


강화 학습과 시계열 예측을 이용한  
알고리즘 트레이딩

# ALGORITHMIC TRADING USING REINFORCEMENT LEARNING AND TIME SERIES PREDICTION



16기 박민규 엄기영 이영노  
17기 조성윤

# CONTENT



## **01. FinRL and Reinforcement Learning**

Brief review of FinRL and Reinforcement Learning

## **02. DLinear Prediction**

Explanation of DLinear prediction model

## **03. Prediction Applied Single Model**

Applying prediction dataset into the 'state' of agents

## **04. Ensemble with sharpe ratio**

Hard Vote the model with 'highest' sharpe ratio

# 01. FINRL AND REINFORCEMENT LEARNING

## 프로젝트 추진 배경

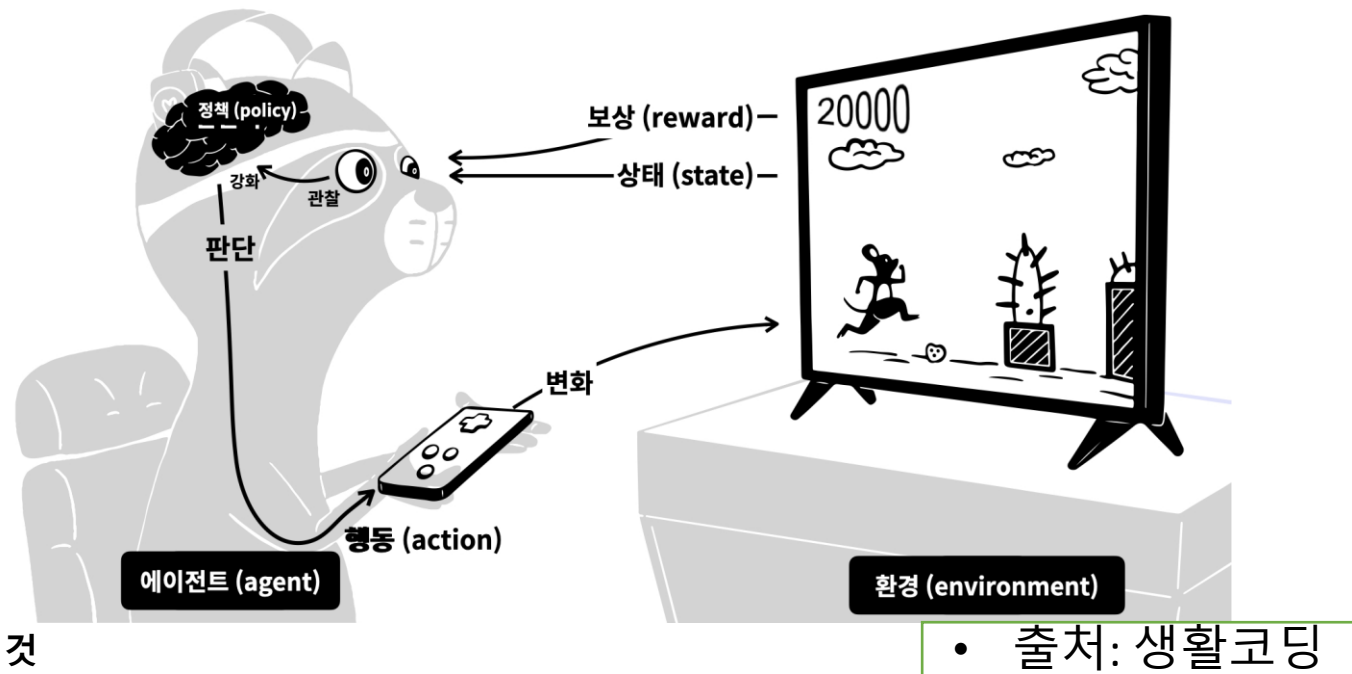
[Web 발신]  
(광고)  
쿠빅홀딩스  
▶ 오늘 무슨일이?!?  
3일 무료방 입장웹  
<https://github.com/MinkyuRamen/KubiqFinancialProject>

- 금융 하면 바로 떠오르는 **주식**. 시계열 예측으로서는 자주 다루어진 분야
- 지금까지 전혀 해보지 못했던 도전을 시도해보고자 했음.
- 알고리즘Trading을 **강화학습(Reinforcement Learning)** 으로 해보자.
- 쿠빅 최초?

# 01. FINRL AND REINFORCEMENT LEARNING

## WHAT IS REINFORCEMENT LEARNING?

- 구성: Environment(주변 환경) & Agent (행동의 주체)
- 학습 방식: 환경과 에이전트 사이의 지속적인 상호작용
- Data = **Trajectory** (상호 작용의 결과물)
  - Environment: State, Reward
  - Agent: Action
- Goal: 누적보상(cumulative reward)의 기댓값을 최대화하는 것
- 목표 달성을 위해 : Policy Improvement



$$\tau = \{S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, \dots, \}$$

# 01. FINRL AND REINFORCEMENT LEARNING

## RL IN STOCK MARKETS

➤ **Environment:** 주식 장 // **Agent:** 트레이더 -> 어떤 알고리즘이냐에 따라 구분!

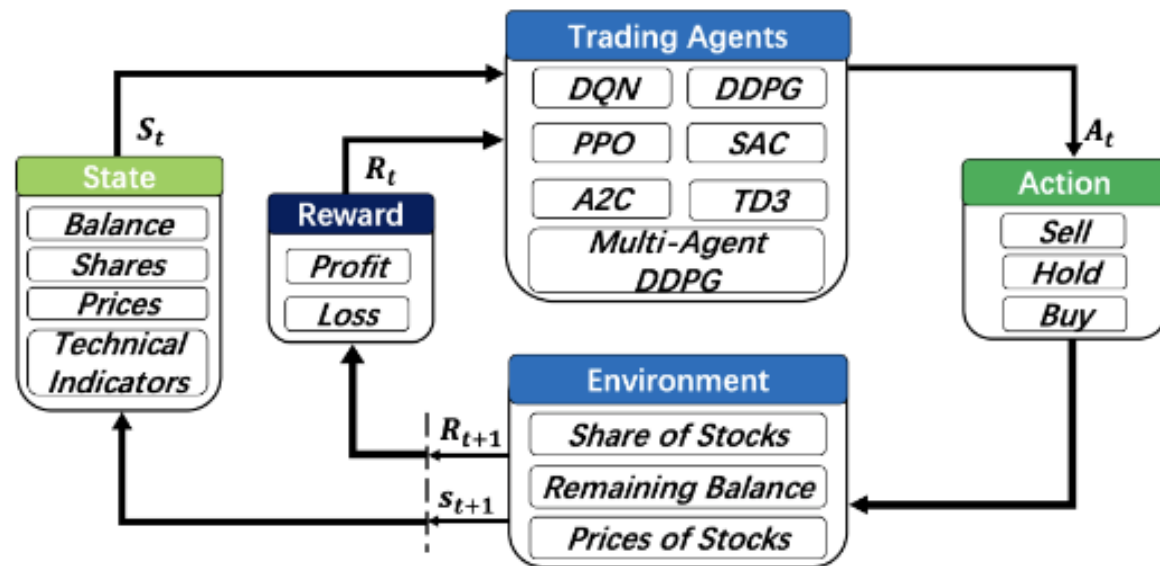
➤ **State:** 가격, 각종 지표, 포트폴리오 잔고 현황 등

➤ **Action:** 매수, 매도, 유지

$\{-k, \dots, -1, 0, 1, \dots, k\}$

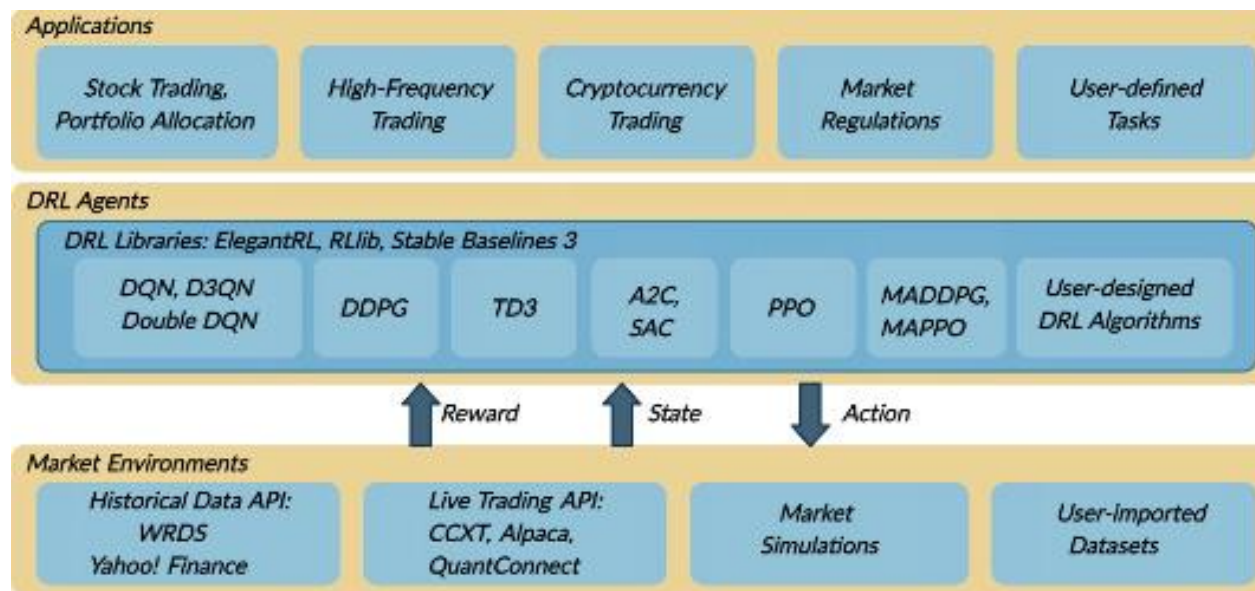
ex) +3 --> 3만큼 주식 portion을 가져간다.

➤ **목표:** 포트폴리오의 수익 극대화!



# 01. FINRL AND REINFORCEMENT LEARNING

## RL IN STOCK MARKETS



- Market Environment를 이용해서 DRL Agent 학습 및 여러 application 수행 가능한 패키지
- 주식 거래, 포트폴리오 최적화, 코인 거래 등 가능.

# 01. FINRL AND REINFORCEMENT LEARNING

## PROBLEM DEFINITION

- **구현 목표: 해당 포트폴리오의 수익률을 최대화할 수 있는 강화학습 기반의 안정적 장기투자 전략 구현**  
독창적인 Trading Strategy!
- 사용된 훈련 데이터셋: **미국 ETF Universe Data Tickers**  
\*안정적인 포트폴리오 자산군. 분산투자로 인한 ETF 자산군 간의 낮은 상관계수
- **DLinear + Prediction** 정보 추가
- FinRL 기반의 Actor-Critic Multi-agent: A2C, DDPG, PPO, TD3, SAC  
--> Voting 기반 **Ensemble 진행** -> select Best Agent!!
- TRAIN\_START\_DATE = '2010-01-01'  
➤ TRAIN\_END\_DATE = '2021-10-01'  
➤ TRADE\_START\_DATE = '2021-10-01'  
➤ TRADE\_END\_DATE = '2023-05-03'
- 성과지표: **Maximize Total Reward( & Sharpe Ratio)**
- Benchmark Comparison: 다우존스, 나스닥 주가 지수

| ETF 유니버스 |        |
|----------|--------|
| Ticker   | Sector |
| XLB      | 소재     |
| XLE      | 에너지    |
| XLF      | 금융     |
| XLI      | 산업재    |
| XLK      | IT     |
| XLP      | 필수소비재  |
| XLU      | 유틸리티   |
| XLV      | 헬스케어   |
| XLY      | 임의소비재  |

$$\text{Maximize: } SR_p = \frac{\mu_p - r_f}{\sigma_p} = \frac{\mathbf{w}^T \mathbf{R}}{\sqrt{\mathbf{w}^T \mathbf{\Sigma} \mathbf{w}}}$$

## 02. DLINEAR PREDICTION

### WHY DO WE NEED DLINEAR?

#### 1 INDICATORS

```
['macd',  
'boll_ub',  
'boll_lb',  
'rsi_30',  
'cci_30',  
'dx_30',  
'close_30_sma',  
'close_60_sma']
```

다른 알고리즘 트레이딩 방식들과 마찬가지로,

FinRL Baseline 역시 **INDICATORS(보조지표)**를 통해 투자 포지션을 결정 (buy/sell/hold) 한다.

- 하지만 FinRL의 INDICATORS는 macd(이동평균수렴확산), sma(이동평균), rsi(상대강도), boll(볼린저밴드) 등 모두 후행지표밖에 없다.  
(쉽게 말하면 주식가격 결정 이후에 나오는 정보)
- 현존하는 알고리즘 지표기반 트레이딩의 한계점 중 하나.



알고리즘 트레이딩에서 사용되 INDICATORS(조지표)



## 02. DLINEAR PREDICTION

### WHY DO WE NEED DLINEAR?

Time Series Forecast SOTA 모델을 이용한 prediction 값을 선행지표로 쓰기로 결정

금융 도메인에선 Nlinear보다 Dlinear의 성능이 보편적으로 우수 > Nlinear가 아닌 **Dlinear 선택**

| Methods     |     | IMP.   | Linear*      |              | NLinear* |              | DLinear*     |              | FEDformer    |              | Autoformer |       | Informer |       |
|-------------|-----|--------|--------------|--------------|----------|--------------|--------------|--------------|--------------|--------------|------------|-------|----------|-------|
| Metric      |     | MSE    | MSE          | MAE          | MSE      | MAE          | MSE          | MAE          | MSE          | MAE          | MSE        | MAE   | MSE      | MAE   |
| Electricity | 96  | 27.40% | <b>0.140</b> | <b>0.237</b> | 0.141    | <b>0.237</b> | <b>0.140</b> | <b>0.237</b> | <u>0.193</u> | <u>0.308</u> | 0.201      | 0.317 | 0.274    | 0.368 |
|             | 192 | 23.88% | <b>0.153</b> | 0.250        | 0.154    | <b>0.248</b> | <b>0.153</b> | 0.249        | <u>0.201</u> | <u>0.315</u> | 0.222      | 0.334 | 0.296    | 0.386 |
|             | 336 | 21.02% | <b>0.169</b> | 0.268        | 0.171    | <b>0.265</b> | <b>0.169</b> | 0.267        | <u>0.214</u> | <u>0.329</u> | 0.231      | 0.338 | 0.300    | 0.394 |
|             | 720 | 17.47% | <b>0.203</b> | 0.301        | 0.210    | <b>0.297</b> | <b>0.203</b> | 0.301        | <u>0.246</u> | <u>0.355</u> | 0.254      | 0.361 | 0.373    | 0.439 |
| Exchange    | 96  | 45.27% | 0.082        | 0.207        | 0.089    | 0.208        | <b>0.081</b> | 0.203        | <u>0.148</u> | <u>0.278</u> | 0.197      | 0.323 | 0.847    | 0.752 |
|             | 192 | 42.06% | 0.167        | 0.304        | 0.180    | 0.300        | <b>0.157</b> | 0.293        | <u>0.271</u> | <u>0.380</u> | 0.300      | 0.369 | 1.204    | 0.895 |
|             | 336 | 33.69% | 0.328        | 0.432        | 0.331    | 0.415        | <b>0.305</b> | 0.414        | <u>0.460</u> | <u>0.500</u> | 0.509      | 0.524 | 1.672    | 1.036 |
|             | 720 | 46.19% | 0.964        | 0.750        | 1.033    | 0.780        | <b>0.643</b> | <b>0.601</b> | <u>1.195</u> | <u>0.841</u> | 1.447      | 0.941 | 2.478    | 1.310 |

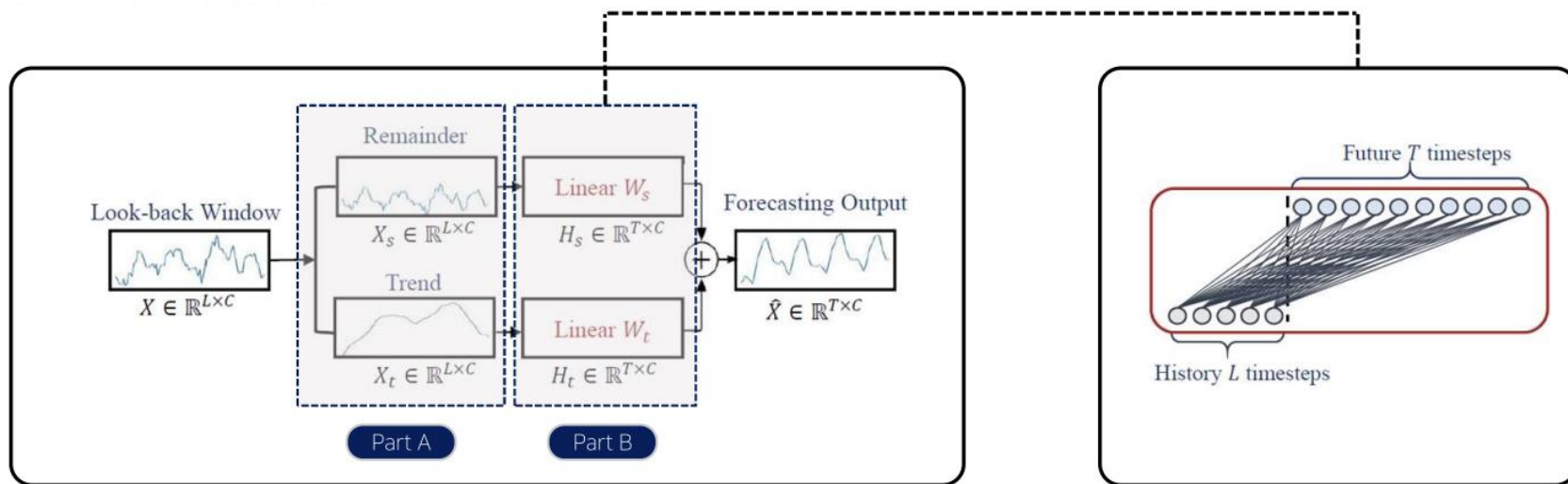
복잡한 Transformer 기반의 Forecasting 모델(Informer, Autoformer, FEDformer) 보다 간단한 Linear-layer 기반의 Forecasting 모델(Nlinear, Dlinear)의 성능이 더 좋다.

## 02. DLINEAR PREDICTION

### DLINEAR STRUCTURE

Dlinear 는 크게 Decomposition Part(Part A), layer linear network(Part B) 그리고 Forecasting Output로 나눌 수 있다.

Part A에선 moving average를 활용하여 **trend와 seasonality** 로 **decompose**,  
Part B에선 decomposition한 trend와 seasonality에 **1-layer linear network**를 **apply**,  
Forecasting output에선 1-layer linear를 거친 **trend와 seasonality**를 **add**.



## 02. DLINEAR PREDICTION

### DLINER STRUCTURE

Dlinear 의 구조는 매우 단순하기 때문에 메인 코드도 **40줄** 밖에 되지 않는다.  
200 epochs를 돌리는데 시간도 **3분** 밖에 소요되지 않는다. -> 알고리즘 트레이딩에 적절

```

27 class LTFS_DLinear(torch.nn.Module):
28
29     def __init__(self, window_size, forecast_size, kernel_size, individual, feature_size):
30         super(LTFS_DLinear, self).__init__()
31         self.window_size = window_size
32         self.forecast_size = forecast_size
33         self.decomposition = series_decomp(kernel_size)
34         self.individual = individual
35         self.channels = feature_size
36         if self.individual:
37             self.Linear_Seasonal = torch.nn.ModuleList()
38             self.Linear_Trend = torch.nn.ModuleList()
39             for i in range(self.channels):
40                 self.Linear_Trend.append(torch.nn.Linear(self.window_size, self.forecast_size))
41                 self.Linear_Trend[i].weight = torch.nn.Parameter((1/self.window_size)*torch.ones([self.forecast_size, self.window_size]))
42                 self.Linear_Seasonal.append(torch.nn.Linear(self.window_size, self.forecast_size))
43                 self.Linear_Seasonal[i].weight = torch.nn.Parameter((1/self.window_size)*torch.ones([self.forecast_size, self.window_size]))
44         else:
45             self.Linear_Trend = torch.nn.Linear(self.window_size, self.forecast_size)
46             self.Linear_Trend.weight = torch.nn.Parameter((1/self.window_size)*torch.ones([self.forecast_size, self.window_size]))
47             self.Linear_Seasonal = torch.nn.Linear(self.window_size, self.forecast_size)
48             self.Linear_Seasonal.weight = torch.nn.Parameter((1/self.window_size)*torch.ones([self.forecast_size, self.window_size]))
49
50     def forward(self, x):
51         trend_init, seasonal_init = self.decomposition(x)
52         trend_init, seasonal_init = trend_init.permute(0,2,1), seasonal_init.permute(0,2,1)
53         if self.individual:
54             trend_output = torch.zeros([trend_init.size(0), trend_init.size(1), self.forecast_size], dtype=trend_init.dtype).to(trend_init.device)
55             seasonal_output = torch.zeros([seasonal_init.size(0), seasonal_init.size(1), self.forecast_size], dtype=seasonal_init.dtype).to(seasonal_init.device)
56             for idx in range(self.channels):
57                 trend_output[:, idx, :] = self.Linear_Trend[idx](trend_init[:, idx, :])
58                 seasonal_output[:, idx, :] = self.Linear_Seasonal[idx](seasonal_init[:, idx, :])
59         else:
60             trend_output = self.Linear_Trend(trend_init)
61             seasonal_output = self.Linear_Seasonal(seasonal_init)
62         x = seasonal_output + trend_output
63         return x.permute(0,2,1)

```

✓  
3분

[20] 1 model = train(model, train\_loader, criterion, optimizer, device, 200, 1)

|      |   |                           |
|------|---|---------------------------|
| 0%   | 1/200 [00:01<04:58, 1.50s/it]Epoch: 1     | Training Loss: 48.150379  |
| 6%   | 11/200 [00:15<03:07, 1.01it/s]Epoch: 11   | Training Loss: 186.467575 |
| 10%  | 21/200 [00:24<02:45, 1.08it/s]Epoch: 21   | Training Loss: 163.137222 |
| 16%  | 31/200 [00:34<02:53, 1.03s/it]Epoch: 31   | Training Loss: 121.302902 |
| 20%  | 41/200 [00:42<02:17, 1.15it/s]Epoch: 41   | Training Loss: 103.253685 |
| 26%  | 51/200 [00:52<02:12, 1.13it/s]Epoch: 51   | Training Loss: 105.056389 |
| 30%  | 61/200 [01:03<02:23, 1.04s/it]Epoch: 61   | Training Loss: 133.377167 |
| 36%  | 71/200 [01:13<02:19, 1.08s/it]Epoch: 71   | Training Loss: 153.780731 |
| 40%  | 81/200 [01:22<01:43, 1.15it/s]Epoch: 81   | Training Loss: 140.970062 |
| 46%  | 91/200 [01:32<01:40, 1.08it/s]Epoch: 91   | Training Loss: 90.491035  |
| 50%  | 101/200 [01:41<01:38, 1.01it/s]Epoch: 101 | Training Loss: 94.355911  |
| 56%  | 111/200 [01:51<01:26, 1.03it/s]Epoch: 111 | Training Loss: 143.144043 |
| 60%  | 121/200 [02:00<01:10, 1.12it/s]Epoch: 121 | Training Loss: 110.194191 |
| 66%  | 131/200 [02:10<01:03, 1.09it/s]Epoch: 131 | Training Loss: 110.310150 |
| 70%  | 141/200 [02:19<01:03, 1.08s/it]Epoch: 141 | Training Loss: 142.625885 |
| 76%  | 151/200 [02:28<00:43, 1.12it/s]Epoch: 151 | Training Loss: 119.909546 |
| 80%  | 161/200 [02:38<00:35, 1.09it/s]Epoch: 161 | Training Loss: 112.236832 |
| 86%  | 171/200 [02:48<00:29, 1.00s/it]Epoch: 171 | Training Loss: 103.230537 |
| 90%  | 181/200 [02:57<00:17, 1.08it/s]Epoch: 181 | Training Loss: 122.718674 |
| 96%  | 191/200 [03:06<00:07, 1.13it/s]Epoch: 191 | Training Loss: 139.167847 |
| 100% | 200/200 [03:15<00:00, 1.02it/s]           |                           |

Applying prediction dataset into the 'state' of agents

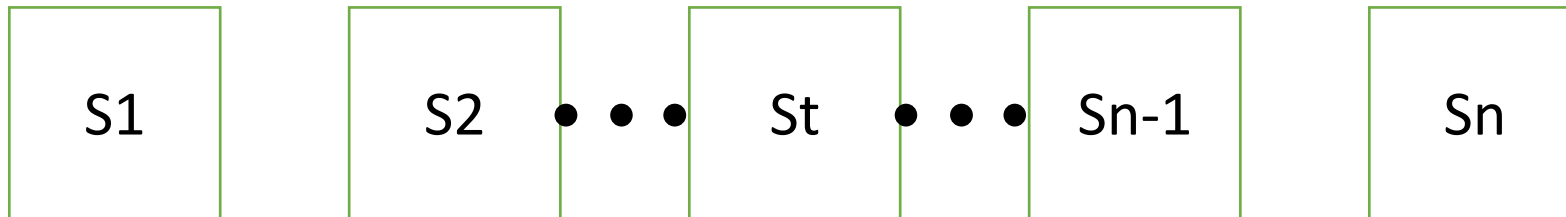
## 03. PREDICTION APPLIED SINGLE MODEL

### IDEA OF PREDICTION

강화학습 역시 데이터의 본질은 **시계열성을 띄고 있음**.

즉 **사전관찰** (현재 시점에서 미래의 정보를 확인하는 것)은 **Cheating**이 될 수 있으므로 **지양**해야.

➤ 그렇다면 어떻게 Prediction Data를 만들고, 각 시점에 집어넣어줄 수 있을까?

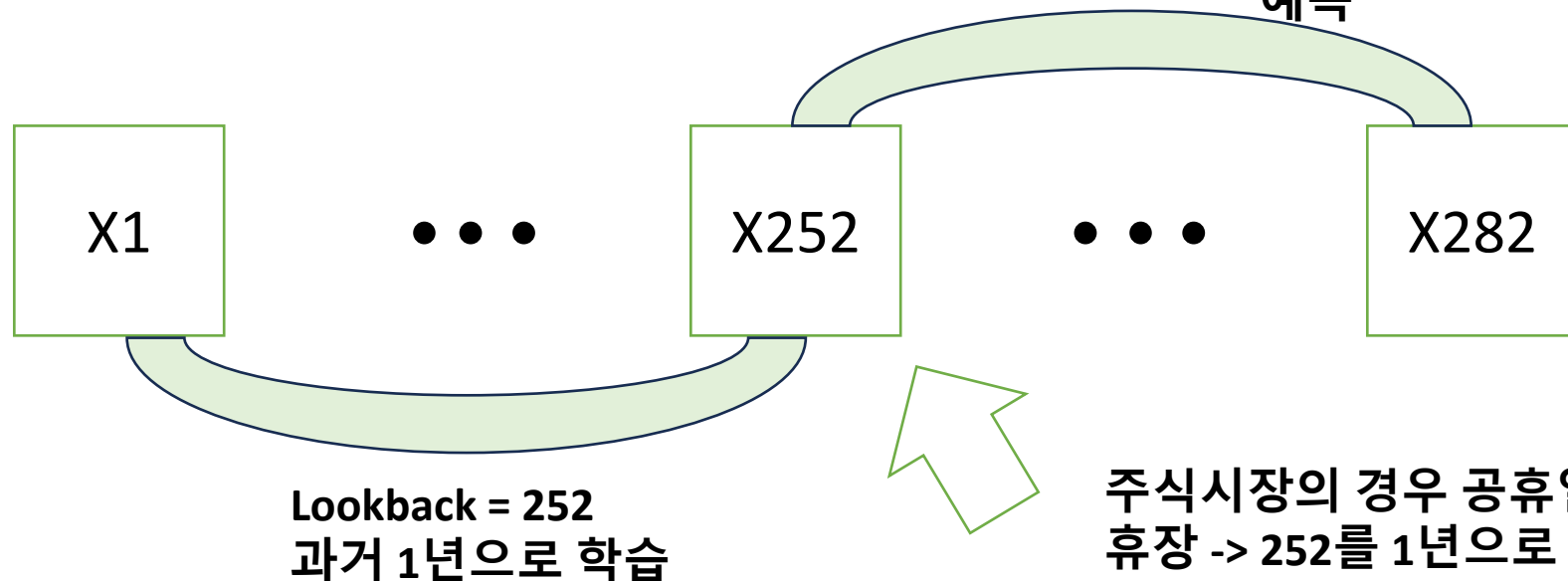


Applying prediction dataset into the 'state' of agents

## 03. PREDICTION APPLIED SINGLE MODEL

### IDEA OF PREDICTION

- 각 시점의 1년전 데이터 ~ 현재 데이터를 학습.  
EX) 현재 시점  $t = 252$ 라면  $t = 1 \sim 252$  까지의 데이터를 학습.
- 30일치의 prediction을 당시시점으로 예측.  
EX)  $t = 252$ 이면  $t = 282$  까지를 예측.



Applying prediction dataset into the 'state' of agents

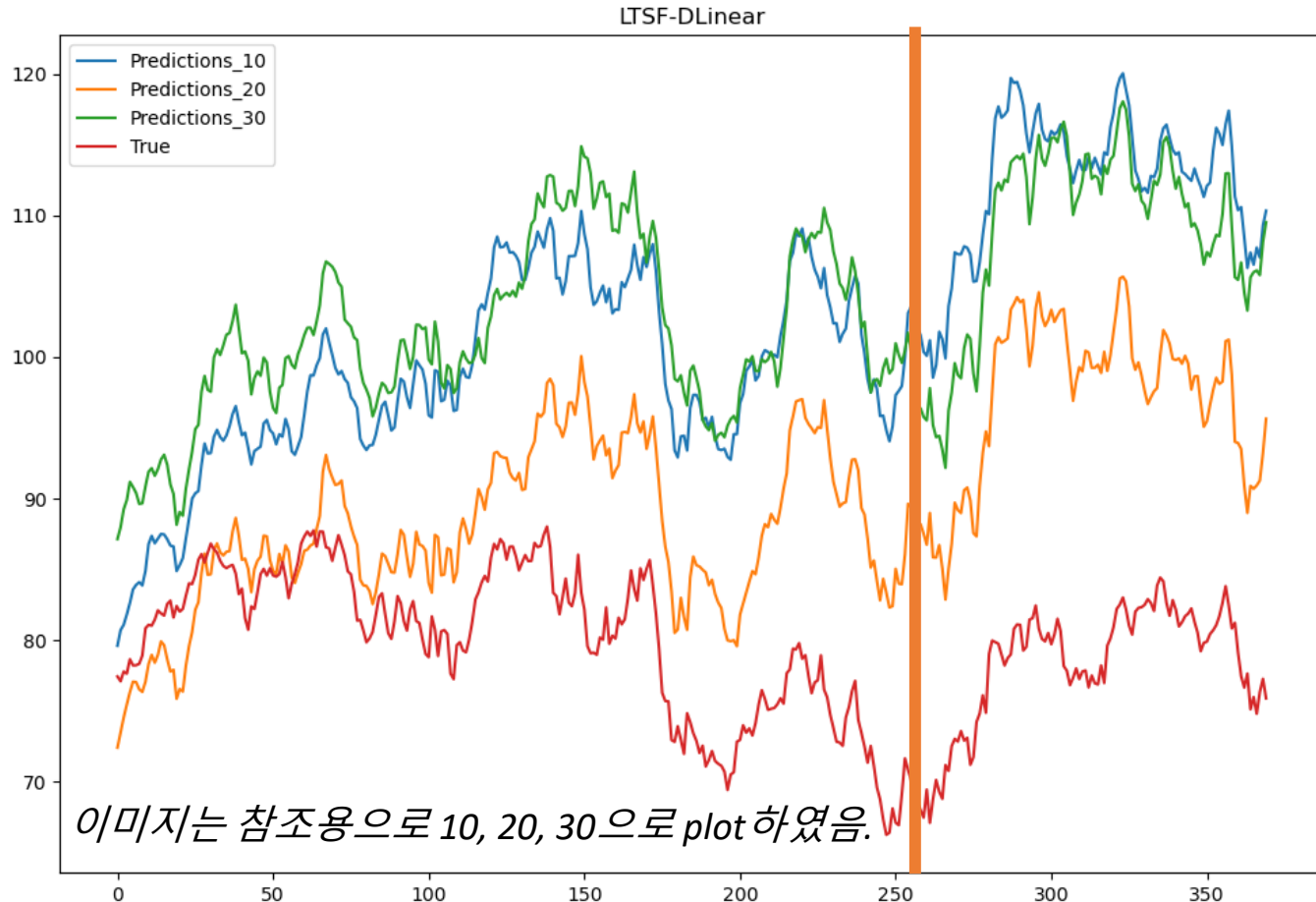
## 03. PREDICTION APPLIED SINGLE MODEL

### PREDICTION RESULT

- 30일 치의 모든 **prediction** 데이터를 주기에는 어려움이 있음.
- **5, 10, 30** 시차 뒤의 prediction만 주기로 결정.

붉은색이 실제 증가, prediction\_n은 해당 시점기준 과거 1년을 통해 학습한 예측모델이 **n시점** 뒤의 값을 예측한 값.

예를 들어, 오른쪽 테두리친  $t = 252$  시점의 선에서 predictions\_10은 262번째 시점을 예측한 값.

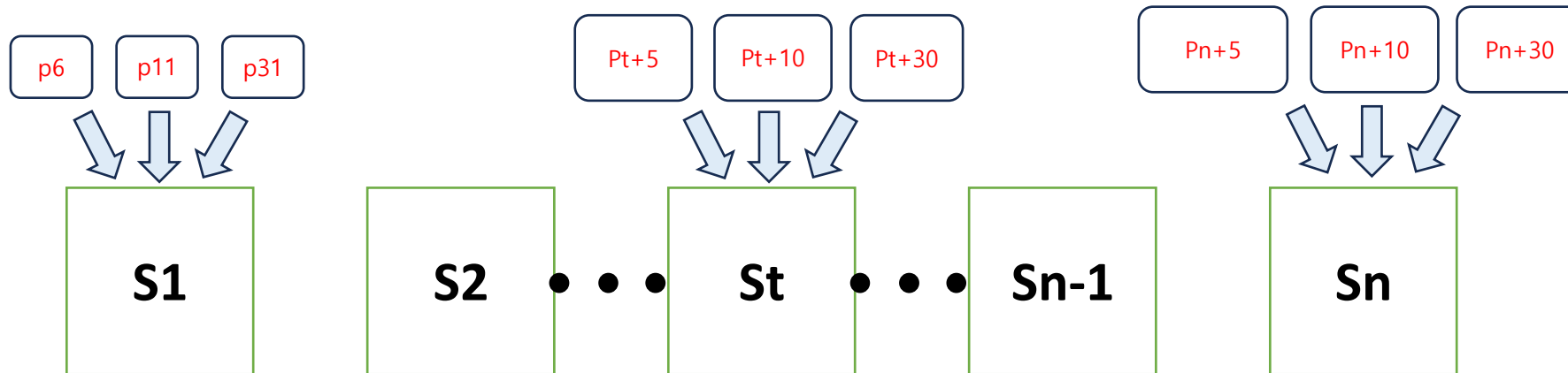


Applying prediction dataset into the 'state' of agents

## 03. PREDICTION APPLIED SINGLE MODEL

### APPLYING INTO THE FINRL

강화학습에 입력되는 각 'State'에 5, 10, 30차시의 증가 예측 결과를 추가로 입력.

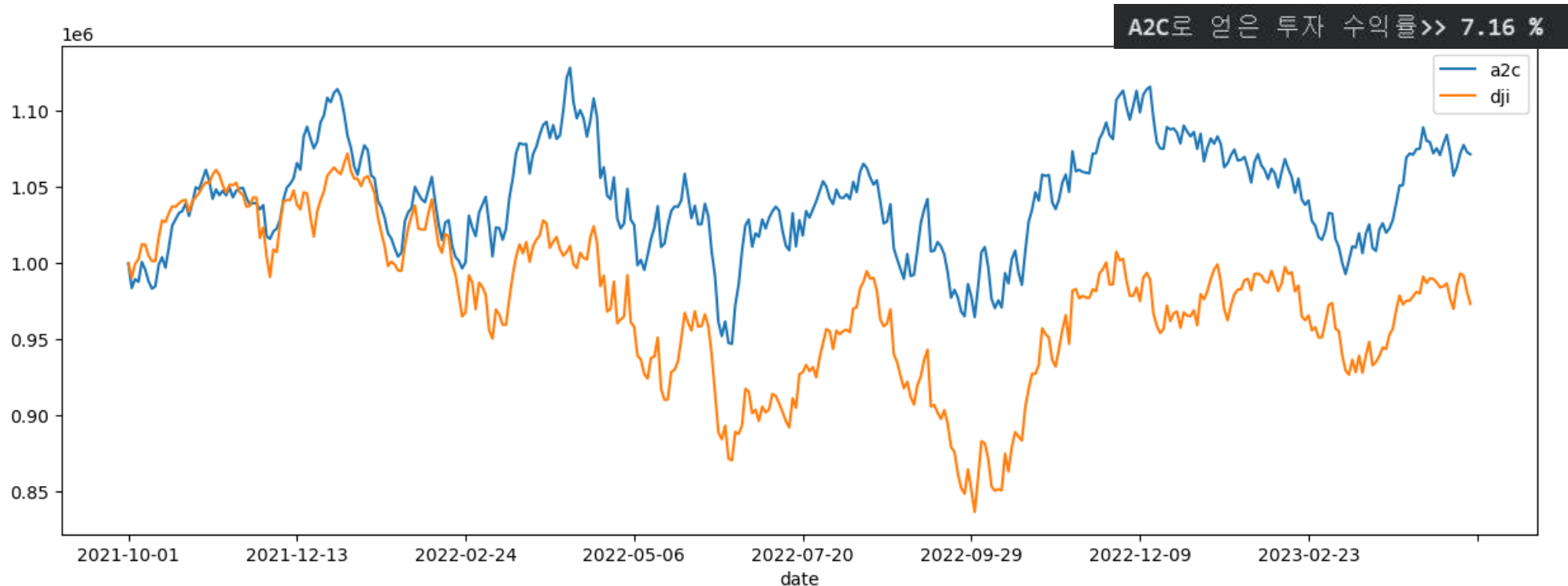


Applying prediction dataset into the 'state' of agents

## 03. PREDICTION APPLIED SINGLE MODEL

### APPLYING INTO THE FINRL

최고 성능을 얻고자함이 아닌, Prediction 데이터 추가가 성능향상에 영향을 주느냐를 분석하기 위함이므로, FinRL, SB3 에서 제공하는 A2C 모델만을 이용해서 실험을 진행. 노란색 그림은 일반 다우존스 지수. **A2C를 이용한 거래가 훨씬 효과적인 성능을 보여줌.**





Applying prediction dataset into the 'state' of agents

## 03. PREDICTION APPLIED SINGLE MODEL

### APPLYING INTO THE FINRL

결과 비교 각각 동일조건에서 Prediction Data를 추가한 경우, 추가하지 않은 경우를 비교. (10회반복)

|   | 수익률    |
|---|--------|
| 0 | -6.43  |
| 1 | -10.46 |
| 2 | 2.28   |
| 3 | -0.51  |
| 4 | 9.06   |
| 5 | 8.84   |
| 6 | -0.47  |
| 7 | 9.76   |
| 8 | -0.60  |
| 9 | 12.34  |

#### With Prediction

평균 수익률>> 2.38 %  
표준편차>> 7.13 %



#### Without Prediction

평균 수익률>> -7.15 %  
표준편차>> 6.95 %

예측값 이용한 경우의 결과가 우수하게 드러남.  
장이 좋지 않았음에도, 꽤나 높은 평균 수익률을 보여주었다.

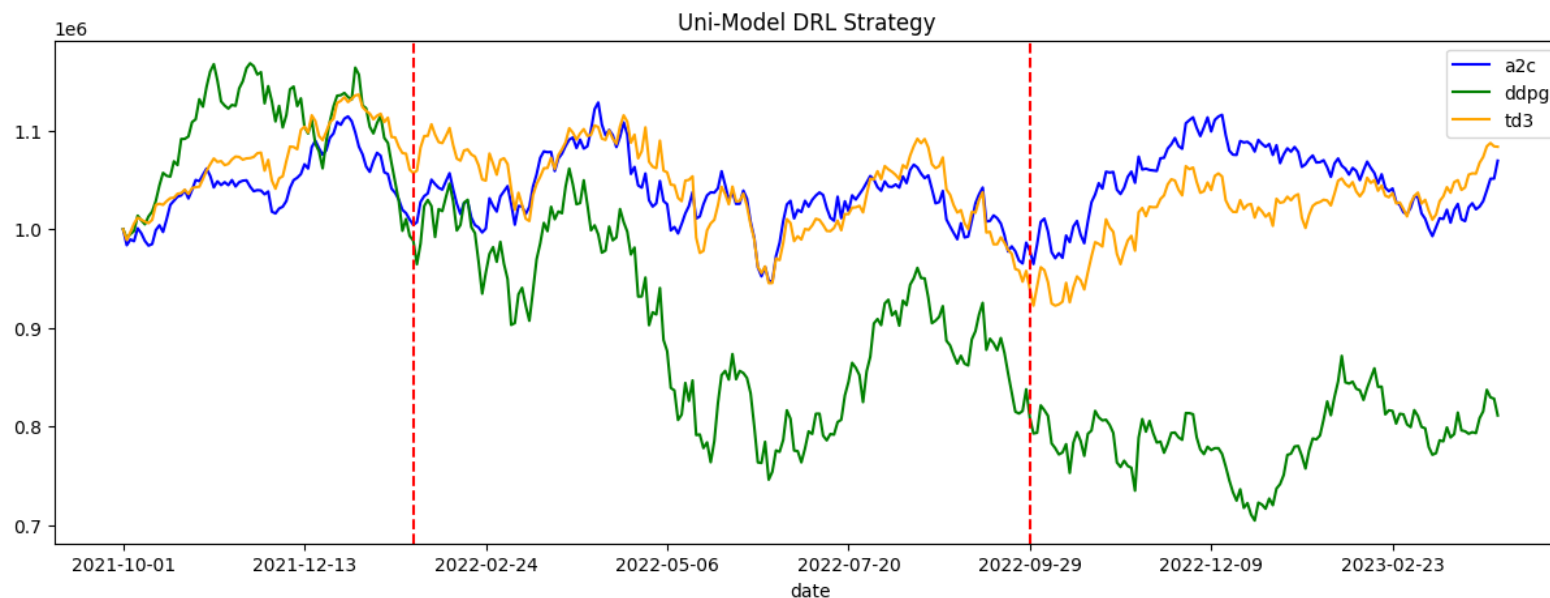
|   | 수익률    |
|---|--------|
| 0 | -13.42 |
| 1 | 0.10   |
| 2 | 4.20   |
| 3 | -8.26  |
| 4 | -1.83  |
| 5 | -13.84 |
| 6 | -6.77  |
| 7 | -11.94 |
| 8 | -1.17  |
| 9 | -18.53 |

Hard Vote the model with 'highest' sharpe ratio

## 04. ENSEMBLE WITH SHARPE RATIO

### PROBLEM DEFINITION

- 구간별로 최대 Reward 를 내는 model이 다름.
  - 단일 모델의 Reward Volatility high.
- 구간별로 **highest reward model** 을 선택하면, 단일 모델 대비 reward 가 개선되지 않을까 ?
- High Volatility에 **Penalty**를 부여할 순 없을까 ?



Hard Vote the model with 'highest' sharpe ratio

## 04. ENSEMBLE WITH SHARPE RATIO

### SOLUTION : ENSEMBLE WITH SHARPE RATIO

- 구간별로 최대 Reward 를 내는 model이 다름.
  - 단일 모델의 Reward Volatility high.
- 구간별로 highest reward model 을 hard vote.
- 안정적 장기투자 관점에서 return volatility에 **Penalty**. 수익의 급격한 변동을 방지. **Risk**를 낮추는 전략.

|   | Iter | Val Start  | Val End    | Model Used | DDPG Sharpe | SAC Sharpe | TD3 Sharpe |
|---|------|------------|------------|------------|-------------|------------|------------|
| 0 | 126  | 2021-10-04 | 2022-01-03 | SAC        | 0.248094    | 0.413804   | 0.388397   |
| 1 | 189  | 2022-01-03 | 2022-04-04 | SAC        | -0.348656   | -0.125916  | -0.282326  |
| 2 | 252  | 2022-04-04 | 2022-07-06 | SAC        | -0.540129   | -0.492693  | -0.534592  |
| 3 | 315  | 2022-07-06 | 2022-10-04 | DDPG       | 0.01164     | -0.031788  | -0.112318  |
| 4 | 378  | 2022-10-04 | 2023-01-04 | DDPG       | 0.08509     | -0.121076  | -0.081355  |

$$S = \left( \frac{R_p - R_f}{\sigma_p} \right)$$

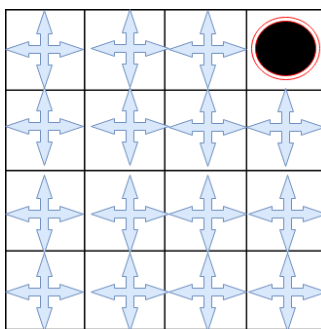
```
if df_total_value["daily_return"].std() != 0:  
    sharpe = (  
        (252**0.5)  
        * df_total_value["daily_return"].mean()  
        / df_total_value["daily_return"].std()  
    )
```

Hard Vote the model with 'highest' sharpe ratio

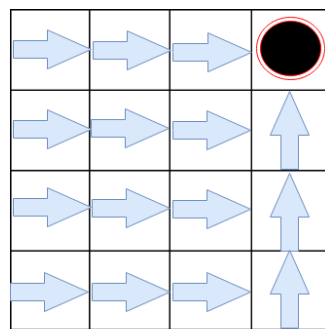
# 04. ENSEMBLE WITH SHARPE RATIO

## 3 STRATEGIES

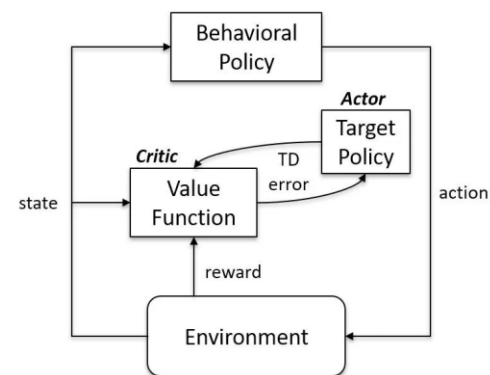
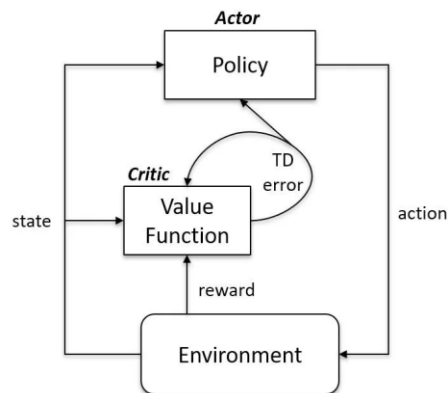
- Terms
  - Behavioral policy** (행동 정책) : Agent 가 action 을 선택할때 사용하는 policy (**sampling**)
  - Target policy** (목표 정책) : 최적의 action function 을 학습하기 위해 update 하는 policy (**update**)
- On-Policy** : Behavioral policy = Target Policy  
 $Q(s, \mathbf{a}')$  는 가장 최근 action sampling ( $\mathbf{a}$ ) 에 사용된 behavioral policy  $\pi(\mathbf{a}'|s)$  의 action ( $\mathbf{a}'$ ) 으로부터 학습
- Off-Policy** : Behavioral policy  $\neq$  Target Policy  
 $Q(s, \mathbf{a}')$  는 target policy  $\pi(\mathbf{a}'|s)$  와 상이한 behavioral policy  $\Omega(\mathbf{a}'|s)$  으로부터 학습



Behavior Policy



Target Policy



Hard Vote the model with 'highest' sharpe ratio

## 04. ENSEMBLE WITH SHARPE RATIO

### 3 STRATEGIES

| All_Ensemble  | On-Policy Algorithms  | Off-Policy Algorithms  |
|---|---|--|
| <ul style="list-style-type: none"><li>A2C, PPO, DDPG, SAC, TD3</li><li>학습 효율성은 저해되지만 최적의 모델 고려 가능</li></ul> | <ul style="list-style-type: none"><li>A2C, PPO</li><li>Target Policy = Behavioral Policy 에 따른 <b>안정적인 학습</b> 학습과정에서 큰 변동성 x</li><li>Action sampling시 target policy에 따라 데이터 수집하기 때문에 optimal policy 에 <b>더 쉽게 수렴</b>.</li><li>✓ Exploration 의 제약</li><li>✓ Action Sampling Bias : 데이터 수집이 현재정책에 의존</li></ul> | <ul style="list-style-type: none"><li>DDPG, SAC, TD3</li><li>Exploration 을 하면서도 Optimal Policy Update. <b>더 많은 Exploration 보장</b></li><li>Replay Buffer : Agent가 경험한 데이터를 저장하여 차후 필요할때 sampling <b>Sampling Efficiency</b></li><li>✓ Convergence</li><li>✓ 과거 수집 데이터에 의존</li></ul> |

Hard Vote the model with 'highest' sharpe ratio

## 04. ENSEMBLE WITH SHARPE RATIO

### IMPLEMENTATION

- Environment : `class StockTradingEnv(gym.env)`
- Agent : `class DRLEnsembleAgent()`  
def `DRL_prediction()` :  
    **Action, \_states = model.predict(trade\_obs)**  
    **Trade\_obs, rewards, ... = trade\_env.step(action)**  
    # **model.predict** 으로 Agent 가 선택한 **action** 반환  
    # **trade\_env.step(action)** 으로 Environment 에서  
    # given action 에 대한 **next state, reward** 반환  
  
def `run_ensemble_strategy_{'OurStrategy'}_ :`  
    Train, Validation 진행  
    Validation : 구간별 sharpe ratio 가 최대인 model 선택  
    선택된 model 은 다음구간 전략으로 사용됨.

```
def DRL_prediction(  
    self, model, name, last_state, iter_num, turbulence_threshold, initial  
):  
    """make a prediction based on trained model"""  
  
    trade_obs = trade_env.reset()  
  
    for i in range(len(trade_data.index.unique())):  
        action, _states = model.predict(trade_obs)  
        trade_obs, rewards, dones, info = trade_env.step(action)  
        if i == (len(trade_data.index.unique()) - 2):  
            # print(env_test.render())  
            last_state = trade_env.envs[0].render()  
  
    df_last_state = pd.DataFrame({"last_state": last_state})
```

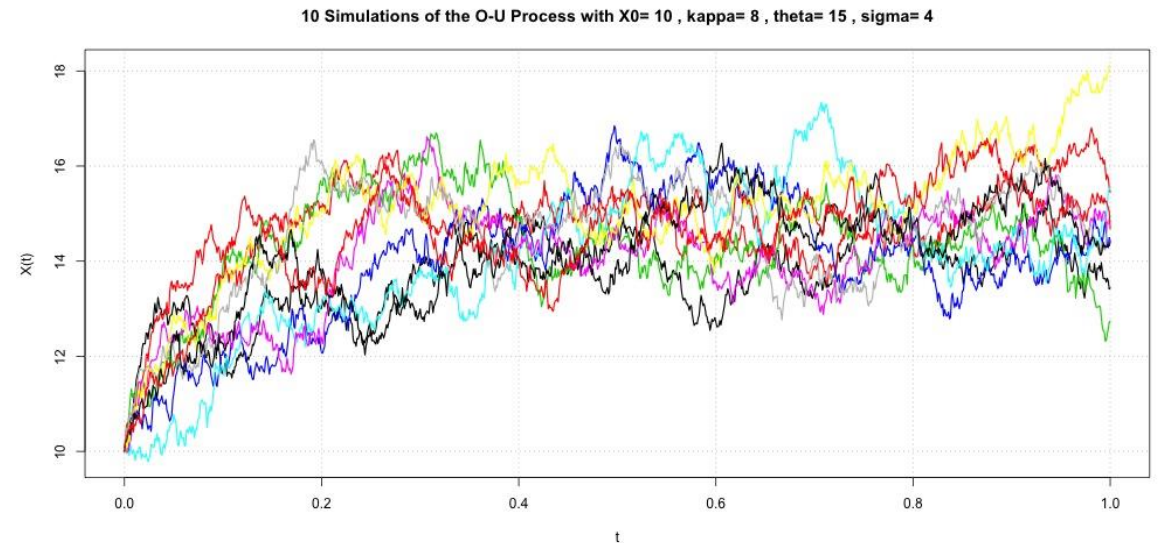
```
# Model Selection based on sharpe ratio  
  
if (sharpe_ddpg >= sharpe_sac) & (sharpe_ddpg >= sharpe_td3):  
    model_use.append("DDPG")  
    model_ensemble = model_ddpg  
  
elif (sharpe_sac >= sharpe_ddpg) & (sharpe_sac >= sharpe_td3):  
    model_use.append("SAC")  
    model_ensemble = model_sac
```

Hard Vote the model with 'highest' sharpe ratio

## 04. ENSEMBLE WITH SHARPE RATIO

### ACTION\_NOISE : ORSTEIN\_UHLENBECK PROCESS

- Orstein Ulhenbeck Process (a.k.a. mean-reversion process, Brownian Motion with friction)
- Brownian Motion 의 입자가 마찰(theta) 의 영향을 받아 자신의 이전 상태로 회귀하려는 경향을 가짐. 이러한 경향성이 시간에 따라 심화되면서 long-term mean 으로 회귀하게 되는 확률과정
- 해당 가정은 주식시장에서 과매수/과매도 상황 아래, long-term mean 으로 회귀하려는 특징 반영



$$dX_t = \theta(\mu - X_t)dt + \sigma dW_t$$

Hard Vote the model with 'highest' sharpe ratio

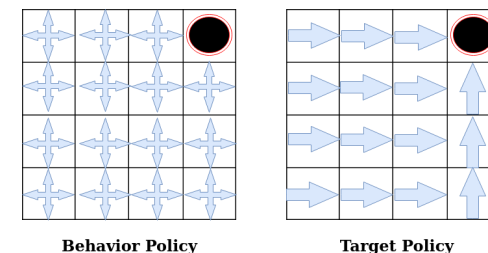
## 04. ENSEMBLE WITH SHARPE RATIO

### WHY OUR PROCESS IN REINFORCEMENT LEARNING?

< 강화학습에 왜 적용되었나? >

```
{'action_noise': OrnsteinUhlenbeckActionNoise(mu=[0, 0, 0, 0, 0, 0, 0, 0, 0, 0], sigma=[0.56389924 0.72619497 0.26787683 0.26787683 0.88537625 0.38545566 0.47702929 0.72945688 1.02494153]), 'buffer_size': 1000000, 'learning_rate': 0.0005, 'batch_size': 64, 'device': 'cuda'}
```

- Off\_Policy Algorithm 에서 Action\_Noise 가 Orstein Uhlenbeck Process 를 따른다는 가설.
- 기존 Uniformly random noise를 parametric noise 로 상정하여
  - (1) Statistical Efficiency (충분한 episode가 확보되었을때)
  - (2) precise dynamics
- OU process의 평균회귀 가설은 moving average를 state\_space로 사용하여 주식시장의 움직임을 capture하고자 하는 DRL모델과 동일한 가설
- 평균회귀 가설이 statistically significant한 경우 움직임을 예측하는데 있어 효과적.



```
if model_kwargs is None:
    model_kwargs = MODEL_KWARGS[model_name]

if "action_noise" in model_kwargs:
    n_actions = self.env.action_space.shape[-1]
    model_kwargs["action_noise"] = NOISE[model_kwargs["action_noise"]](
        mean=np.zeros(n_actions), sigma=sigma_ * np.ones(n_actions), theta=theta_ * np.ones(n_actions)
    )
print(model_kwargs)
```

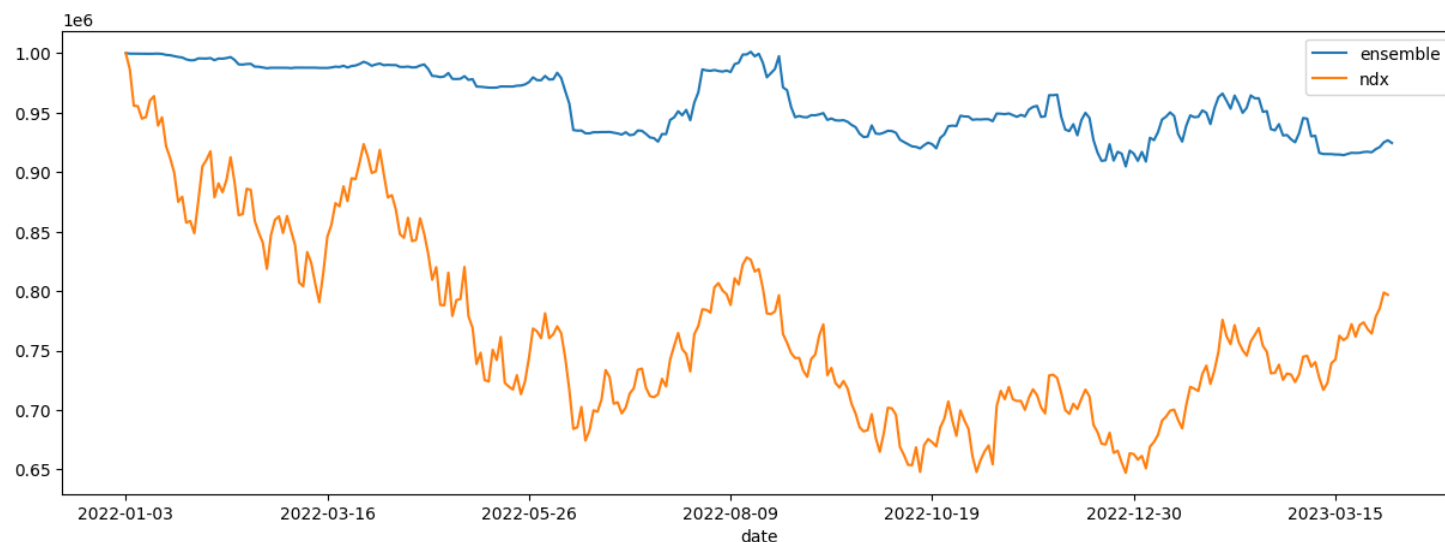


Hard Vote the model with 'highest' sharpe ratio

## 04. ENSEMBLE WITH SHARPE RATIO

### RESULTS : (1) ALL\_ENSEMBLE

|   | Iter | Val Start  | Val End    | Model Used | A2C Sharpe | PPO Sharpe | DDPG Sharpe | SAC Sharpe | TD3 Sharpe |
|---|------|------------|------------|------------|------------|------------|-------------|------------|------------|
| 0 | 126  | 2021-10-04 | 2022-01-03 | PPO        | 0.326257   | 0.583925   | 0.222091    | 0.37878    | 0.319924   |
| 1 | 189  | 2022-01-03 | 2022-04-04 | SAC        | -0.320031  | -0.320185  | -0.089441   | -0.017085  | -0.250471  |
| 2 | 252  | 2022-04-04 | 2022-07-06 | A2C        | -0.52318   | -0.593024  | -0.523324   | -0.534957  | -0.526993  |
| 3 | 315  | 2022-07-06 | 2022-10-04 | A2C        | 0.027793   | 0.021532   | -0.136567   | -0.058768  | 0.014166   |
| 4 | 378  | 2022-10-04 | 2023-01-04 | SAC        | -0.108112  | -0.100759  | -0.155534   | -0.083647  | -0.13775   |



```
=====Get Backtest Results=====
Annual return      -0.060900
Cumulative returns  -0.075536
Annual volatility   0.093849
Sharpe ratio       -0.624716
Calmar ratio       -0.632560
Stability          0.611913
Max drawdown       -0.096275
Omega ratio        0.874434
Sortino ratio      -0.843192
Skew               NaN
Kurtosis           NaN
Tail ratio         0.903092
Daily value at risk -0.012057
dtype: float64
```

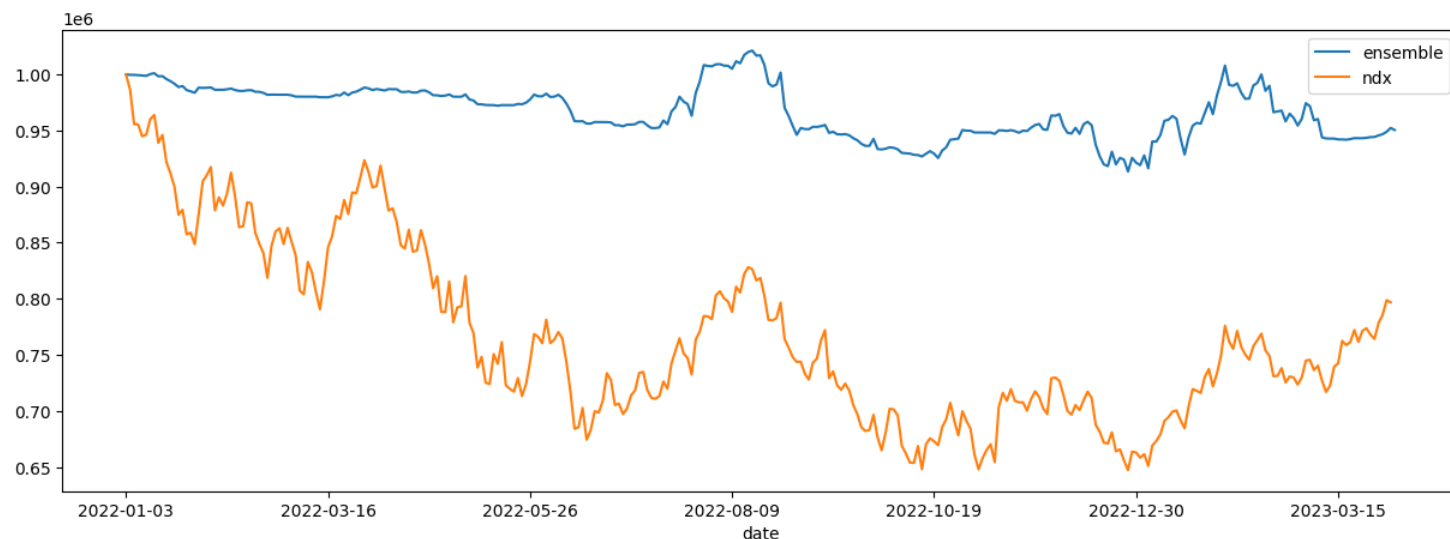
```
=====Get Baseline Stats=====
[*****100%*****]
Shape of DataFrame: (314, 8)
Annual return      -0.166662
Cumulative returns  -0.203216
Annual volatility   0.308976
Sharpe ratio       -0.437664
Calmar ratio       -0.472349
Stability          0.480914
Max drawdown       -0.352837
Omega ratio        0.933247
Sortino ratio      -0.609704
Skew               NaN
Kurtosis           NaN
Tail ratio         1.018180
Daily value at risk -0.039464
```

Hard Vote the model with 'highest' sharpe ratio

## 04. ENSEMBLE WITH SHARPE RATIO

### RESULTS : (2) ON\_POLICY ENSEMBLE

|   | Iter | Val Start  | Val End    | Model Used | A2C Sharpe | PP0 Sharpe |
|---|------|------------|------------|------------|------------|------------|
| 0 | 126  | 2021-10-04 | 2022-01-03 | PPO        | 0.278107   | 0.451536   |
| 1 | 189  | 2022-01-03 | 2022-04-04 | PPO        | -0.244684  | -0.215169  |
| 2 | 252  | 2022-04-04 | 2022-07-06 | A2C        | -0.523688  | -0.54537   |
| 3 | 315  | 2022-07-06 | 2022-10-04 | A2C        | -0.120781  | -0.303323  |
| 4 | 378  | 2022-10-04 | 2023-01-04 | A2C        | -0.086978  | -0.187604  |



```
=====Get Backtest Results=====
Annual return      -0.039846
Cumulative returns -0.049557
Annual volatility   0.095266
Sharpe ratio        -0.380597
Calmar ratio        -0.377323
Stability           0.327644
Max drawdown        -0.105602
Omega ratio         0.919837
Sortino ratio       -0.522220
Skew                NaN
Kurtosis            NaN
Tail ratio          0.972269
Daily value at risk -0.012146
dtype: float64
```

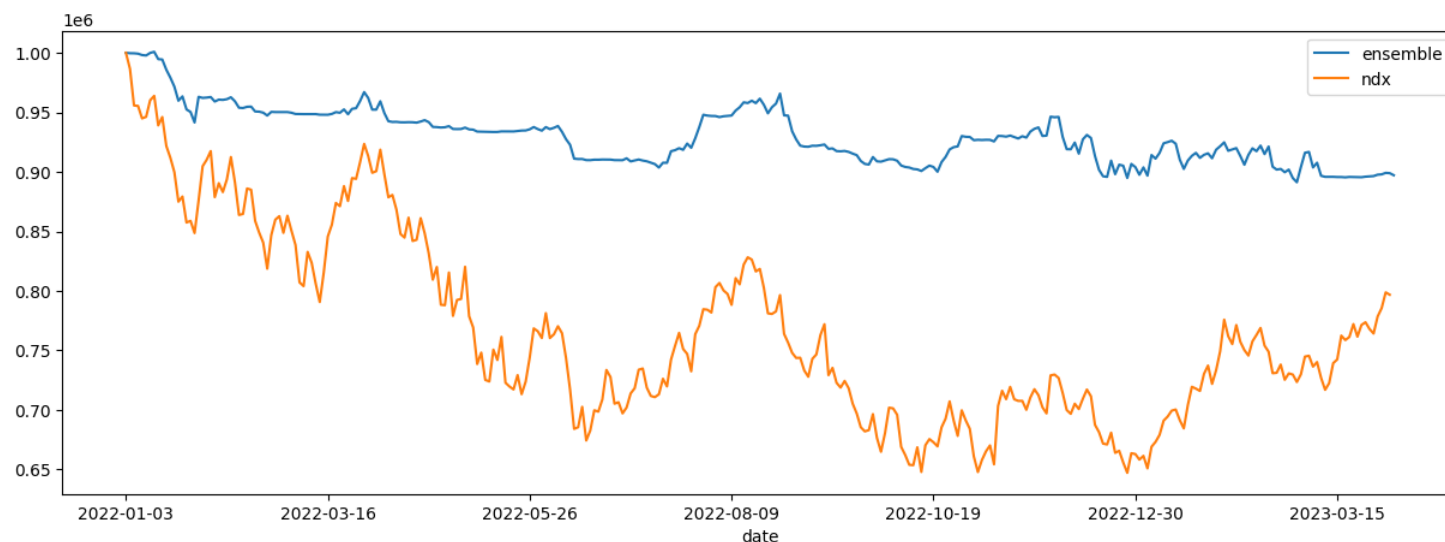
```
=====Get Baseline Stats=====
[*****100%*****]
Shape of DataFrame: (314, 8)
Annual return      -0.166662
Cumulative returns -0.203216
Annual volatility   0.308976
Sharpe ratio        -0.437664
Calmar ratio        -0.472349
Stability           0.480914
Max drawdown        -0.352837
Omega ratio         0.933247
Sortino ratio       -0.609704
Skew                NaN
Kurtosis            NaN
Tail ratio          1.018180
Daily value at risk -0.039464
```

Hard Vote the model with 'highest' sharpe ratio

## 04. ENSEMBLE WITH SHARPE RATIO

### RESULTS : (3) OFF\_POLICY ENSEMBLE

|   | Iter | Val Start  | Val End    | Model Used | DDPG Sharpe | SAC Sharpe | TD3 Sharpe |
|---|------|------------|------------|------------|-------------|------------|------------|
| 0 | 126  | 2021-10-04 | 2022-01-03 | SAC        | 0.248094    | 0.413804   | 0.388397   |
| 1 | 189  | 2022-01-03 | 2022-04-04 | SAC        | -0.348656   | -0.125916  | -0.282326  |
| 2 | 252  | 2022-04-04 | 2022-07-06 | SAC        | -0.540129   | -0.492693  | -0.534592  |
| 3 | 315  | 2022-07-06 | 2022-10-04 | DDPG       | 0.01164     | -0.031788  | -0.112318  |
| 4 | 378  | 2022-10-04 | 2023-01-04 | DDPG       | 0.08509     | -0.121076  | -0.081355  |



```
=====Get Backtest Results=====
Annual return      -0.083149
Cumulative returns -0.102833
Annual volatility   0.085803
Sharpe ratio       -0.972010
Calmar ratio       -0.759375
Stability          0.590605
Max drawdown       -0.109497
Omega ratio        0.819928
Sortino ratio      -1.311449
Skew               NaN
Kurtosis           NaN
Tail ratio         0.861816
Daily value at risk -0.011141
dtype: float64
```

```
=====Get Baseline Stats=====
[*****100%*****]
Shape of DataFrame: (314, 8)
Annual return      -0.166662
Cumulative returns -0.203216
Annual volatility   0.308976
Sharpe ratio       -0.437664
Calmar ratio       -0.472349
Stability          0.480914
Max drawdown       -0.352837
Omega ratio        0.933247
Sortino ratio      -0.609704
Skew               NaN
Kurtosis           NaN
Tail ratio         1.018180
Daily value at risk -0.039464
```

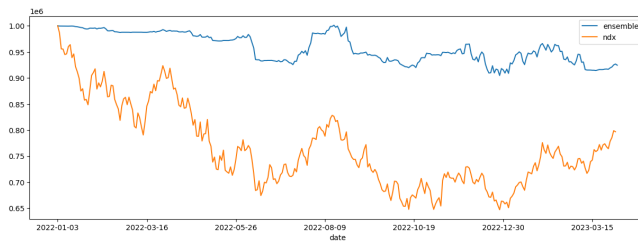
Hard Vote the model with 'highest' sharpe ratio

# 04. ENSEMBLE WITH SHARPE RATIO

## RESULTS : 3 STRATEGIES

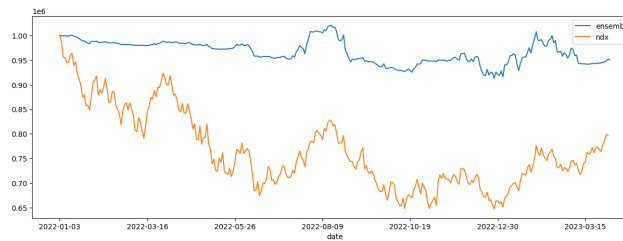
### All\_Ensemble

- A2C, PPO, DDPG, SAC, TD3
- Cumulative Returns : -0.07554
- Annual Volatility : 0.093849
- Ensemble took : 356 minutes



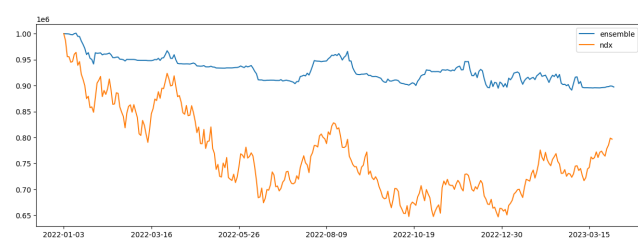
### On-Policy Algorithms

- A2C, PPO
- Cumulative Returns : **-0.04956**
- Annual Volatility : 0.095266
- Ensemble took : 78 minutes



### Off-Policy Algorithms

- DDPG, SAC, TD3
- Cumulative Returns : -0.10283
- Annual Volatility : 0.085803
- Ensemble took : 264 minutes



```
=====Get Baseline Stats=====
Cumulative returns  -0.203216
Annual volatility   0.308976
```

# CONCLUSION

---

## 의의

- Prediction 으로 강화학습의 성능을 향상시킬 수 있음을 확인
- 스터디에서 쉽게 배우지 못하는 새로운 주제를 할 수 있었음
- Dlinear를 FinRL의 indicator로 추가해보았다.

## 한계

- XAI 관점에서 투자의사결정 설명이 어려움 (Black Box)
- 파라미터 튜닝 과정에서 모델(Optuna)의 한계로 적용하지 못했음.
- 하이퍼파라미터가 너무 많아 모듈화 간소화가 어려웠음.

**EOD**

