    'initial_balance_coef': 10, # 초기 자본금 계수, 초기 자본금 = 최대종가 * 비율정밀도 * 계수, 1일 때, 최대가격을 기준으로 모든 비율의 주식을 구매할 수 있도록 함
    'start_index': 0, # 학습 시작 인덱스
    'end_index': 500, # 학습 종료 인덱스
    'window_size': 20, # 학습에 사용할 데이터 수, 최근 수치에 따라 얼마나 많은 데이터를 사용할지 결정
    'proportion_precision': 4, # 비율 정밀도 (결정할 수 있는 비율의 수) 20이면 0.05 단위로 결정
    'commission': .0003, # 수수료
    'selling_tax': .00015, # 매도세
    'reward_threshold': 0.03, # 보상 임계값 : 수익률이 이 값을 넘으면 보상을 1로 설정
}

```

그림 26. 환경에 대한 hyperparameter

환경은 상황에 따라 다를 수 있다. 다양한 환경에서 학습을 진행하기 위해서, option 을 만들어 두었다. option 을 통해 초기 자본금, 학습시작일자, 종료일자, agent 에게 제공되는 최근 주식데이터의 범위, 거래 단위, 거래수수료, 매도세, 보상임계치를 설정할 수 있다. 각 요소들은 환경에 따라 다를 수 있으며, 학습에 다양한 영향을 끼칠 수 있다. 이러한 수치들을 조정하면서, 학습을 진행할 수 있도록 했다.

2. Action Space

```

# 가능한 Action은 보유비율 증가, 유지, 감소 세 가지
self.action_space = gym.spaces.Discrete(3)

```

그림 27. Action space

Agent 가 취할 수 있는 행동은 매도, 매수, 관망으로 이산적으로 정의했다.

3. Observation Space(state)

```
# Agent가 획득하는 데이터의 형태 (감시할 기간, 감시할 데이터의 수)
self.shape = (option['window_size'], self.number_of_properties)

self.observation_space = gym.spaces.Dict({
    # 보유 주식 비율
    "holding": gym.spaces.Discrete(self.option['proportion_precision'])

    # 평균 매수가
    "avg_buy_price": gym.spaces.Box(
        low=0,
        high=np.inf,
        dtype=np.float32
    ),

    # 주가데이터는 종목의 수 * 사용할 데이터 수
    "stock_data": gym.spaces.Box(
        low=np.inf,
        high=np.inf,
        shape=self.shape,
        dtype=np.float32
    )
})
```

그림 28. Observation space 코드

Agent 가 관측할 수 있는 state 는 현재 보유량, 평균 매수가, 주가데이터이다. dictionary 형태로, 각 특성에 맞게 정의했다.

reset()

reset() 메서드는 환경을 초기화시킨다. 시간을 첫 번째 날짜로 되돌리고, 포트폴리오와 수익을 초기값으로 되돌린다.

```
def reset(self) -> ObsType:
    # 초기화
    self._holdings = 0
    self._avg_buy_price = 0
    self._current_proportion = 0
    self._current_proportion = np.clip(self._current_proportion, 0, self.proportion_precision)
    self._profit = 1.0
    self._total_value = self.init_balance
    self._balance = self.init_balance
    self._current_index = self.start_index
    self._reward_huddle = self.init_balance
    self._history = []
    self._done = False
    self._state = {}
    # 초기 상태 반환
    return self._observe()
```

그림 29. reset 메서드

self._get_reward()

`self._get_reward()` 는 최근 상황에 대한 reward 를 제공한다. 많은 주식을 보유하고 있을 때, 최근 주가가 상승했으면 +보상을 주고, 하락했으면 -보상을 주는 식이다. 작은 차트의 변화에도 agent 가 민감하게 작동하는 것을 방지하기 위해, 일정 수준(`reward_threshold`) 이상의 변동이 있을 때에만 돌파한 정도에 따라 보상이 발생하게 작성하였다. 보상이 발생하면 해당 시점의 포트폴리오 가치를 기준(`reward_huddle`)으로 임계점을 돌파해야만 다음 보상이 발생한다. 이는 지속적인 보상으로 인해 agent 가 보상에 둔감해지는 것을 방지한다. 보상의 크기는 임계점을 얼마나 돌파했는지에 따라 결정된다.

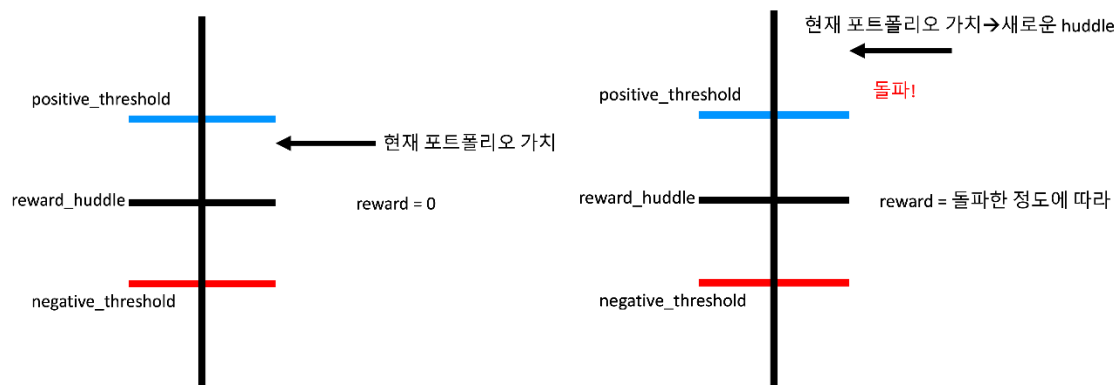


그림 30, 그림 31. Get_reward 메서드의 이해를 돕기 위한 이미지

```
def _get_reward(self) -> float:
    # 보상은 수익이 reward_huddle의 +- {reward_threshold}% 범위에 들면 0,
    # 아니면 수익의 +- /{reward_threshold}% 만큼의 보상
    reward_coefficient = self._reward_huddle * self.reward_threshold
    positive_threshold = self._reward_huddle + reward_coefficient
    negative_threshold = self._reward_huddle - reward_coefficient
    if self._total_value > positive_threshold or self._total_value < negative_threshold:
        reward = (self._total_value - self._reward_huddle) / reward_coefficient
        self._reward_huddle = self._total_value
        return reward
    else:
        return 0
```

그림 32. get_reward 메서드

step(ActType)

`step()` 메서드는 주어진 `action` 을 환경에 반영하고, 그에 따른 `reward` 와 다음 `state`, 환경 종료 여부를 반환한다. 마지막 일자에 다다르면 환경이 종료되는 것이다. 환경이 종료되면 `reset()` 을 통해 환경을 초기화시키고 다시 학습을 진행할 것이다.

```
def step(self, action: ActType) -> Tuple[ObsType, float, bool, dict]:
    # 종료 여부 확인
    if self._done:
        raise Exception("Episode is done")

    # 매수
    if action == 0:
        if not (self._current_proportion == self.proportion_precision):
            # 매수량 결정하기
            self._current_proportion += 1
            amount = int(self._get_hold_amount() - self._holdings)
            self._buy(amount)

    # 매도
    elif action == 1:
        if not (self._current_proportion == 0):
            # 매도량 결정하기
            self._current_proportion -= 1
            amount = int(self._holdings - self._get_hold_amount())
            self._sell(amount)

    # action 후 기록
    info = dict(
        index=self._current_index,
        total_value=self._total_value,
        profit=self._profit,
        action=action
    )
    self._update_history(info)

    # 다음 index로 넘어가기
    self._current_index += 1

    self._get_total_value()
    self._get_profit()
    self._observe()
    reward = self._get_reward()

    return self._state, reward, self._done, info
```

그림 33. step 메서드

수정 방향

그러나 작성된 환경으로 학습을 진행했을 때 두 가지 문제점이 발생했다.

1. 학습이 제대로 이루어지지 않았다.

학습한 모델의 투자수익이 만족스럽지가 않았다. 보상함수를 좀 더 구체적으로 수정함으로써 이를 해결하고자 한다.

2. 관망을 내놓는 경우가 많았다.

학습한 모델의 Action 이 관망인 경우가 너무 많았다. 관망하지 않았을 때의 보상을 조금 더 구체화하고 극대화하여, 모델로 하여금 관망을 선택하지 않게 유도하는 방식, 관망했을 때의 패널티를 주는 방식, 관망이라는 Action 을 없애고 다음 주가에 대한 Action 으로 action space 를 수정하는 방식으로 환경을 수정할 예정이다.

4-3. 서비스 개발

4-3-1. Front-end

메인 페이지에 대한 레이아웃이 완성되었으며, 상세페이지에 대한 구상이 완료되었다.

메인 페이지

서비스에서 제공하는 종목에 대한 개요를 담은 카드를 보여준다. 각 카드에는 로고, 종목이름, 종목코드, 가격, 변동률, AI의 추천도 요약을 벡지 형식으로 보여준다. 각 카드를 클릭할 경우 해당 종목에 대한 상세페이지를 볼 수 있다. 다양한 종목이 추가될 경우를 보여주기 위해 삼성전자는 이름으로 더미데이터를 넣어 둔 상태이다.

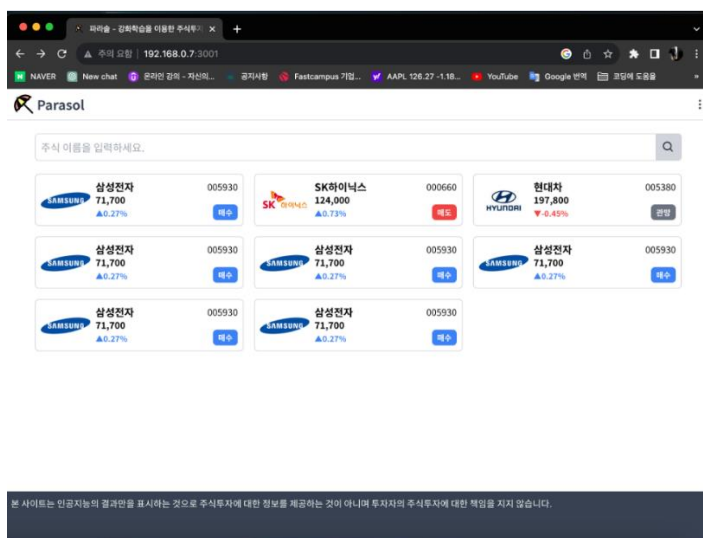


그림 34. pc형 메인 페이지

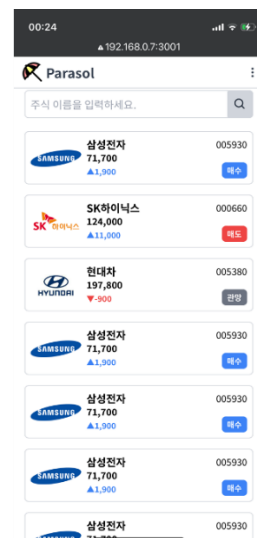


그림 35. 모바일형 메인페이지

상세 페이지

구상하고 있는 상세 페이지 레이아웃은 아래 그림과 같다. 해당 종목에 대한 AI가 추천하는 Action에 대한 정보, 최근 종목의 그래프, 최근 일자에 AI가 내놓은 추천, 크롤링한 최신 기사를 보여준다. 그래프는 현재 구현되지 않은 상태로 이미지를 크롤링하는 방식으로 만들어 놓았다. 그래프를 표현할 방법을 모색하는 중이다.



그림 36. pc 형 상세페이지

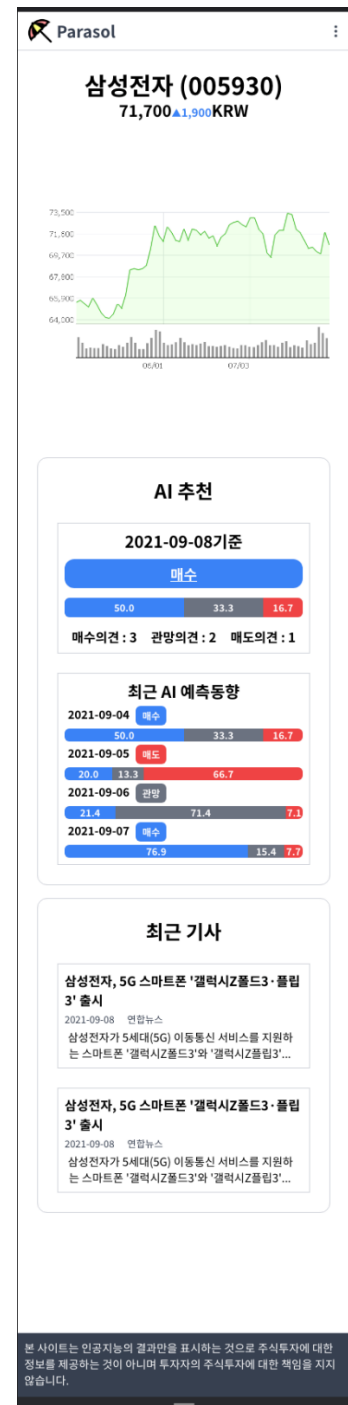


그림 37. 모바일형 상세페이지