강화학습 트레이딩 에이전트 개발

안성찬

목차

시작 전 고려사항

효율적 시장가설(EMH)에 따라 과거 정보를 이미 반영한 금융 데이터에서 강화학습이 패턴을 학습할 수 있는가?

효율적 시장가설(EMH)은 과거 가격이 이미 모든 정보를 반영한다고 가정하지만, 실제 시장에서는 단기적인 비효율성이 존재할 가능성이 크다. 시장이 항상 완벽하게 효율적이지는 않으며, AI가 비효율성이 발생하는 특정 상황(예: 공포 매도, 과매수, 기술적 오류 등)을 포착할 가능성이 있다고 가정한다.

금융 시장에서 agent의 action이 환경에 거의 영향을 주지 않는 상황은 강화학습에서 Non-Markovian 문제로 볼 수 있는가? 그렇다면 이를 어떻게 해결할 수 있는가?

금융 시장에서 agent의 행동이 환경 상태에 영향을 주지 않는 문제는 Non-Markovian 특성과 외생적 특성이 혼재된 복합적 환경으로 볼 수 있다.

이 문제를 해결하기 위해,
Sliding Window로 시계열 데이터를 구성하여 시장 참여자들의 행동 패턴을 학습한다.
Transformer와 같은 메모리 기반 모델을 사용해 시계열 데이터를 학습하고 과거 패턴에서 유의미한 정보를 추출한다.

강화학습 에이전트 개요

1. 데이터

2. 환경

3. 에이전트

Data

Binance에서 받은 15분 데이터 25만개.
5분 데이터 75만개

'기존의 시장 참여자들이 차트에 만들어 내는 패턴이나 경향이 존재한다'는 가설을 입증하기 위해, 가격 데이터만 사용.

Chart Data

15분 데이터의 Open, High, Low, Close로 구성됨.
이 데이터로 환경의 Step 진행하여 Reward를 계산
테스팅, 시각화를 하는데 필요.

Training Data

close data를 슬라이딩 윈도우로 구성하여 에이전트에 observation으로 전달할 데이터
각 행(시간)별로 정규화하여 가격의 패턴이나 변화만 포착하도록 함.

예를 들어 해당 시간의 스칼라 차트 데이터가 observation으로 전달되는 윈도우의 마지막 값으로 들어감. 해당 시간 이전의 데이터들을 보는 것.

Environment

1

미실현 손익을 포함한 잔고를 reward로

장점: 현재 포지션의 가치 변화를 실시간으로 반영

단점: 평가손익이 시장 변동성에 따라 크게 변동 -> 보상의 불안정
단기적인 평가손익을 최적화하려고 하여 비효율적인 거래를 생성함.
평가손익이 반영되지만, 실제로 실현되지 않을 수 있으므로 보상이 비현실적

Vs

2

실현 손익만 reward로

장점 : 실현된 손익에만 보상을 받으므로,
보상이 안정적이고 명확함.

단점 : 보상이 포지션 종료 시점에서만 계산되므로,
에이전트가 학습할 기회가 줄어들 수 있음.

Environment

Gym-traiding-env 환경이 구현되어 있지만, 코드 전체를 확실히 알기 힘들고, 미실현 손익을 포함한 잔고를 reward로 하는 단점이 있음.

학습이 되는 것을 잘 보기 위해서 영향을 받는 요소들을 줄이고, 1과 -1 action을 하고, 청산 시에 보상을 받는 환경을 새로 만듦.

https://github.com/ClementPerroud/Gym-Trading-Env

- 직접 만들어 봤지만, 포지션 청산에 따른 잔고 추적이 제대로 이뤄지지 않음.

```python
def step(self, action):
    """환경 단계"""
    # action을 -1부터 1까지의 범위로 변환
    action = (action - 10) / 10.0  # 0 ~ 20을 -1 ~ 1로 변환

    # 현재 Close 가격
    current_price = self.data[self.current_step]  # 현재 Close 가격

    # 포지션 크기 결정 (잔고 기반)
    position_change = self.balance * abs(action) / current_price  # BTC 단위

    # 행동에 따라 포지션 조정
    pnl = 0  # 초기 PnL
    if action > 0:  # 매수 (롱 포지션 증가)
        self._open_position(position_change, current_price)
    elif action < 0:  # 매도 (숏 포지션 증가)
        self._open_position(-position_change, current_price)
    elif action == 0 and self.current_position != 0:  # 포지션 청산
        pnl = self._close_position(current_price)

    # 포지션 감소 (부분 청산)
    if action * self.current_position < 0:  # 반대 방향의 행동 (부분 청산 발생)
        pnl = self._reduce_position(action, current_price)

    # 미실현 수익 업데이트
    self.unrealized_pnl = (
        self.current_position * (current_price - self.entry_price)
        if self.current_position != 0
        else 0
    )

    # 보상 계산: 포지션 청산 시 보상만 반영
    reward = pnl

    # 다음 단계로 이동
    self.current_step += 1
    done = self.current_step >= self.num_steps - 1

    obs = self._get_observation()

    return obs, reward, done, {}, self.current_position
```

- 학습에 집중하기 위해 문제가 없다고 확신할만한 간단한 환경을 직접 만들어 봄

→

```python
if action == 1:  # 롱 포지션 진입
    if self.position == -1:  # 숏 포지션 청산
        current_price = self.price_data.iloc[self.current_step, 0]
        reward = self.position * np.log(current_price / self.entry_price)  # 숏 청산 보상
        self.position = 0  # 포지션 없음으로 전환
        self.entry_price = None
    elif self.position == 0:  # 포지션 없음에서 롱 포지션 진입
        self.position = 1
        self.entry_price = self.price_data.iloc[self.current_step, 0]
elif action == -1:  # 숏 포지션 진입
    if self.position == 1:  # 롱 포지션 청산
        current_price = self.price_data.iloc[self.current_step, 0]
        reward = self.position * np.log(current_price / self.entry_price)  # 롱 청산 보상
        self.position = 0  # 포지션 없음으로 전환
        self.entry_price = None
    elif self.position == 0:  # 포지션 없음에서 숏 포지션 진입
        self.position = -1
        self.entry_price = self.price_data.iloc[self.current_step, 0]

self.cum_reward += reward
```

E2Env (Entry and Exit Environment)

포지션 관리:
롱(1), 숏(-1), 중립(0) 세 가지 포지션 상태로 구성.
진입과 청산 이후 포지션은 독립적으로 설정되며, 이전 상태의 영향을 받지 않음.

보상 계산:
Reward = 수익률 = 포지션 * (청산가격 − 진입가격) /진입가격, Entry Exit 할 때마다 수수료 0.002

학습 구조:
온라인 TD(시간차 학습) 방식으로 학습 염두.

간단하지만 제대로 학습할 수 있는 강화학습 환경 설계.

Agent 학습

Stable-Baselines3의 Proximal Policy Optimization (PPO)

PPO는 연속적, 이산적 액션 공간을 모두 지원

검증된 구현체를 통해 효율적으로 학습 환경을 설계

검증된 라이브러리를 사용함으로써 에이전트 설계의 복잡성을 줄이고 환경 개발, 학습 검증에 집중.

하이퍼파라미터 튜닝

learning_rate: 신경망 학습 속도를 조정하며, 너무 높으면 불안정, 너무 낮으면 느린 학습.

gamma: 미래 보상의 중요도를 설정, 금융 환경에서는 보통 0.99로 장기 보상을 강조.

clip_range: 정책 업데이트 범위를 제한해 학습 안정성을 높임, 일반적으로 0.1~0.3 사용.

ent_coef: 탐험 정도를 조정해 새로운 행동을 시도하도록 유도, 환경에 따라 10^-2, 10^-4 사용.

Reward 구조의 단점 개선

차트가 끝나면 다시 반복하여

에이전트가 학습할 기회가 줄어든다는 단점 개선

Policy and Value Networks

MLP

장점:

Fully Connected Layer로만 구성되어 구현이 간단하고 직관적.

계산량이 적고 학습 속도가 빠름.

단점:

시퀀스 데이터의 순서나 상관관계를 직접적으로 학습하지 못함.

각 입력을 독립적으로 처리하므로, 시퀀스 내 장기적 의존성을 학습하지 못함.

시계열 데이터에서 높은 성능을 기대하기 어려움.

Self-Attention 구조(Custom Policy)

Embedding → Positional Encoding → Encoder → FC Layer.

장점:

입력 시퀀스의 모든 시점 간 상관관계를 학습할 수 있음.

Positional Encoding을 통해 데이터의 순서 정보를 활용 가능.

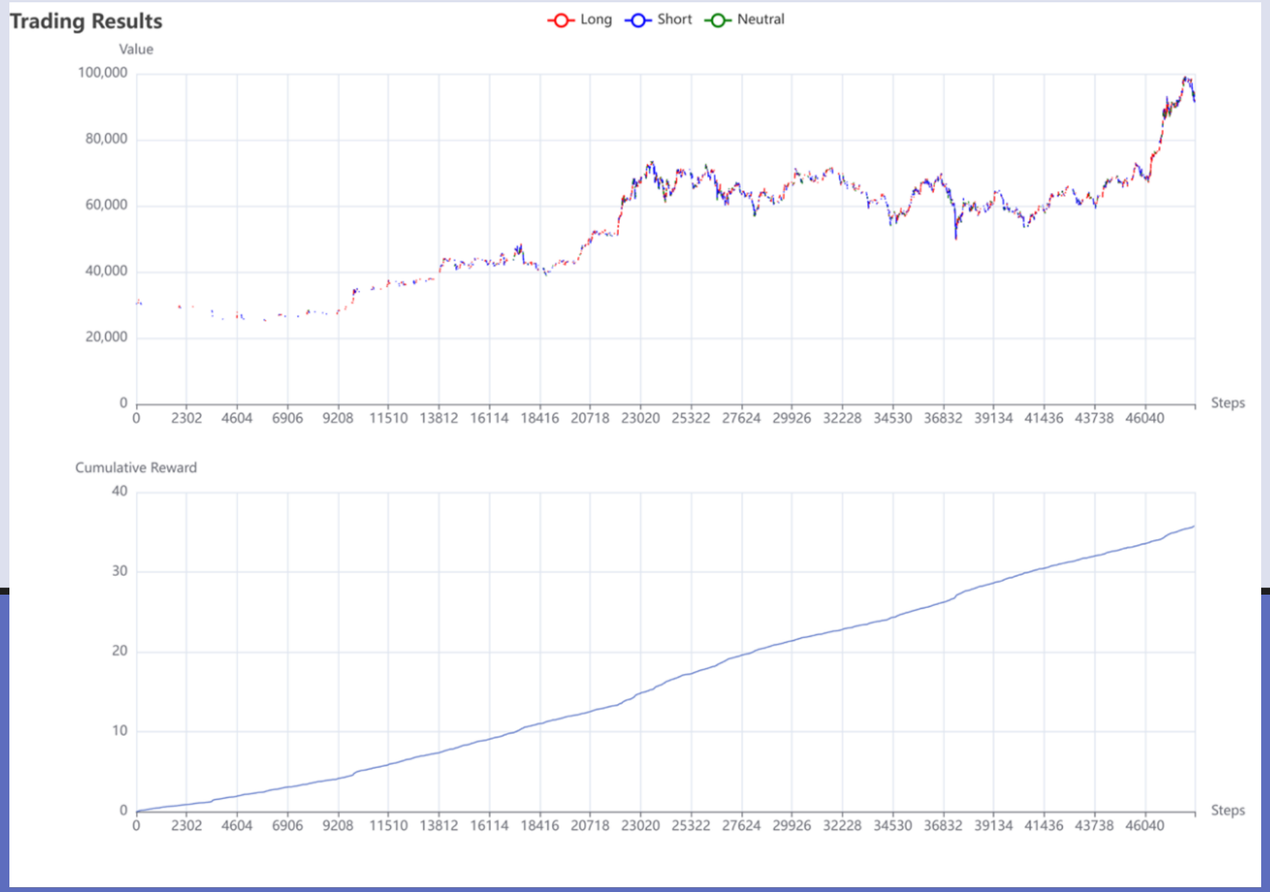
입력 시퀀스 전체를 고려한 학습이 가능

단점:

계산량이 많아, 학습 속도가 느림.

코드 구조와 하이퍼파라미터 튜닝이 복잡.

성능 평가


Trading Results
Long
Short
Neutral
Value
100,000
80,000
60,000
40,000
20,000
0
0 2302 4604 6906 9208 11510 13812 16114 18416 20718 23020 25322 27624 29926 32228 34530 36832 39134 41436 43738 46040
Steps
Cumulative Reward
40
30
20
10
0
0 2302 4604 6906 9208 11510 13812 16114 18416 20718 23020 25322 27624 29926 32228 34530 36832 39134 41436 43738 46040
Steps

성능 평가


Trading Results
Long
Short
Neutral
Value
100,000
80,000
60,000
40,000
20,000
0
0
2302
4604
6906
9208
11510
13812
16114
18416
20718
23020
25322
27624
29926
32228
34530
36832
39134
41436
43738
46040
Steps
Cumulative Reward
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-0.6
Steps

성능 평가


Trading Results
Long
Short
Neutral
Value
100,000
80,000
60,000
40,000
20,000
0
0 2302 4604 6906 9208 11510 13812 16114 18416 20718 23020 25322 27624 29926 32228 34530 36832 39134 41436 43738 46040
Steps
Cumulative Reward
0.8
0.6
0.4
0.2
0
-0.2
-0.4
0 2302 4604 6906 9208 11510 13812 16114 18416 20718 23020 25322 27624 29926 32228 34530 36832 39134 41436 43738 46040
Steps

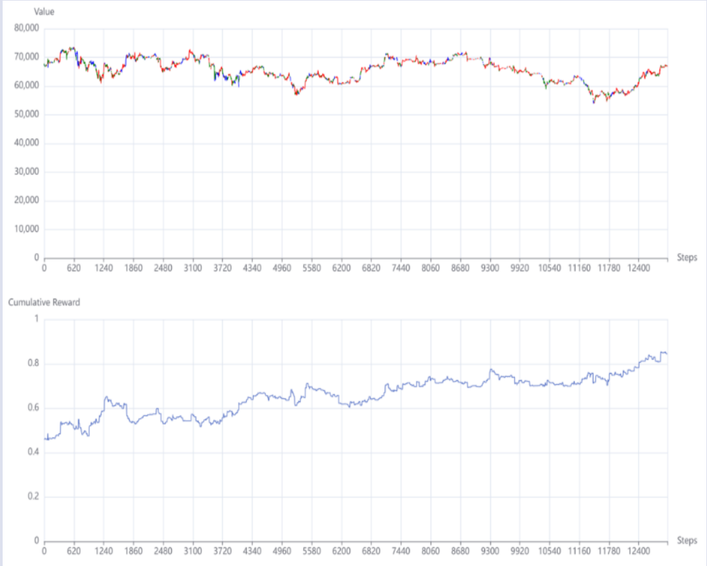
성능 평가


Trading Results
Long
Short
Neutral
Value
100,000
80,000
60,000
40,000
20,000
0
0
2302
4604
6906
9208
11510
13812
16114
18416
20718
23020
25322
27624
29926
32228
34530
36832
39134
41436
43738
46040
Steps
Cumulative Reward
0.8
0.6
0.4
0.2
0
-0.2
-0.4
0
2302
4604
6906
9208
11510
13812
16114
18416
20718
23020
25322
27624
29926
32228
34530
36832
39134
41436
43738
46040
Steps

성능 평가


Trading Results
Long
Short
Neutral
Value
100,000
80,000
60,000
40,000
20,000
0
0
2302
4604
6906
9208
11510
13812
16114
18416
20718
23020
25322
27624
29926
32228
34530
36832
39134
41436
43738
46040
Steps
Cumulative Reward
1.2
0.9
0.6
0.3
0
-0.3
0
2302
4604
6906
9208
11510
13812
16114
18416
20718
23020
25322
27624
29926
32228
34530
36832
39134
41436
43738
46040
Steps

성능 평가



시장 수익률은 떨어져도,
cumulative reward는 올라가는 모습

MultiAgentE3 (Entry and Exit Environment)

Hierarchical
Asynchronous
Ensemble

Method

멀티 에이전트 구조

데이터 준비:
각 윈도우 별(ex: 50, 150, 600, 2400) 데이터 분리 및 전처리.

환경 생성:
각 윈도우 크기에 대해 TradingEnv 생성 후 DummyVecEnv로 래핑.

병렬 학습:
multiprocessing.Pool을 이용해 각 윈도우 환경에서 PPO 모델 병렬 학습.

통합 환경 구성:
학습된 4개의 모델을 FinalTradingEnv로 통합.

최종 학습:
통합 환경에서 PPO로 Final 에이전트 추가 학습 진행.

755000 * 50,
755000 * 150,
755000 * 600,
755000 * 2400

총 2,416,000,000개의 데이터. 에피소드를 반복하면 더 늘어남.

현재 데이터와 환경의 규모로 인해 학습 과정에 집중해야 하므로, 테스트는 학습이 안정화된 이후로 계획을 조정

Policy and Value Networks

어려웠던 점

데이터, 환경, 에이전트, 학습 알고리즘, 네트워크 설계를 모두 파악해야 학습이 가능.

슬라이딩 윈도우로 입력 데이터를 생성하는 과정에서 차원과 크기 조정에 어려움.

롱/숏 포지션, 포지션 청산, 수수료 계산 등 금융 규칙을 반영하는 보상 체계 설계.

학습이 잘 안 됨
Entry와 Exit에 fee를 붙이면 에이전트가 0(유지, 관망)만 반환, 우상향하는 데이터에 fee를 빼면 1만 반환(95% 이상)

하이퍼파라미터 튜닝의 어려움. 학습률, 탐험률 등 값 설정에 따른 성능 변화를 반복 실험.
-> 데이터가 많아시간이 오래 걸림
학습 과정에서 발생한 문제가 어디서 일어났는 지 알기가 힘들었다.(데이터 누수, 보상설계)

배운 점

PPO, A2C 등 알고리즘 실험 및 하이퍼파라미터 튜닝의 복잡성 경험.
신경망 네트워크까지 연결하는 과정

하이퍼파라미터 설정이 모델 안정성에 미치는 영향을 이해.

실제 환경을 강화학습 환경으로 구현하는 방법과 노하우 습득

배운 점

보상

보상 체계를 설계하면서 보상의 방향성이 학습에 얼마나 큰 영향을 미치는지 알게 되었고, 잘못된 보상 체계가 에이전트를 오도할 수 있다는 것을 직접 경험했습니다. 예를 들어, 매수와 매도포지션의 전환 시 수익률 계산과 수수료 반영의 미세한 차이가 보상 신호를 크게 왜곡할 수 있었습니다.

끈기

학습이 제대로 되지 않을 때는 데이터, 환경, 네트워크, 알고리즘 등 여러 요소를 하나씩 점검하고 개선해야 했습니다. 이 과정에서 실수와 오류를 끊임없이 수정하며 작은 성공을 쌓아가는 과정의 중요성을 느꼈습니다. 또한, 문제를 단순히 "작동하도록" 만드는 것을 넘어, "효과적으로" 해결하기 위해 각 요소를 통합적으로 분석하고 개선하는 사고방식을 배웠습니다.

감사합니다.

https://github.com/ahnsc00/AbstractAlpha