



Algorithm Trading

나는 놀고먹으며
부자가 될 수 있을까

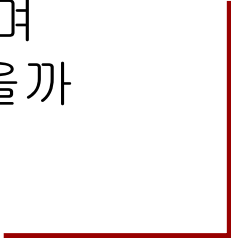


Table of Contents

1. About Algorithm Trading (Recap.)
2. Trading Strategy
3. Experimental Result
4. Discussion and Future Work

Algorithm Trading

Algorithm Trading



대한민국 환경부 등급 기준 ⓘ

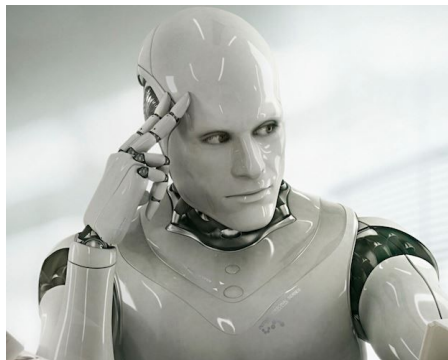
☁ 10° >



매우나쁨



데이터 없음



Buy
Sell
Hold

Companies to Predict

대기오염수치 데이터와 주가와의 관계

전기차 : 미세먼지 저감조치에 따른 석유 및 석탄 등의 화력발전 비율 감소와 신재생에너지의 사용량 증가

제약 : 대기오염은 안구질환 및 호흡기질환을 유발하며, 발암물질을 포함하고 있음. 따라서 제약 산업은 대기오염과 관련이 있다.

마스크 : 대기오염, 특히 미세먼지를 막기 위해 일반인들이 가장 많이 구매하는 제품 중 하나가 마스크임

대기환경설비 : 대기오염이 늘어남에 따라 대기환경을 정화할 수 있는 설비에 대한 관심도 높아짐.

Backtesting

- Backtesting이란?

새로운 전략개발 또는 기존 전략의 검증을 위해 과거 데이터를 가지고, 실험하고자 하는 전략의 성과를 측정해 보는 과정이다.

- 단, 과거 데이터를 사용하기 때문에 과거의 높은 수익률이 미래에도 재현될 것이라고 보장할 수 없다 : 높은 수익률을 낸 전략이라 하더라도, 시간이 지나 실패할 수 있다.

Trading Strategies

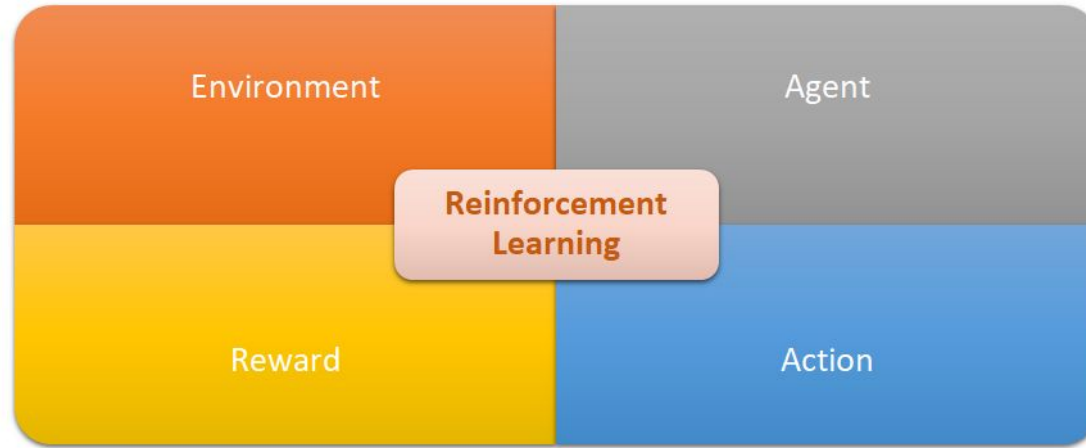
1. Reinforcement Learning
2. Pair Trading
3. NAS with Weight Agnostic Neural Networks
4. Price Predicting Model

Strategy 1

? Search with Reinforcement Learning

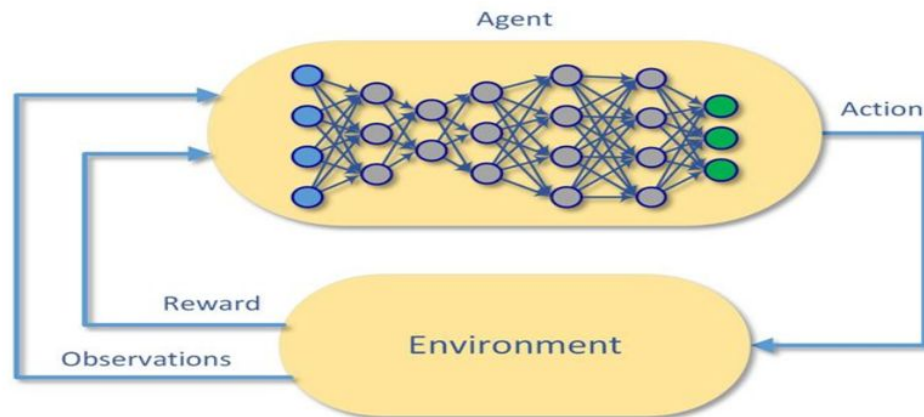
“Train the agent which determines Buy/Sell/Hold”

Reinforcement Learning on Algorithm Trading



***Reinforcement learning (RL)** is an area of **machine learning** concerned with how **software agents** ought to take actions in **an environment** so as to maximize some notion of **cumulative reward**.*

Reinforcement Learning on Algorithm Trading



***Reinforcement learning (RL)** is an area of **machine learning** concerned with how **software agents** ought to take actions in **an environment** so as to maximize some notion of **cumulative reward**.*

Reinforcement Learning - Winix

```
In [18]: performance = strategy.run(steps=100000)
#performance = strategy.run(steps=10000, epis
```

```
Finished running strategy.
Total episodes: 23 (100000 timesteps).
Average reward: -0.1271378570353822.
```

```
In [19]: performance.head()
```

```
Out[19]:
```

	step	balance	net_worth
0	1.0	10000.000000	10000.000000
1	2.0	10000.000000	10000.000000
2	3.0	10000.000000	10000.000000
3	4.0	2522.826289	9874.311452
4	5.0	2522.826289	9080.577835

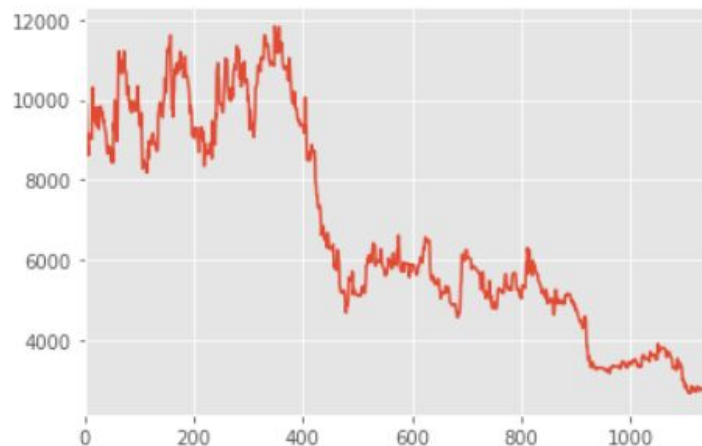
```
In [20]: performance.tail()
```

```
Out[20]:
```

	step	balance	net_worth
1144	1145.0	1475.908756	2671.808851
1145	1146.0	24.884983	2654.995328
1146	1147.0	24.884983	2628.090914
1147	1148.0	0.468286	2654.970837
1148	1149.0	1952.129665	2611.641055

```
In [21]: %matplotlib inline
performance.net_worth.plot()
```

```
Out[21]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb786b02e10>
```



Reinforcement Learning - 하이마트

```
In [13]: performance.head()
```

Out[13]:

	step	balance	net_worth
0	1.0	7520.707476	9971.202364
1	2.0	8155.291367	10066.945397
2	3.0	8621.740120	10025.296492
3	4.0	8955.432572	9974.142572
4	5.0	9950.432906	9950.432906

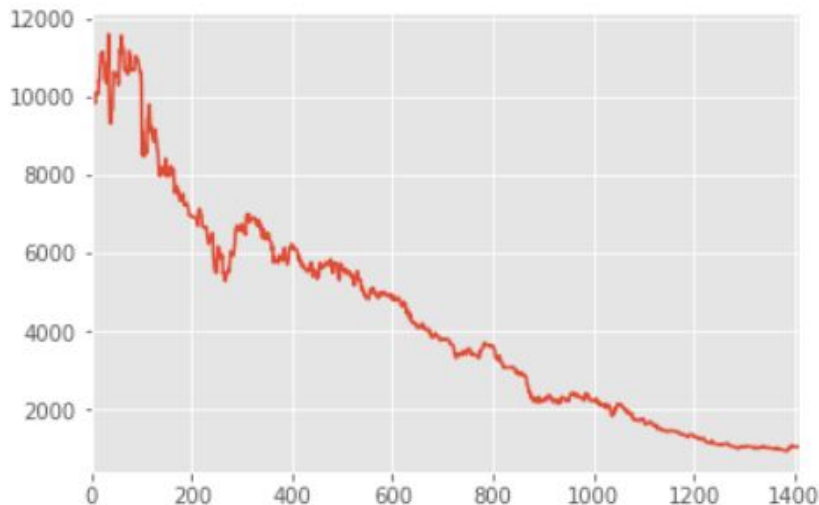
```
In [14]: performance.tail()
```

Out[14]:

	step	balance	net_worth
1403	1404.0	933.003749	1053.621933
1404	1405.0	933.003749	1055.666309
1405	1406.0	474.367651	1055.572251
1406	1407.0	615.840367	1049.946277
1407	1408.0	615.840367	1062.529057

```
In [15]: %matplotlib inline  
performance.net_worth.plot()
```

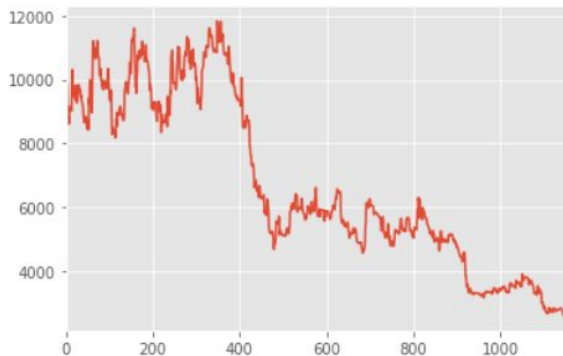
Out[15]: <matplotlib.axes._subplots.AxesSubplot at 0x7f0f36b7a828>



Reinforcement Learning - 실패한 이유

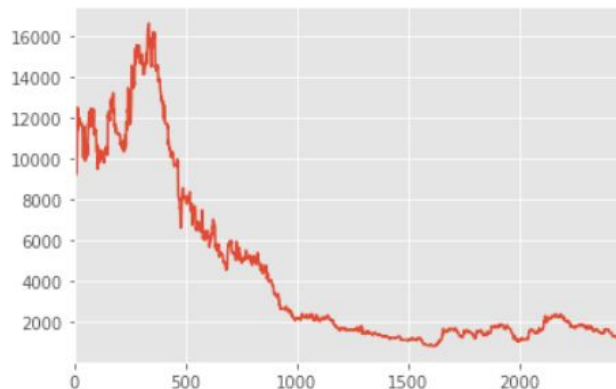
```
In [21]: %matplotlib inline
         performance.net_worth.plot()
```

Out[21]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb786b02e10>



```
In [15]: %matplotlib inline
         performance.net_worth.plot()
```

Out[15]: <matplotlib.axes._subplots.AxesSubplot at 0x7fdbcf6a8ef0>



1. Modulation이 심하다. 동일한 setting이었음에도 불구하고 잃은 정도가 크게 차이가 났다.

Reinforcement Learning - 실패한 이유

```
from tensortrade.rewards import SimpleProfit  
  
reward_scheme = SimpleProfit()
```

The **simple** profit scheme returns a reward of -1 for not holding a trade, 1 for holding a trade, 2 for purchasing an instrument, and a value corresponding to the (positive/negative) profit earned by a trade if an instrument was sold.

2. Reward Scheme이 단순했다.
SimpleProfit() : 너무 단순한 전략이기 때문에 크게 손해를 보았다고 추측.

Reinforcement Learning - 실패한 이유

```
from stable_baselines.common.policies import MlpLnLstmPolicy, CnnLstmPolicy
from stable_baselines import PP02, A2C
```

```
model = PP02
policy = MlpLnLstmPolicy
params = {"learning_rate": 1e-5, "nminibatches": 1 }
```

```
from tensortrade.strategies import StableBaselinesTradingStrategy

strategy = StableBaselinesTradingStrategy(environment=environment,
                                          model=model,
                                          policy=policy,
                                          model_kwargs=params)
```

- Base RL Class
- Policy Networks
- A2C
- ACER
- ACKTR
- DDPG
- DQN
- GAIL
- HER
- PPO1
- PPO2
- SAC
- TD3
- TRPO

3. Parameter Tuning : Model 전략의 문제?

PPO2 보다 성능이 좋은 다양한 전략이 있지만, 현재 TensorTrade가 개발중이기 때문에 적용에 다소 문제가 있었다.

Reinforcement Learning - 실패한 이유

Traceback:

```
<ipython-input-62-6011190cb82f> in <module>()
      7 strategy.environment = environment
      8
----> 9 tuned_performance = strategy.tune(epochs=10)
/usr/local/lib/python3.6/dist-packages/tensortrade/strategies/stable_baselines_strategy.py in tune
      81
      82     def tune(self, steps: int = None, episodes: int = None, callback: Callable[[pd.DataFrame], bool] = None):
--> 83         raise NotImplementedError
      85     def run(self, steps: int = None, episodes: int = None, callback: Callable[[pd.DataFrame], bool] = None):
NotImplementedError
```



ViralBrain commented 26 days ago

Author + 🗨️ ...

I realized that this function is still in development...



notadamking added the **enhancement** label 26 days ago

4. Hyperparameter Tuning

아직 개발중인 기능으로 현재까지는 사용할 수 없다.

Strategy 2

Pair Trading

“정말로 위험을 최소화하면서 수익을 낼까?”

Pair trading

Idea

- 장기적으로 볼 때 페어로 묶인 주가들은 비슷한 추세를 보이지만 단기적으로는 주가들이 서로 다르게 움직일 때가 있다. 이를 일시적 불균형이라 한다.
- 일시적 불균형 상태에 있는 페어를 이용하면 일종의 차익거래 (Arbitrage)를 할 수 있다. 불균형 상태에서 균형 상태로 갈 때 차익을 얻을 수 있는 것이다.
- 매수 - 매도 포지션을 동시에 취하면 시장 전체 위험에 중립적이다.

Pair trading

1. 페어찾기

```
S1 = data[keys[i]]  
S2 = data[keys[j]]  
result = coint(S1, S2)  
score = result[0]  
pvalue = result[1]
```

	S1	S2
1	6400	187790
2	6400	9440
3	12690	20150
4	65510	9520
5	12690	93370
6	12690	6400
7	86520	96770
8	34940	243070
9	65950	131030
10	65950	83550
11	42040	6400
12	45060	640

13	1820	222080
14	2720	71200
15	2720	34940
16	9440	187790
17	2720	58610
18	1540	34940
19	9520	96350
20	65510	108230
21	1540	71200
22	1540	2720
23	640	71200
24	83550	131030

25	520	58610
26	78600	222080
27	3850	33500
28	640	222080
29	520	7610
30	4690	22100
31	4690	71320
32	640	22100
33	6400	2700
34	640	71320
35	640	66570
36	6400	100090

Pair trading

2. 트레이딩 전략 수립

- 매수 신호 포착시, S1을 n 개 매수하고, S2를 $n * \text{Ratio}$ 개 매도한다.
- 매도 신호 포착시, S1을 n 개 매도하고, S2를 $n * \text{Ratio}$ 개 매수한다

Pair trading

3. 트레이딩 전략 실행 코드

```
# Trade using a simple strategy
def trade(S1, S2, window1, window2):

    # If window length is 0, algorithm doesn't make sense, so exit
    if (window1 == 0) or (window2 == 0):
        return 0

    # Compute rolling mean and rolling standard deviation
    ratios = S1/S2
    ma1 = ratios.rolling(window=window1,
                        center=False).mean()
    ma2 = ratios.rolling(window=window2,
                        center=False).mean()
    std = ratios.rolling(window=window2,
                        center=False).std()
    zscore = (ma1 - ma2)/std

    # Simulate trading
    # Start with no money and no positions
    money = 0
    countS1 = 0
    countS2 = 0
```

```
for i in range(len(ratios)):
    # Sell short if the z-score is > 1
    if zscore[i] < -1:
        money += S1[i] - S2[i] * ratios[i]
        countS1 -= 1
        countS2 += ratios[i]
        #print('Selling Ratio %s %s %s %s'%(money, ratios[i], countS1, countS2))
    # Buy long if the z-score is < -1
    elif zscore[i] > 1:
        money -= S1[i] - S2[i] * ratios[i]
        countS1 += 1
        countS2 -= ratios[i]
        #print('Buying Ratio %s %s %s %s'%(money, ratios[i], countS1, countS2))
    # Clear positions if the z-score between -.5 and .5
    elif abs(zscore[i]) < 0.75:
        money += S1[i] * countS1 + S2[i] * countS2
        countS1 = 0
        countS2 = 0
        #print('Exit pos %s %s %s %s'%(money, ratios[i], countS1, countS2))

return money + S1[len(ratios) - 1]*countS1 + S2[len(ratios) - 1]*countS2
```

Pair trading

4. 결과값

	S1	S2	money
1	6400	187790	-658567.0
2	6400	9440	-192236.0
3	12690	20150	-24829.0
4	65510	9520	-19067.0
5	12690	93370	-4881.0
6	12690	6400	-74.0
7	86520	96770	803.0
8	34940	243070	5507.0
9	65950	131030	17073.0
10	65950	83550	20212.0
11	42040	6400	20608.0
12	45060	640	22672.0

13	1820	222080	23112.0
14	2720	71200	24125.0
15	2720	34940	30175.0
16	9440	187790	35001.0
17	2720	58610	35752.0
18	1540	34940	54046.0
19	9520	96350	62199.0
20	65510	108230	77149.0
21	1540	71200	78198.0
22	1540	2720	88803.0
23	640	71200	95427.0
24	83550	131030	102123.0

25	520	58610	151070.0
26	78600	222080	194639.0
27	3850	33500	243484.0
28	640	222080	267431.0
29	520	7610	304636.0
30	4690	22100	454843.0
31	4690	71320	474749.0
32	640	22100	648445.0
33	6400	2700	662973.0
34	640	71320	663507.0
35	640	66570	764606.0
36	6400	100090	1774205.0

Pair trading

5. 한계

- 페어 트레이딩 전략을 백테스팅하는 패키지 존재 X
 - 다른 모델들과 결합하기 힘들
 - 독자적으로 사용되었을 때도 성능을 평가하는데 한계
- * 개인이 페어 트레이딩 전략사용하는 것은 현실적으로 불가능

Strategy 3

Network Architecture Search with Weight Agnostic Neural Networks

“Find the best network architecture which determines Buy/Sell/Hold”

Find the best architecture with Weight Agnostic Neural Networks



대한민국 환경부 등급 기준 ⓘ

☁ 10° >



매우나쁨



데이터 없음



Buy
Sell
Hold

Find the best architecture with Weight Agnostic Neural Networks



대한민국 환경부 등급 기준 ⓘ

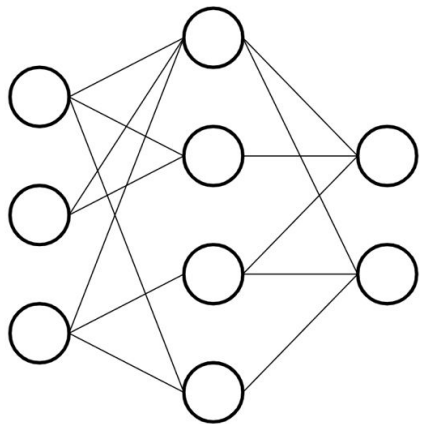
☁ 10° >



매우나쁨



데이터 없음

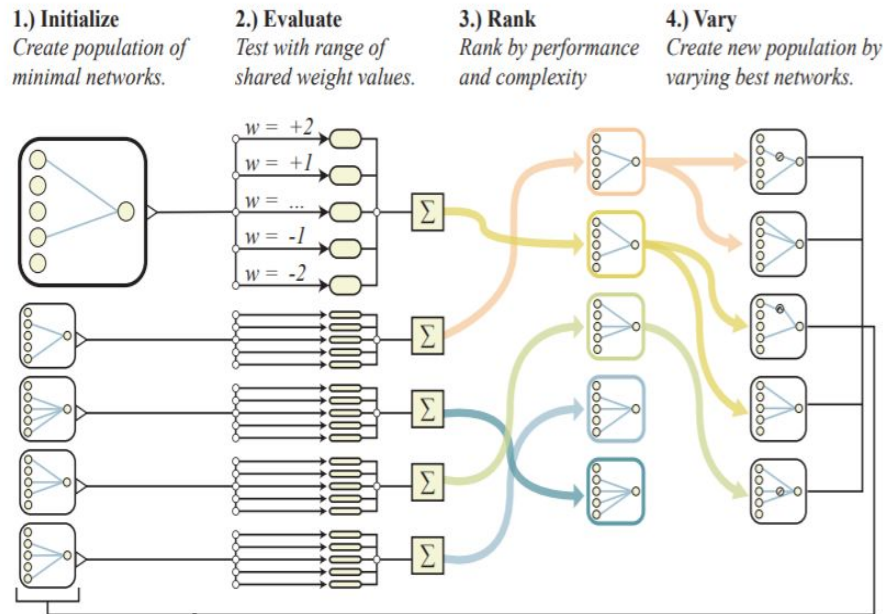


Best Network Architecture?

Buy
Sell
Hold

WANN Topology Search (NEAT Algorithm)

1. Input Node와 Output Node가 p의 확률로 연결된 초기 Generation 생성
2. 모든 Connection을 동일한 weight(-2, -1, -0.5, 0.5, 1, 2)로 설정하고 Backtesting하여 Train Set Return, Valid Set Return을 출력
3. Train Set Return을 기준으로 Rank를 매겨 상위 3개 개체만 남김
4. Add Node / Add Connection / Change Activation을 확률적으로 수행하여 다음 세대 생성
5. Valid Set Return이 감소하기 시작하면 종료 (Overfit되기 시작하는 시점)



graphtorch

KU-BIG / **graphtorch**

graphtorch

Package converts sparse graph matrix to PyTorch model

Installation

```
pip install graphtorch
```

not uploaded yet

Examples

Create sparse matrix with essential information

```
from graphtorch import SparseMatrix

mat1 = np.array([[0,2,0,0,2,0,0,0,0,0],
                 [2,0,2,0,0,0,0,0,0,0],
                 [0,2,0,2,0,0,0,0,0,0],
                 [0,0,0,0,1,1,0,0,0,0],
                 [0,0,0,0,0,1,1,0,0,0],
                 [0,0,0,0,0,0,0,3,0,0],
                 [0,0,0,0,0,0,0,0,3,0]])

in_dim = 5
out_dim = 2
mat_wann1 = SparseMatrix(mat1, in_dim, out_dim)
```

- Pytorch는 Sparse한 NN을 만들어주지 않음
- 개별 Connection마다 Activation을 다르게 설정해줄수 없는 한계

**Sparse Graph를 표현하는 Matrix를
받아서 Pytorch model로 만들어주는
패키지가 있을까?**

graphtorch

KU-BIG / graphtorch

graphtorch

Package converts sparse graph matrix to PyTorch model

Installation

```
pip install graphtorch
```

not uploaded yet

Examples

Create sparse matrix with essential information

```
from graphtorch import SparseMatrix

mat1 = np.array([[0,2,0,0,2,0,0,0,0,0],
                 [2,0,2,0,0,0,0,0,0,0],
                 [0,2,0,2,0,0,0,0,0,0],
                 [0,0,0,0,1,1,0,0,0,0],
                 [0,0,0,0,0,1,1,0,0,0],
                 [0,0,0,0,0,0,0,3,0,0],
                 [0,0,0,0,0,0,0,0,3,0]])

in_dim = 5
out_dim = 2
mat_wann1 = SparseMatrix(mat1, in_dim, out_dim)
```

- Pytorch는 Sparse한 NN을 만들어주지 않음
- 개별 Connection마다 Activation을 다르게 설정해줄수 없는 한계

Sparse Graph를 표현하는 Matrix를 받아서 Pytorch model로 만들어주는 패키지가 있을까?

-> 없음

graphtorch

KU-BIG / graphtorch

graphtorch

Package converts sparse graph matrix to PyTorch model

Installation

```
pip install graphtorch
```

not uploaded yet

Examples

Create sparse matrix with essential information

```
from graphtorch import SparseMatrix

mat1 = np.array([[0,2,0,0,2,0,0,0,0,0],
                 [2,0,2,0,0,0,0,0,0,0],
                 [0,2,0,2,0,0,0,0,0,0],
                 [0,0,0,0,1,1,0,0,0,0],
                 [0,0,0,0,0,1,1,0,0,0],
                 [0,0,0,0,0,0,0,3,0,0],
                 [0,0,0,0,0,0,0,0,3,0]])

in_dim = 5
out_dim = 2
mat_wann1 = SparseMatrix(mat1, in_dim, out_dim)
```

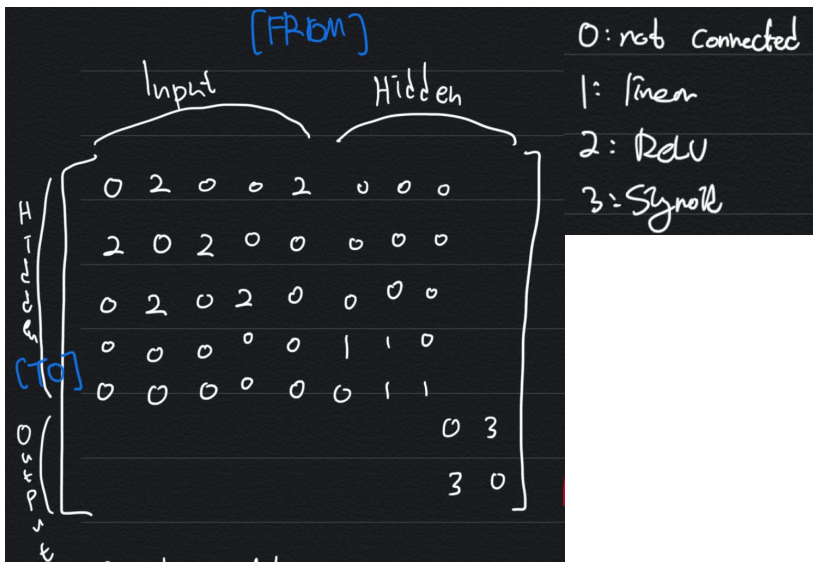
- Pytorch는 Sparse한 NN을 만들어주지 않음
- 개별 Connection마다 Activation을 다르게 설정해줄수 없는 한계

Sparse Graph를 표현하는 Matrix를 받아서 Pytorch model로 만들어주는 패키지가 있을까?

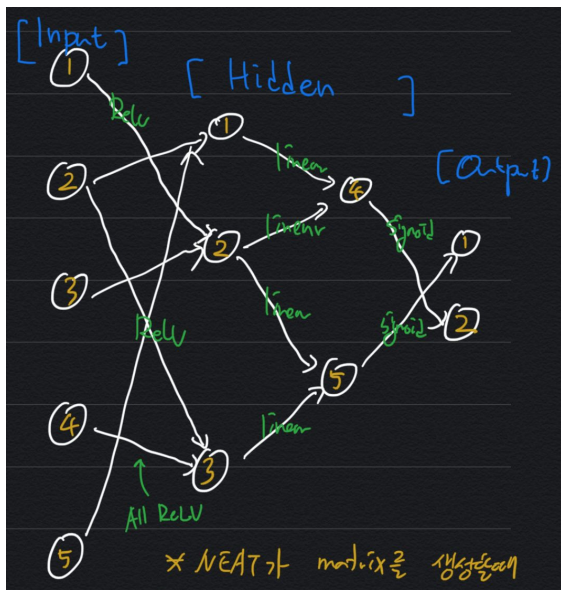
-> 그래서 만듦

graphtorch

class SparseMatrix()



class SparseModel()



graphtorch

class SparseMatrix()

Create sparse matrix with essential information

```
from graphtorch import SparseMatrix

mat1 = np.array([[0,2,0,0,2,0,0,0,0,0],
                 [2,0,2,0,0,0,0,0,0,0],
                 [0,2,0,2,0,0,0,0,0,0],
                 [0,0,0,0,0,1,1,0,0,0],
                 [0,0,0,0,0,0,1,1,0,0],
                 [0,0,0,0,0,0,0,0,0,3],
                 [0,0,0,0,0,0,0,0,3,0]])

in_dim = 5
out_dim = 2
mat_wann1 = SparseMatrix(mat1, in_dim, out_dim)
```

class SparseModel()

Create sparse torch model with SparseMatrix

```
from graphtorch import SparseModel

activations = [None, None, nn.ReLU(), nn.Sigmoid()]
constant_weight = 1
model = SparseModel(mat_wann1, activations, constant_weight)

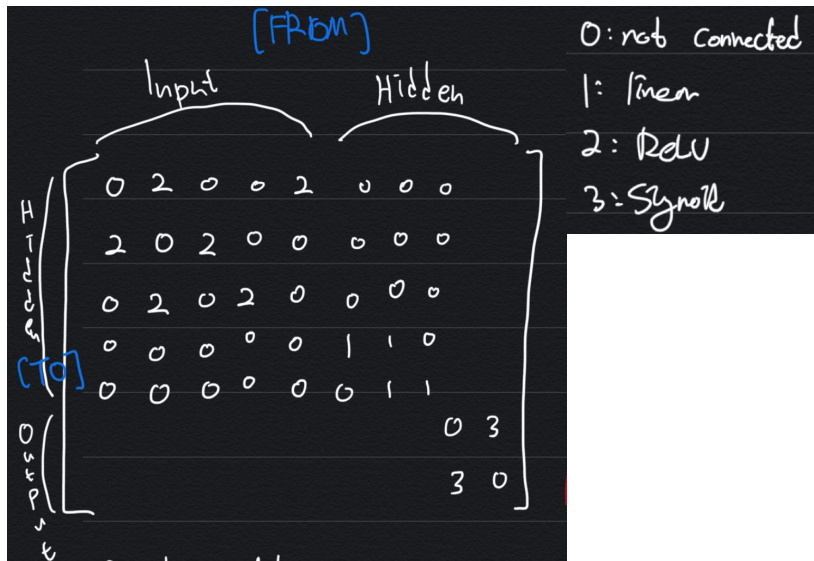
numpy_input = np.array([[1,2,3,4,5],
                        [6,7,8,9,10],
                        [11,12,13,14,15]])

numpy_input = torch.from_numpy(numpy_input).float()
output, nodes = model(numpy_input)
```


graphtorch

class SparseMatrix()

class SparseMatrix()



Create sparse matrix with essential information

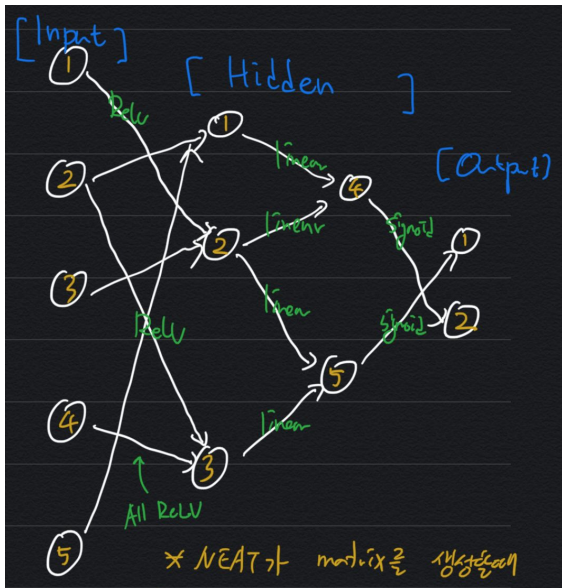
```
from graphtorch import SparseMatrix

mat1 = np.array([[0,2,0,0,2,0,0,0,0,0],
                 [2,0,2,0,0,0,0,0,0,0],
                 [0,2,0,2,0,0,0,0,0,0],
                 [0,0,0,0,0,1,1,0,0,0],
                 [0,0,0,0,0,0,1,1,0,0],
                 [0,0,0,0,0,0,0,0,0,3],
                 [0,0,0,0,0,0,0,0,3,0]])

in_dim = 5
out_dim = 2
mat_wann1 = SparseMatrix(mat1, in_dim, out_dim)
```

graphtorch

class SparseModel()



class SparseModel()

Create sparse torch model with SparseMatrix

```
from graphtorch import SparseModel

activations = [None, None, nn.ReLU(), nn.Sigmoid()]
constant_weight = 1
model = SparseModel(mat_wann1, activations, constant_weight)

numpy_input = np.array([[1,2,3,4,5],
                        [6,7,8,9,10],
                        [11,12,13,14,15]])

numpy_input = torch.from_numpy(numpy_input).float()
output, nodes = model(numpy_input)
```

NEAT Algorithm

1.) Initialize

Create population of minimal networks.

2.) Evaluate

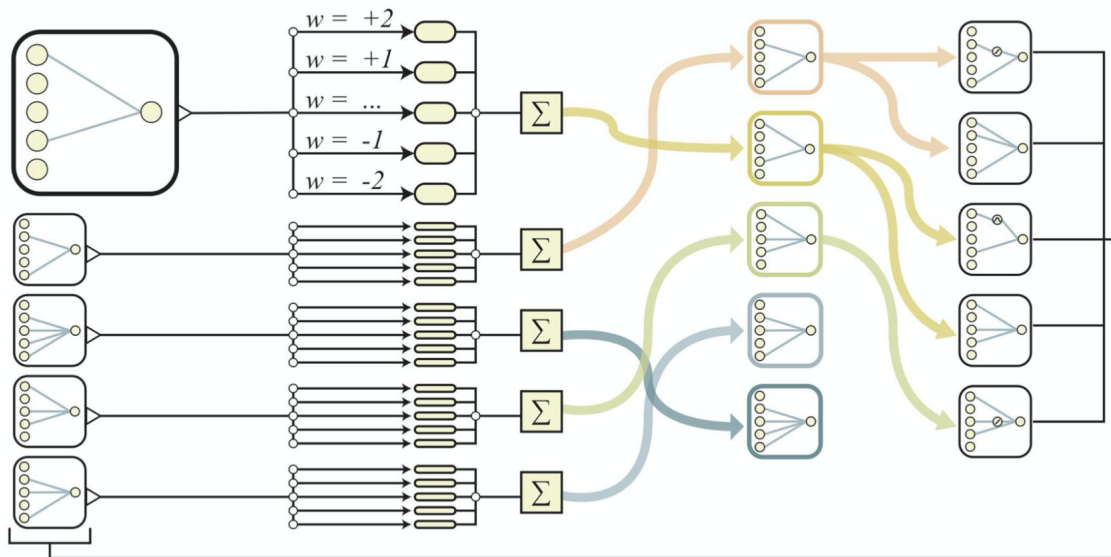
Test with range of shared weight values.

3.) Rank

Rank by performance and complexity

4.) Vary

Create new population by varying best networks.



누군가 이걸 만들어냈을까?

NEAT Algorithm

1.) Initialize

Create population of minimal networks.

2.) Evaluate

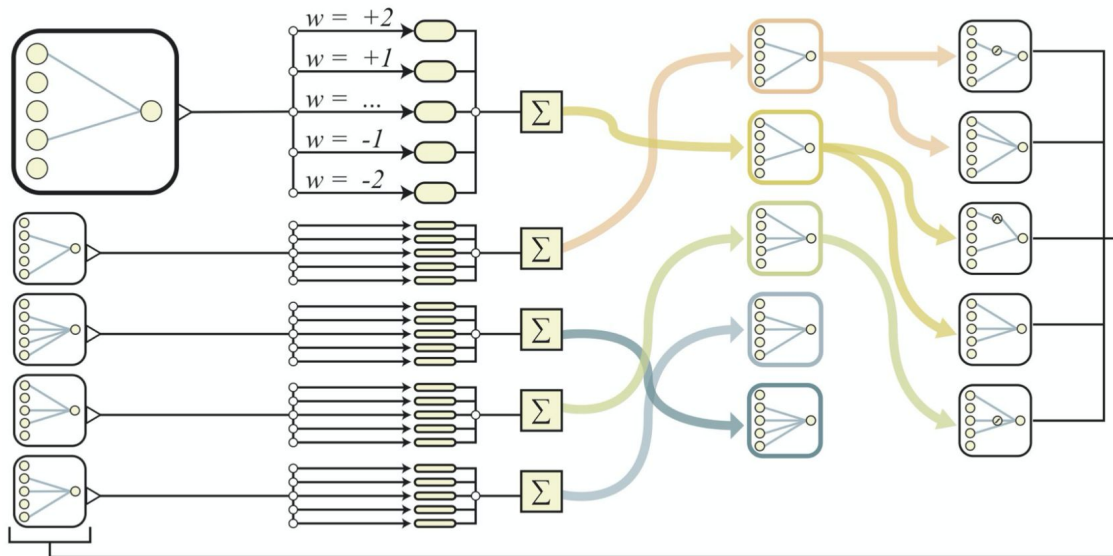
Test with range of shared weight values.

3.) Rank

Rank by performance and complexity

4.) Vary

Create new population by varying best networks.



누군가 이걸 만들어냈을까?

헛된 기대를 가지는 것보단

직접 짜는게 더 빠름

NEAT Algorithm

1.) Initialize

Create population of minimal networks.

2.) Evaluate

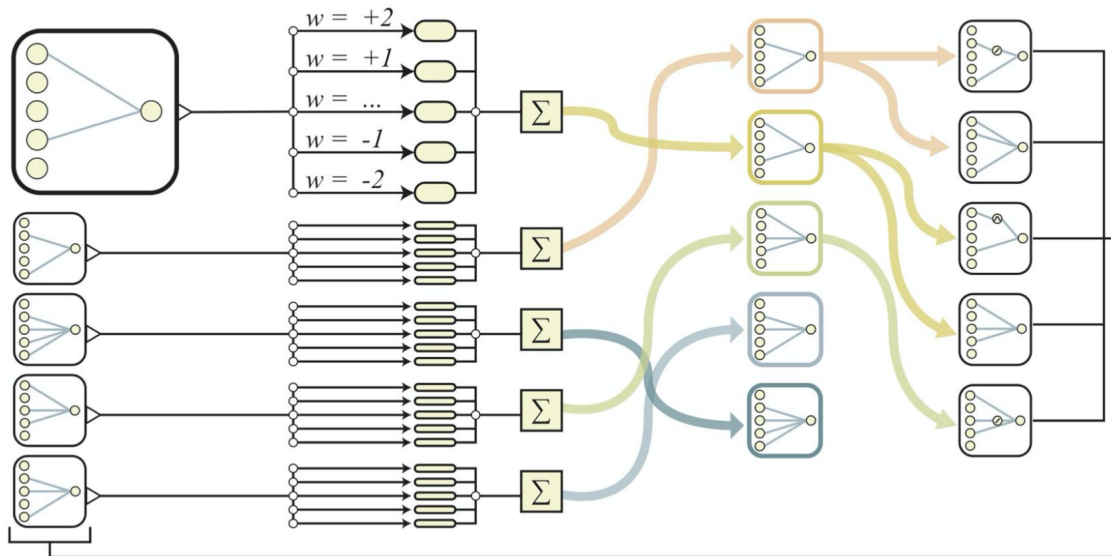
Test with range of shared weight values.

3.) Rank

Rank by performance and complexity

4.) Vary

Create new population by varying best networks.



```
def create_first_generation()
```

```
def get_fitness_value()
```

```
def change_activation()
```

```
def add_connection()
```

```
def add_node()
```

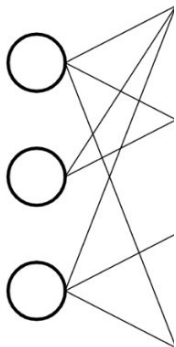
저거 두개 짜는데 6시간걸림

코딩테스트 면제시켜줘야해...

궁금하면

<https://github.com/KU-BIG/graphtorch>

Desired Input/Output

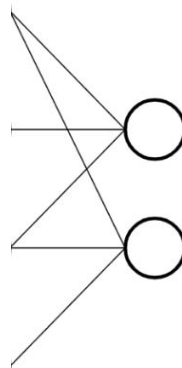


Input Dimension : 11

금융데이터 : Open, Close, ...

미세먼지 데이터 : PM2.5, PM10,

... ? ...



Output Dimension : 3

Softmax : Buy/Sell/Hold

Result

	matrix	score_train	score_valid	generation	child
0	<graphtorch.SparseMatrix object at 0x7fcbcd998...	-550.0	-550.0	0	0
1	<graphtorch.SparseMatrix object at 0x7fcbcd998...	-550.0	-100.0	0	1
2	<graphtorch.SparseMatrix object at 0x7fcbcd998...	-550.0	-550.0	0	2
3	<graphtorch.SparseMatrix object at 0x7fcbcd998...	-1000.0	-550.0	0	3
4	<graphtorch.SparseMatrix object at 0x7fcbcd998...	-1000.0	-100.0	0	4
5	<graphtorch.SparseMatrix object at 0x7fcbcd8bf...	-1000.0	-550.0	1	0
6	<graphtorch.SparseMatrix object at 0x7fcbcd8bf...	-1000.0	-550.0	1	1
7	<graphtorch.SparseMatrix object at 0x7fcbcd998...	-1000.0	-100.0	1	2
8	<graphtorch.SparseMatrix object at 0x7fcbcd8f1...	-1000.0	-100.0	1	3
9	<graphtorch.SparseMatrix object at 0x7fcbcd8f1...	-1000.0	-550.0	1	4
10	<graphtorch.SparseMatrix object at 0x7fcbcd909...	-1000.0	-550.0	2	0
11	<graphtorch.SparseMatrix object at 0x7fcbcd84b...	-1000.0	-550.0	2	1
12	<graphtorch.SparseMatrix object at 0x7fcbcd84b...	-1000.0	-550.0	2	2
13	<graphtorch.SparseMatrix object at 0x7fcbcd84b...	-1000.0	-550.0	2	3
14	<graphtorch.SparseMatrix object at 0x7fcbcd84b...	-1000.0	-550.0	2	4
15	<graphtorch.SparseMatrix object at 0x7fcbcd7c9...	-1000.0	-550.0	3	0
16	<graphtorch.SparseMatrix object at 0x7fcbcd868...	-1000.0	-550.0	3	1
17	<graphtorch.SparseMatrix object at 0x7fcbcd7cd...	-1000.0	-550.0	3	2
18	<graphtorch.SparseMatrix object at 0x7fcbcd7cd...	-1000.0	-550.0	3	3
19	<graphtorch.SparseMatrix object at 0x7fcbcd770...	-1000.0	-1000.0	3	4
20	<graphtorch.SparseMatrix object at 0x7fcbcd761...	-550.0	-550.0	4	1
21	<graphtorch.SparseMatrix object at 0x7fcbcd761...	-550.0	-550.0	4	2
22	<graphtorch.SparseMatrix object at 0x7fcbcd754...	-550.0	-550.0	4	3
23	<graphtorch.SparseMatrix object at 0x7fcbcd754...	-550.0	-550.0	4	4
24	<graphtorch.SparseMatrix object at 0x7fcbcd761...	-1000.0	-550.0	4	0



Strategy 4

Price-predicting Model

“Train the model which predicts n days after closing price”

6-1. Backtesting 결과 (이화산업 000760)

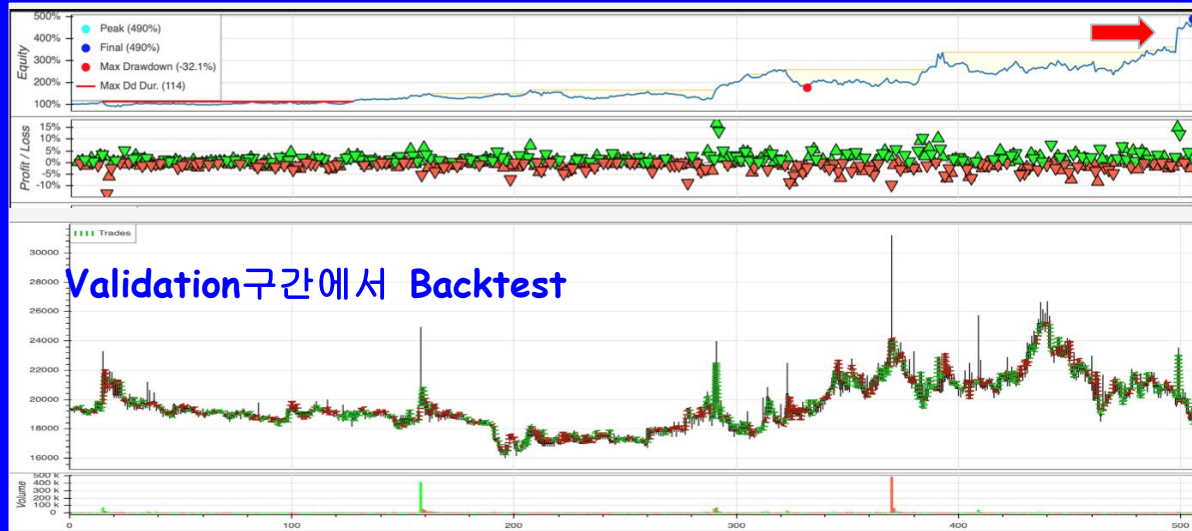


Price Prediction

- Training Period : 2013.08.01 ~ 2016.12.31
- Validation Period : 2017.01.01 ~ 2018.12.31
- Training R2 Score : 0.9987
- Validation R2 Score : 0.7461

Backtesting Result

- End Equity : 490%
- Maximal Drawdown : -32.1%
- Sortino Ratio : 0.15
- Sharpe Ratio : 0.10
- Exposure : 99.67%
- Win Rate : 49.01%



Price Predicting Model

```
# hyperparameter
```

```
time_shift = -1
```

```
x_train = x[x.index < datetime.datetime(2018,10,1)].values  
x_valid = x[x.index >= datetime.datetime(2018,10,1)].values  
y_train = y[y.index < datetime.datetime(2018,10,1)].values  
y_valid = y[y.index >= datetime.datetime(2018,10,1)].values  
others_train = others[others.index < datetime.datetime(2018,10,1)]  
others_valid = others[others.index >= datetime.datetime(2018,10,1)]
```

```
# model
```

```
model = lgbm(num_threads=num_threads)  
#model = xgb(nthread=num_threads)  
#model = cat(thread_count=num_threads, verbose=0)
```

```
model.fit(x_train, y_train)
```

```
y_train_hat = model.predict(x_train)  
y_valid_hat = model.predict(x_valid)
```

1일 이후를 예측하도록 설계
-> time shift = -1

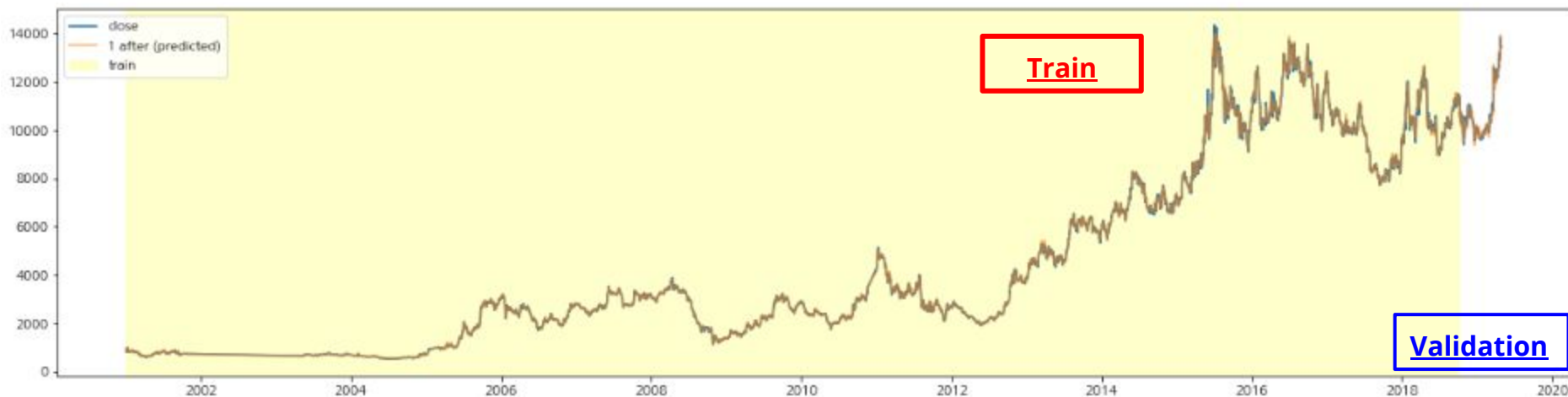


train/valid 데이터 설정



lgbm 사용 -> predict

Price Predicting Model



이제 본격적인 Backtesting 진행!

Price Predicting Model

```
data = pd.DataFrame(x_valid, columns=x_columns)
data = data.merge(others_valid, how='inner')
yhat = model.predict(data.drop(['Open', 'High', 'Low'], axis=1))
data['time shifted'] = yhat
```

data

	Close	Volume	PM10	Open	High	Low	time shifted	date
0	11400.0	273623.0	16.083333	11600	11800	11300	11283.606492	2018-10-01
1	11250.0	268640.0	21.458333	11350	11500	11050	11058.408351	2018-10-02
2	11450.0	220187.0	19.083333	11300	11500	11150	11437.476481	2018-10-04
3	11100.0	174180.0	10.291667	11350	11400	10950	11302.898090	2018-10-05
4	11400.0	152633.0	20.541667	11100	11550	11000	11441.288075	2018-10-08
...
137	12900.0	353524.0	76.291667	13150	13200	12800	12979.941417	2019-04-23
138	13150.0	455360.0	47.791667	13000	13350	12950	13222.269082	2019-04-24
139	13850.0	1148812.0	32.545455	13250	13900	13200	13785.635672	2019-04-25
140	13550.0	663063.0	8.500000	13800	14100	13400	13934.538890	2019-04-26
141	13450.0	2970765.0	21.166667	14250	15400	13200	13685.983583	2019-04-29

Close, 미세먼지 데이터
활용해서 yhat predict



Backtesting에 활용할
Dataset 완성!!

Price Predicting Model

```
class SmaCross(Strategy):  
  
    def init(self):  
        print(self.data)  
        # 한줄씩 호출되는듯  
        #self.data = data  
        self.close = self.data['Close']  
        self.time_shifted_price = self.data['time shifted']  
        self.daycount = 0  
        self.hold = False  
  
    def next(self):  
        if (self.close < self.time_shifted_price) and (self.hold == False):  
            self.buy()  
            self.hold = True  
            self.daycount = 0  
  
            elif (self.close >= self.time_shifted_price) and (self.hold == True):  
                self.sell()  
                self.hold = False  
  
            elif (self.daycount > 2):  
                self.sell()  
                self.hold=False  
  
        self.daycount += 1
```

Strategy 적용:

- close 값이 예측된 close값보다 작고, hold하고 있지 않으면 -> 산다.
- close 값이 예측된 close값보다 크고, hold하고 있으면 -> 판다.
- 이틀보다 주식을 많이 가지고 있다면 -> 판다.
[time shift=-1: 1일 후를 예측하는 것이기 때문]
- 위 세가지 중 한가지를 수행하고 난 후, daycount를 1일 추가한다.

Price Predicting Model

```
bt = Backtest(data, SmaCross, cash=10000, commission=0)
```

```
C:\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: UserWarning: Entry point for launching an IPython kernel.
```

```
bt.run()
```

```
<backtesting._util._Data object at 0x0000015864DB2EC8>
```

Start	0
End	141
Duration	141
Exposure [%]	98.5816
Equity Final [\$]	17963.3
Equity Peak [\$]	17963.3
Return [%]	79.633
Buy & Hold Return [%]	17.9825
Max. Drawdown [%]	-15.0389
Avg. Drawdown [%]	-4.92309
Max. Drawdown Duration	43
Avg. Drawdown Duration	10.6667

# Trades	69
Win Rate [%]	52.1739
Best Trade [%]	10.5023
Worst Trade [%]	-7.12871
Avg. Trade [%]	0.78731
Max. Trade Duration	3
Avg. Trade Duration	2.01449
Expectancy [%]	2.28661
SQN	2.20225
Sharpe Ratio	0.250499
Sortino Ratio	0.532752
Calmar Ratio	0.0523517
_strategy	SmaCross
dtype:	object

Price Predicting Model

```
bt = Backtest(data, SmaCross, cash=10000, commission=0)
```

```
C:\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: UserWarning: Entry point for launching an IPython kernel.
```

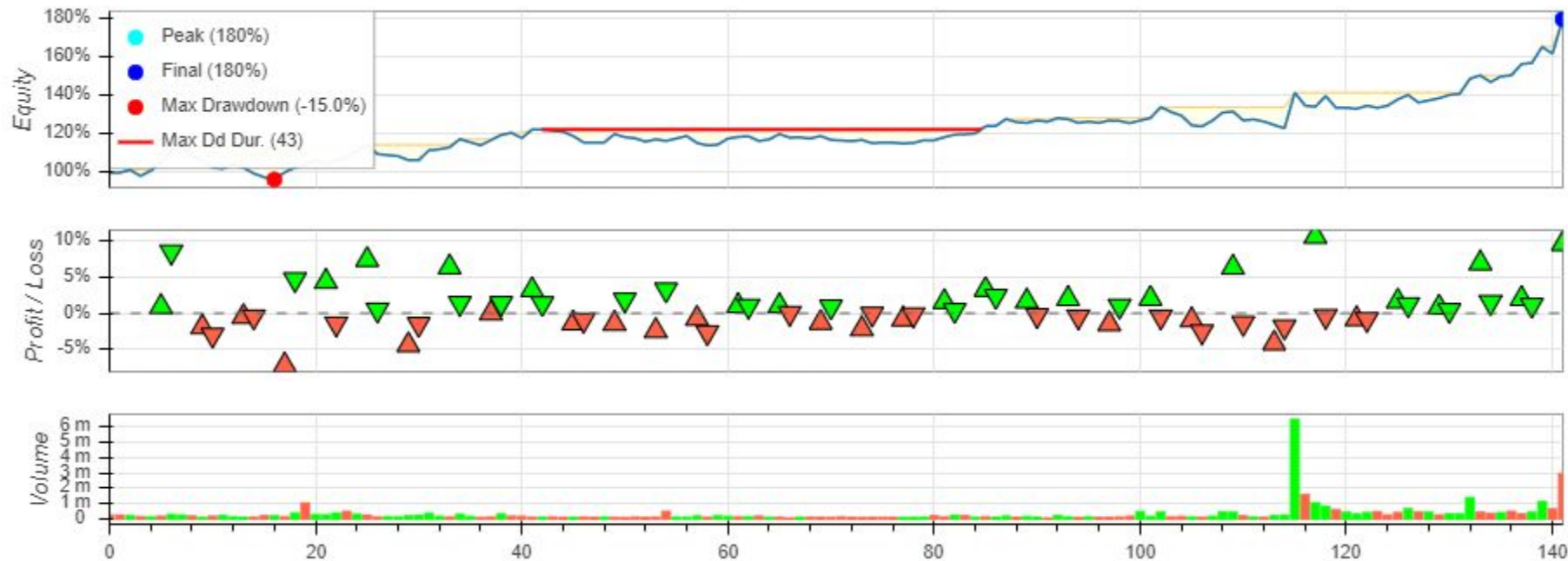
```
bt.run()
```

```
<backtesting._util._Data object at 0x0000015864DB2EC8>
```

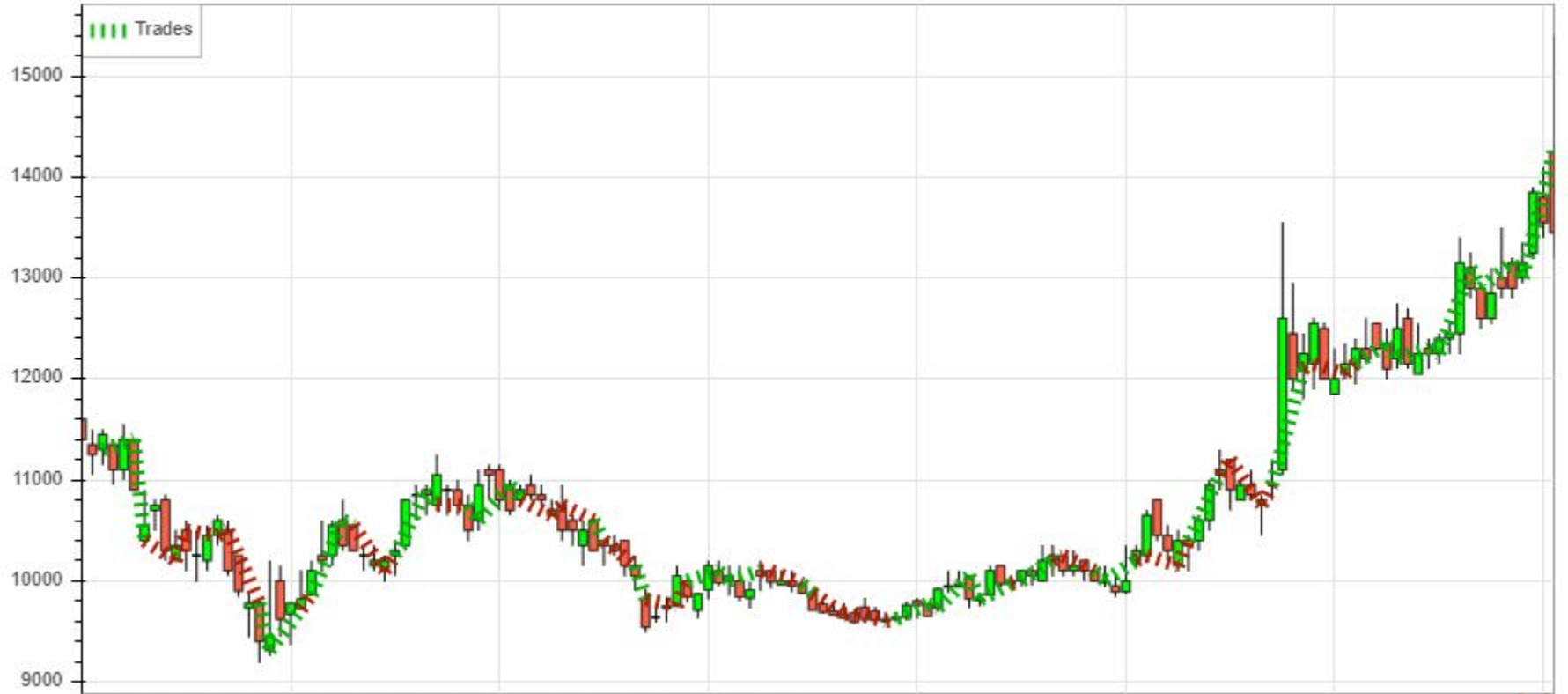
Start	0
End	141
Duration	141
Exposure [%]	98.5816
Equity Final [\$]	17963.3
Equity Peak [\$]	17963.3
Return [%]	79.633
Buy & Hold Return [%]	17.9825
Max. Drawdown [%]	-15.0389
Avg. Drawdown [%]	-4.92309
Max. Drawdown Duration	43
Avg. Drawdown Duration	10.6667

# Trades	69
Win Rate [%]	52.1739
Best Trade [%]	10.5023
Worst Trade [%]	-7.12871
Avg. Trade [%]	0.78731
Max. Trade Duration	3
Avg. Trade Duration	2.01449
Expectancy [%]	2.28661
SQN	2.20225
Sharpe Ratio	0.250499
Sortino Ratio	0.532752
Calmar Ratio	0.0523517
_strategy	SmaCross
dtype:	object

Price Predicting Model



Price Predicting Model



Price Predicting Model

- 한계: 몇몇 종목에서만 좋은 수익률을 보인다.

	003850.csv	009440.csv	067170.csv	071840.csv	002720.csv
Start	0	0	0	0	0
End	141	141	141	141	141
Duration	141	141	141	141	141
Exposure	98.58156	98.58156	98.58156	96.4539	98.58156
Equity Final	17963.3	14222.13	13674.45	13160.94	12819.29
Equity Peak	17963.3	17639.79	16528.4	14244.58	17169.1
Return [%]	79.63301	42.22131	36.74448	31.60941	28.19287

1. 적절한 전략이 아니라서?
2. Backtesting 자체의 한계라서?
3. 미세먼지와 관련이 없는 주식이라서?
4. etc..

095190.csv	066130.csv	009520.csv	011320.csv	119650.csv	126880.csv
0	0	0	0	0	0
141	141	141	141	141	141
141	141	141	141	141	141
96.4539	96.4539	98.58156	96.4539	96.4539	98.58156
4390.912	4108.637	4035.896	3099.767	3044.284	2572.33
12896.54	13705.32	13297.78	13689.84	11202.43	11318.88
-56.0909	-58.9136	-59.641	-69.0023	-69.5572	-74.2767
62.93785	56.41026	20.34314	63.39468	113.981	57.46367
-73.9546	-77.3242	-70.7317	-84.628	-74.3417	-79.1844

Experimental Result

Experimental Result

강화학습	Pair-trading	WANN	Profit-Predict model
수익을 굉장히 낮음	알려진 바와 같이 위험 최소화, 수익을 남	굉장히 복잡한 결과	수익을 있으나, 종목별로 다름

Discussion and Future Work

Discussion and Future Work

- **Strategy 1**

- 강화학습은 알고리즘 트레이딩 모델로 적합하지 않다.
- 하지만 모듈이 개발중에 있으므로 발전가능성이 있다.

- **Strategy 2**

- Pair trading을 하면 정말로 안정적으로 수익을 창출해낼수 있을까? Y
- 하지만 패키지의 부재로 다른 모델들과의 결합이나, 깊이있는 분석에는 한계
- 다른 모델들과 결합시키는 방식을 연구해 볼 필요
- 깊이있는 분석을 위해 주식에 대한 공부가 필요

- **Strategy 3**

- Sparse Graph를 표현하는 Matrix를 받아서 Pytorch model로 만들어주는 패키지인 graphtorch를 새로 만듦
- 하지만 아직도..

- **Strategy 4**

- 다양한 전략 테스트하고, 가장 높은 수익률이 나는 전략을 채택했으나 최적의 전략이라고 보장할 수는 없음.
- 수익률이 크게 나는 종목도 있지만, 되려 높은 손해를 내기도 한다.(불안정성)
- 보다 더 안정적인 모델을 찾아 나가야 할 것.

감사합니다.