

2019년 논문 발표

The Effectiveness of the Trading with Reinforcement Learning : through Policy Gradients and Neural Networks

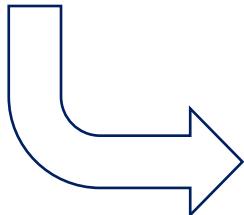
2019.11

고은환

- ✓ 강화학습의 Trading decision 선택 로직 설정
- ✓ Finance 관점에서의 전략 설정
- ✓ 모델에 대한 검증
- ✓ 위의 세 가지를 결합한 코드 구현
- ✓ Input data (Index)를 통해 1차 결과 확인
- ✓ 추가 보완 사항 확인

3 Parts

1. Data validation
2. Feature learning (through data transformation)
3. Based on learning, determine trading decision (policy)



아래의 두 가지 2번과 3번이 Trading decision을 수행하게 되는 Agent의 전략 또는 행동 선택 알고리즘
이 전략은 Machine learning 관점과 Finance 관점으로 나뉨

In terms of Machine Learning

- ✓ 기본적으로 강화학습 모델의 알고리즘으로 구성
- ✓ Trading decision을 수행하는 Agent가 자기가 속한 Environment 또는 State에서 목표 함수를 최대화하는 행동을 선택하며, 이를 바탕으로 전략을 설정

DRL (Direct Reinforcement Learning)

- ✓ 일반적인 DRL은 one-layer RNN로 구성되어 있음
- ✓ Price sequence: P_1, P_2, \dots, P_t
- ✓ Return at each time point t : $Z_t = P_t - P_{t-1}$
- ✓ Based on the market conditions, trading decision (policy) $\delta_t \in \{long, short\} = \{1, 0\}$ is made on each time point t .
- ✓ Profit R_t made by the trading model is obtained by $R_t = \delta_{t-1}Z_t - c|\delta_t - \delta_{t-1}|$.

$$\text{Profit } R_t = \delta_{t-1} Z_t - c |\delta_t - \delta_{t-1}|$$

- ✓ 1st term: Market fluctuation에 의해 만들어지는 P/L
- ✓ 2nd term: Time point t 에 포지션을 변경했을 때 발생하는 Total cost
 - TC(c): Mandatory fee only if $\delta_t \neq \delta_{t-1}$,
- ✓ 위 함수는 typical DRL frameworks에서 정의되는 value function
 - When getting the value function in each time point, the accumulated value throughout the whole training period can be defined as

$$\max_{\Theta} U_T(R_1, \dots, R_T | \Theta)$$

where $U_T\{\cdot\}$ is accumulated rewards in the period of $1, \dots, T$. ($U_T = \sum_{t=1}^T R_t$)

Trading action with non-linear function in DRL

- ✓ Purpose: to approximate the trading action (policy) at each time point by $\delta_t = \text{sigmoid}[\langle \mathbf{w}, \mathbf{f}_t \rangle + b + u\delta_{t-1}]$.
- ✓ $\langle \cdot, \cdot \rangle$: inner product
- ✓ \mathbf{f}_t : Feature vector of the current market condition at time t
- ✓ \mathbf{w}, b : Coefficient for the feature regression
- ✓ Recent m return values are **directly** adopted as the feature vector

$$\mathbf{f}_t = [Z_{t-m+1}, \dots, Z_t] \in \mathbb{R}^m$$

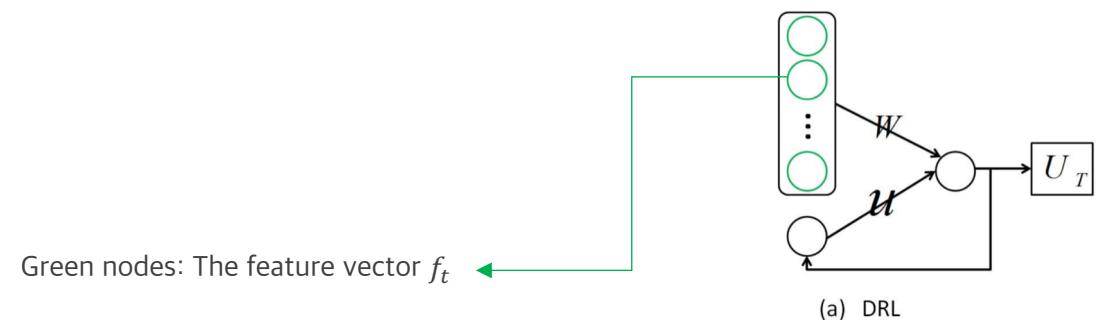
Trading action with non-linear function in DRL

- ✓ $\delta_t = \text{sigmoid}[\langle \mathbf{w}, f_t \rangle + b + u\delta_{t-1}]$.
- ✓ $u\delta_{t-1}$: 최근의 매매 의사결정을 고려하기 위해 포함 → to discourage the agent to frequently change the trading positions and hence, to avoid heavy TCs.
- ✓ sigmoid(\cdot): maps the function into the range of (0, 1) to approximate the final trading decision
- ✓ 본 모델 DRL의 목표: to learn such a parameter set $\Theta = \{\mathbf{w}, u, b\}$ that can maximize the global reward function

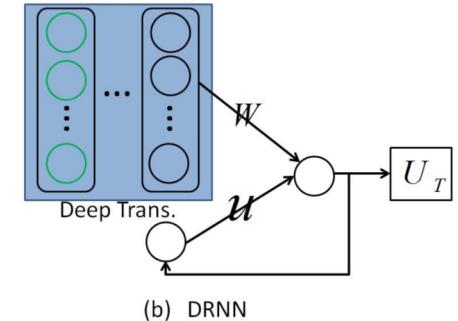
$$\max_{\Theta} U_T(R_1, \dots, R_T | \Theta)$$

The weakness of DRL

- ✓ 먼저 그림 (a)에서 보듯이 DRL의 구조는 Recurrent structure
 - δ_{t-1} is used as the input of the neuron to calculate δ_t (이전에 선택한 행동을 현재 선택에 고려)
 - to incorporate the long time memory into the learning system
 - to keep the past trading actions in the memory to discourage changing trading positions frequently.
 - Recursively generate trading actions (\rightarrow learning policy directly)
- ✓ 문제: the lack of a feature learning part to robustly summarize the (noisy) market conditions
- ✓ DRL은 Non-linear regression이지만 one-layer NN이므로 이를 보완할 필요성 \rightarrow Make it Deeper!



Development of DRL, DDR (Deep Direct Reinforcement)



- ✓ Make it deeper!
- ✓ Introduce the prevalent DL into DRL for simultaneously feature learning and dynamic trading.
- ✓ DRNN(Deep Recurrent Neural Network) + DRL = DDR
- ✓ 즉 기본 DRL에서는 1 layer의 RNN(LSTM)이 기본 틀이었지만, 이를 deep하게 만들어 부족한 feature learning part 보완
- ✓ Hierarchically transform the information from layer to layer.

⇒ 기본 프레임워크 DRL을 Deep하게 만드는 것을 Deep transformation (Deep representation)이라고 한다.

(→ It encourages much informative feature representations for a specific learning task.)

New trading action with improved feature learning

- ✓ Deep representation: $\mathbf{f}_t = g_d(\mathbf{f}_t)$
 - obtained by hierarchically transforming the input vector f_t through the DNN with a non-linear function $g_d(\cdot)$
- ✓ New trading action at time point t is subject to $\delta_t = \text{sigmoid}[\langle \mathbf{w}, g_d(\mathbf{f}_t) \rangle + b + u\delta_{t-1}]$
- ✓ 이때 얼만큼 Deep 하느냐 → The number of hidden layers = 4
 - The number of node per hidden layer = [128, 128, 128, 20]

Summary

- ✓ The whole optimization framework is given as follows:

$$\max_{\{\Theta, g\{\cdot\}\}} U_T(R_1, \dots, R_T)$$

$$s.t. R_t = \delta_{t-1} Z_t - c |\delta_t - \delta_{t-1}|$$

$$\delta_t = \text{sigmoid}[\langle \mathbf{w}, \mathbf{F}_t \rangle + b + u \delta_{t-1}]$$

$$\mathbf{F}_t = g_d(\mathbf{f}_t)$$

- ✓ Two groups of parameters to be learned:
Trading parameters $\Theta = \{\mathbf{w}, u, b\}$ & Deep transformations $g_d(\cdot)$

In terms of Finance

✓ 설정 사항 (Attributes)

- 초기 투자금: initialBalance = 100,000,000 (1억원)
- 현재 잔고: currentBalance
- 보유 증권 수: numHoldingSecurities
- 포트폴리오 가치: portfolioValue (=투자금 잔고 + (현재가격 * 수량))
- 기존 포트폴리오 가치: basePortfolioValue (학습 이전의 포트폴리오 가치)

Strategy Composition

- ✓ Agent의 상태 2가지
 1. 증권 보유 비율 (최대 보유 가능한 주식 수 대비 현재 보유하고 있는 증권 수의 비율)
→ 행동 선택의 편중을 막기 위한 데이터 (보유 중 or 보유하고 있지 않을 때의 수량이 한 쪽으로 쏠리는 상황을 막기 위함)
 2. 포트폴리오 가치 비율 (직전에 지연 보상이 발생했을 때의 포트폴리오 가치 대비 현재 포트폴리오 가치의 비율)
→ 포트폴리오의 수익률을 고려하기 위해 필요한 데이터
- ✓ 거래 수수료: TRADING_CHARGE = 01.00015 (0.015%)
- ✓ 거래세: TRADING_TAX = 0.003 (0.3%)
- ✓ 가능한 매매 행동의 종류 = {Long, Short, Hold}
 - 이중 앞의 DRNN (Policy_network)에서 확률이 구해질 행동은 매수와 매도 2가지
 - Hold는 매수와 매도의 확률을 고려해서 선택된 행동이 수행 불가능 할 경우에 Hold 하는 것으로 설정

Strategy Composition

- ✓ 매수 또는 매도가 나왔는데 관망을 하는 경우, 즉 매수/매도가 불가능한 경우를 고려
 - 매수가 결정됐을 때 가지고 있는 잔금이 최소 매매 금액보다 부족한 경우 Hold
 - 매도가 결정됐을 때 매매하고자 하는 단위가 최소 매매 단위보다 작은 경우 Hold
 - Confidence를 통해 매매 단위에 붙은 $+ \alpha$ 를 모두 매매 하지 못한다면, 가능한 만큼을 매매하도록 설정
- ✓ 거래 시 가능한 단위 및 최소 / 최대 매매 단위 설정
 1. 기본 거래 가능 단위 = (기본 1) + α
→ 이 alpha는 DRNN에 의해 구해지는 각 행동들의 확률 값이 클 수록 (Confidence가 높을 수록) 더 많은 단위의 매매가 가능하도록 alpha를 설정 (기준은 0.1 단위당 한 단위 증가)
 2. minTradingUnit = 1
 3. (maxTradingUnit = (기본 1) + max α)

Strategy Composition

- ✓ Agent가 행동하는 배경, 즉 환경 (Environment / State)
 - Date, Open, High, Low, AdjClose, Volume
 - 강화학습 모델 중 policy_network 부분, 즉 DRNN에서의 Training에 사용되는 인풋 데이터와는 다름
 - 말 그대로 Agent가 행동할 수 있는 Market condition / environment를 나타내기 위한 데이터

Strategy Composition

✓ Delayed reward 와 Batch-size data

- Immediate reward: 바로 전일의 포트폴리오 가치와 비교해서 학습을 시키기도 하지만,
- Delayed reward: Threshold를 설정하여 (위 또는 아래로) 넘는 경우 이전에 threshold를 넘은 다음 날부터 지금까지의 행동을 모아서 학습시키기도함
- 앞에서 언급한 포트폴리오 가치 비율은 지연 보상이 일어났을 때 포트폴리오 가치가 어떻게 변하는지를 확인
- 이 지연 보상이 발생할 때마다 직전 학습 시점의 포트폴리오 가치(basePortfolioValue)를 갱신하여, 현재 포트폴리오 가치가 (이 기준 포트폴리오와 비교하여) 가치가 증가했는지 감소했는지 비교
- 즉시 보상과 지연 보상으로 나누는 이유
 - : 즉시 보상을 판단하는 이유는 지연 보상을 판단할 수 있게 될 때까지 시간이 걸리기 때문에, 그때까지의 학습에 필요한 기준이 되는 방향을 정해주기위함

How to deal with Overfitting?

✓ 반드시 해결해야 할 문제: Overfitting

- L1 Regularization (Lasso)
- Model selection (k-fold Cross Validation)
- Early stopping
- Dropout

L1 Regularization (LASSO)

- ✓ 여러 Data feature 및 Data transformation을 통해 차원이 높아진 차수들의 계수 Weight(혹은 Theta)를 줄이는 방법
- ✓ Loss function을 최소화함과 동시에 \mathbf{W} 도 줄이는 해결책

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^m \left(h_{\theta}(x^{(i)}) - y^{(i)} \right)^2 + \frac{\lambda}{2} \sum_{j=1}^n |\theta_j|$$

- ✓ \mathbf{W} 가 계속해서 업데이트 될 때, \mathbf{W} 의 원소인 어떤 w_i 는 0 이 되도록 한다. (계속해서 특정 상수를 빼기 때문)
- ✓ 따라서, L1은 영향을 크게 미치는 핵심적인 feature \mathbf{f}_t 들만 반영하도록 한다.

Model Selection (k-fold Cross Validation)

- ✓ 새로운 Data set에 대해 반응하는 모델의 성능을 추정하는 방법
 - 특정 데이터를 Training set과 Test set으로 분할 (7 : 3)
 - Training set을 다시 k개의 그룹(fold)로 나눔
 - k개로 나누어진 Training set에 대해 서로 다른 Validation Fold를 지정하며 아래 과정 k번 반복
 - (k - 1)개의 Train Fold에 대해 학습 진행
 - 나머지 1개의 Validation Fold에 대해 성능 측정
 - 위에서 얻은 각 Hyperparameter의 k개의 결과에 대한 평균을 계산하여 이 평균 값을 대표 Hyperparameter로 지정
 - 마지막으로 이 Hyperparameters를 바탕으로 Test Data에 대해 모델 1회 Test

Dropout

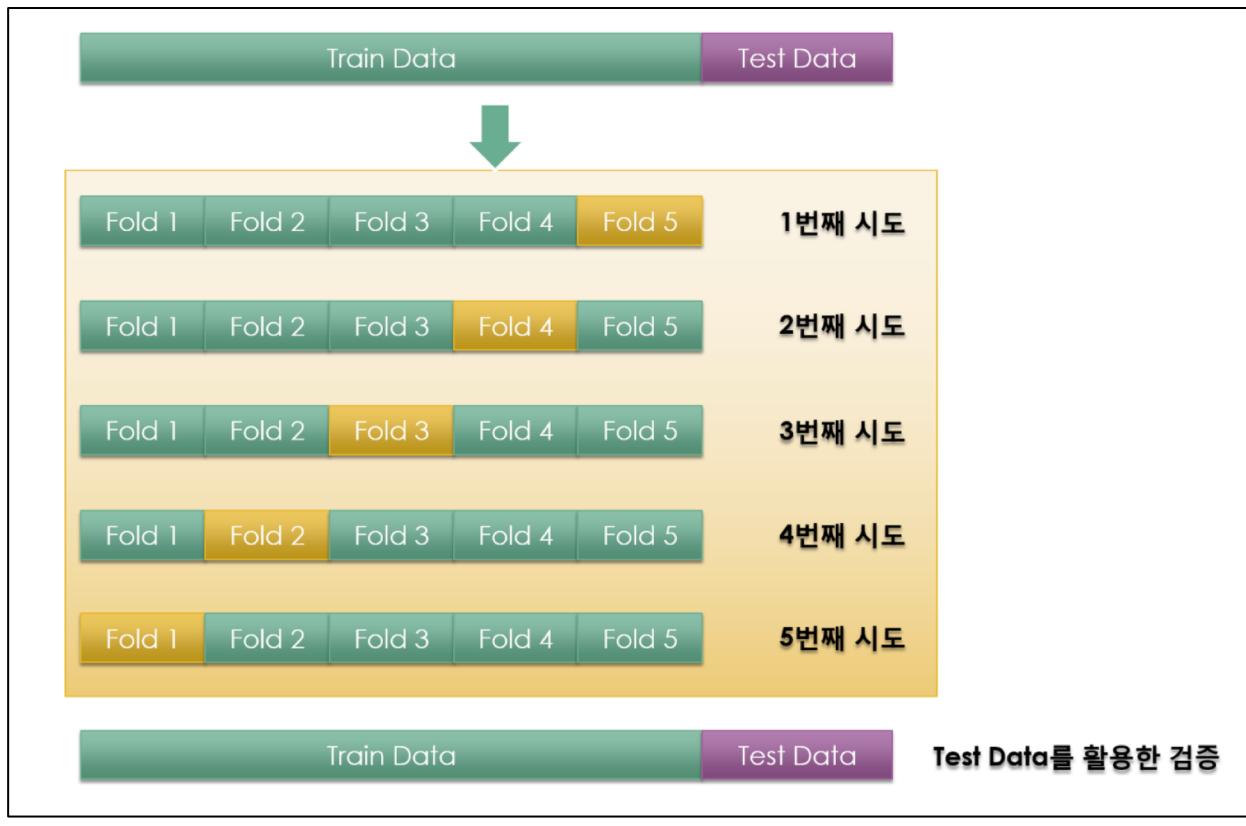
- ✓ 복잡한 Neural Network에서 몇몇 Node들의 관계를 끊어버리는 것
- ✓ 전체적으로 보았을 때, 올바르게 학습할 수 있다는 원리
- ✓ Input layer와 Hidden layer에 있는 Node들을 정해진 Dropout rate 만큼 제거
- ✓ 진행되는 학습 차수마다 랜덤으로 Node들에 적용되며 반복

Early Stopping

- ✓ K-fold Cross Validation을 통해 나누어진 3개의 Set 중 Validation set 과정에서 Overfitting이 일어날 때, 정해진 Threshold를 넘어서는 순간 학습을 Stop
- ✓ 그 후, Test set으로 남겨둔 데이터로 Final evaluation 진행

L1 Regularization (LASSO)

- ✓ 여러 Data feature 및 Data transformation을 통해 차원이 높아진 차수들의 계수 Weight(혹은 Theta)를 줄이는 방법
 - ✓ Loss function을 최소화함과 동시에 \mathbf{W} 도 줄이는 해결책
- $$J(\theta) = \frac{1}{2m} \sum_{i=1}^m \left(h_{\theta}(x^{(i)}) - y^{(i)} \right)^2 + \frac{\lambda}{2} \sum_{j=1}^n |\theta_j|$$
- ✓ \mathbf{W} 가 계속해서 업데이트 될 때, \mathbf{W} 의 원소인 어떤 w_i 는 0 이 되도록 한다. (계속해서 특정 상수를 빼기 때문)
 - ✓ 따라서, L1은 영향을 크게 미치는 핵심적인 feature \mathbf{f}_t 들만 반영하도록 한다.



Dropout

- ✓ 복잡한 Neural Network에서 몇몇 Node들의 관계를 끊어버리는 것
- ✓ 전체적으로 보았을 때, 올바르게 학습할 수 있다는 원리
- ✓ Input layer와 Hidden layer에 있는 Node들을 정해진 Dropout rate 만큼 제거
- ✓ 진행되는 학습 차수마다 랜덤으로 Node들에 적용되며 반복

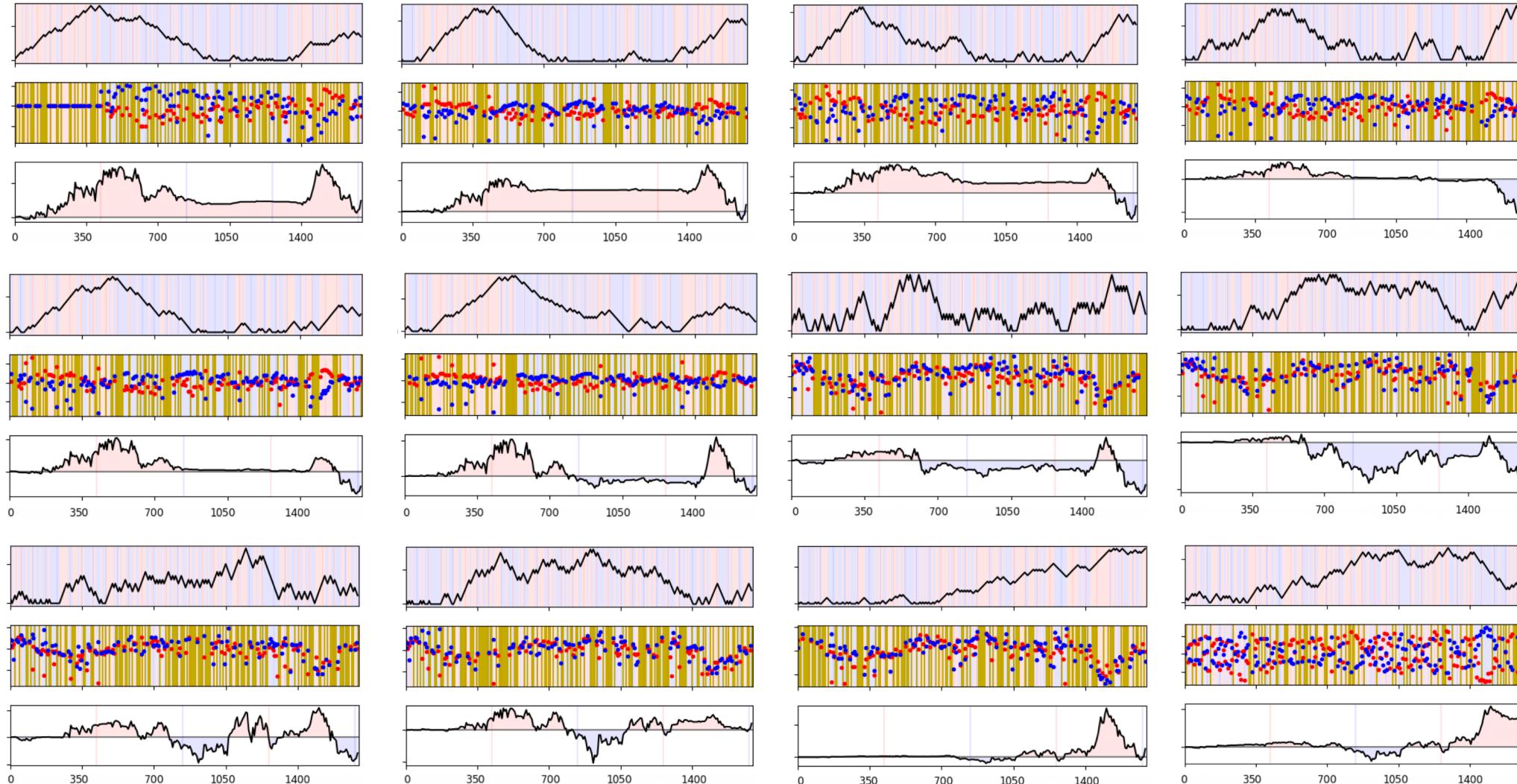
Early Stopping

- ✓ K-fold Cross Validation을 통해 나뉘어진 3개의 Set 중 Validation set 과정에서 Overfitting이 일어날 때, 정해진 Threshold를 넘어서는 순간 학습을 Stop
- ✓ 그 후, Test set으로 남겨둔 데이터로 Final evaluation 진행

Data for Learning

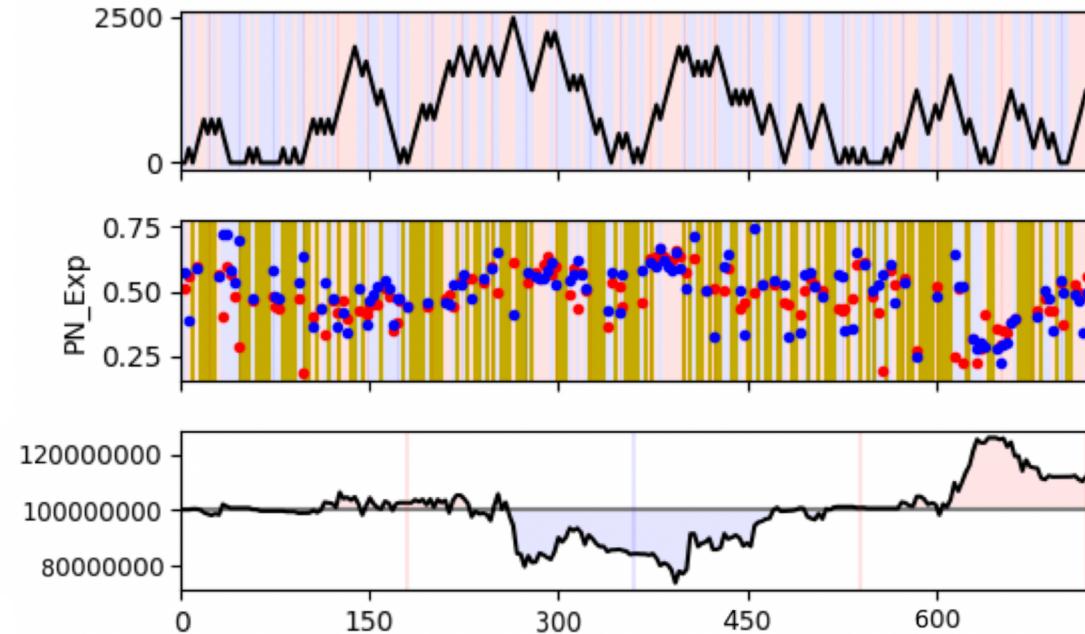
- ✓ 강화학습 (DDR)의 Environment 구성과 Agent 행동 선택을 위한 데이터 (2009년 ~ 2019년의 시장 Index)
 - In-Sample: 2009년 9월 ~ 2016년 9월의 시장 데이터
 - Out-Sample: 2016년 9월 ~ 2019년 9월 시장 데이터
- ✓ 에이전트의 상태
 - 투자 수행의 주체는 에이전트의 상태에 따라 같은 환경에서도 다른 투자를 결정할 수 있으므로 학습 데이터로 고려
 1. 증권 보유 비율 (최대 보유 가능한 주식 수 대비 현재 보유하고 있는 증권 수의 비율)
 2. 포트폴리오 가치 비율 (직전에 지연 보상이 발생했을 때의 포트폴리오 가치 대비 현재 포트폴리오 가치의 비율)

In-Sample 결과



시장 수익률: 19.73%
모델 수익률: 31.15%

Out-Sample 결과



시장 수익률: 16.71%
모델 수익률: 19.27%

추가 사항

- ✓ 공매도 기능
- ✓ Sharpe Ratio
- ✓ 일반적인 팩터 전략(ETF) 결과와 비교



감사합니다