

개인 사정으로 오랫동안 글을 쓰지 못했다. 그동안 금융 이론과 딥러닝을 결합하는 문제에 관심이 많았는데 정리할 시간을 갖지 못했다. 앞으로는 시간이 날 때마다 조금씩이라도 정리해 보기로 하겠다.

목차

- 1) 마코비츠-네트워크 (1) : MPN 개요
- 2) 마코비츠-네트워크 (2) : MPN 학습 (teacher forcing)
- 3) 마코비츠-네트워크 (3) : 성능 평가
- 4) 마코비츠-네트워크 (4) : Metric
- 5) 동영상 강의 (유료) : [Insight Campus](#)
- 6) 마코비츠-네트워크 (5) : 성능개선 (트랜스포머와 추가학습)

첫 번째 주제로 마코비츠의 포트폴리오 이론과 딥러닝 기술 (LSTM, CNN, Transformer 정도 ?)을 결합하는 문제를 다뤄 보기로 한다. 마코비츠-네트워크를 구성해서 미래의 최적 포트폴리오를 구성하고, 그 성과를 평가해 보기로 한다.

금융 이론과 딥러닝 기술을 개별적으로 따로따로 기술하려면 내용이 너무 많아지기 때문에, 개별적인 내용은 모두 알고 있다고 가정하고, 두 기술을 결합하는 방법에 집중하기로 한다.

마코비츠의 포트폴리오 이론 (Markowitz, 1952, Portfolio selection theory)은 여러 위험 자산들로 포트폴리오를 구성할 때 특정 목표 함수를 극대화 시키도록 각 자산의 최적 투자 비율을 결정하는 것과 관련이 있다. 이런 포트폴리오를 구성하면 자산들의 개별 위험 (비체계적 위험)을 줄일 수 있다. 목표 함수로는 위험조정수익률 (Sharp, Treynor 지수 등)을 사용한다.

$$\begin{aligned}
 R &= \{r_1, r_2, \dots, r_N\} && \xrightarrow{\hspace{10em}} \text{(1) N개 자산에 대한 개별 기대수익률} \\
 W &= \{w_1, w_2, \dots, w_N\} && \xrightarrow{\hspace{10em}} \text{(2) N개 자산에 대한 투자 비율} \\
 r_p &= W \cdot R && \xrightarrow{\hspace{10em}} \text{(3) 포트폴리오의 기대수익률} \\
 \sigma_p^2 &= W \cdot C \cdot W^T && \xrightarrow{\hspace{10em}} \text{(4) 포트폴리오의 기대분산} \\
 \underset{W}{\operatorname{Max}} \sum [W \cdot R - \gamma W \cdot C \cdot W^T - \lambda \|W\|^2] & && \xrightarrow{\hspace{10em}} \text{(5) 목표 함수} \\
 W_i > 0, \sum W = 1 & && \xrightarrow{\hspace{10em}} \text{(6) 공매도를 사용하지 않는 경우의 제한 조건} \\
 \sum W = 0 & && \xrightarrow{\hspace{10em}} \text{(7) 공매도를 사용할 경우의 제한 조건}
 \end{aligned}$$

마코비츠의 이론을 정리하면 위의 수식과 같다. N개의 위험 자산에 대해 식 (5)의 목표 함수를 극대화 시키는 최적 투자 비율 (2)를 추정할 수 있다. 그러면 이 포트폴리오의 기대수익률과 위험의 척도인 분산은 식 (3)과 (4)로 측정된다. 최적화를 위한 목표 함수로는 Sharp ratio 등을 사용할 수도 있지만, 여기서는 식 (5)의 수식을 사용하기로 한다.

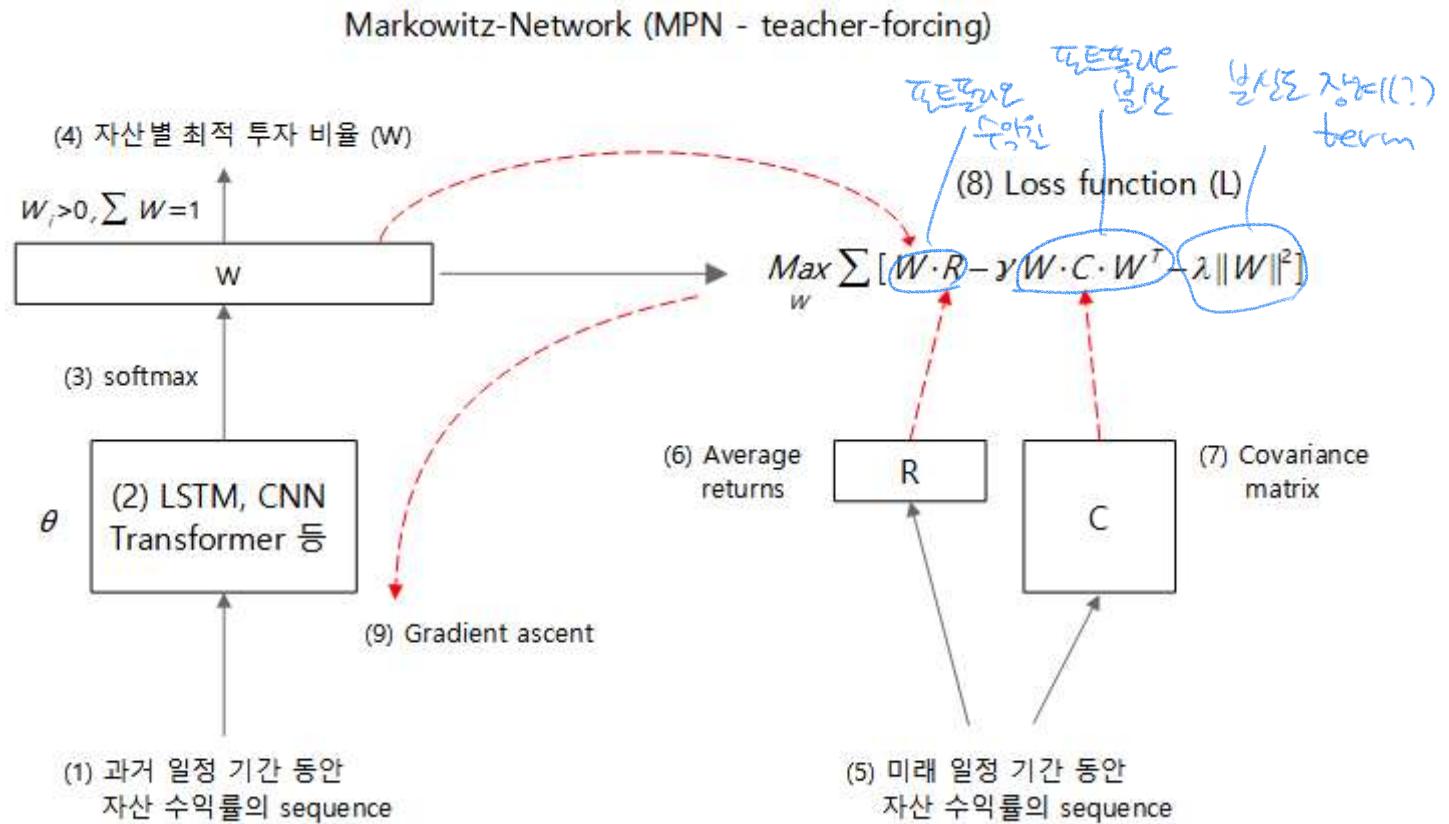
식 (5)에서 γ (감마)는 위험에 대한 비중을 조절하는 상수다. γ 를 크게 지정하면 위험을 크게 고려해서 기대수익률은 작지만 안전한 포트폴리오를 구축하려는 의도이고, γ 를 작게 지정하면 위험은 크게 고려하지 않고 기대수익률을 높이려는 의도이다. λ (람다)는 regularization의 크기를 조절하는 상수다. λ 를 크게 설정하면 W 가 특정 종목에 치우치지 않고 고르게 분포한다 (over confidence 방지 목적). 반대로 λ 를 너무 작게 설정하면 과거에 실적이 좋았던 자산들을 과도하게 매입하는 현상이 발생한다.

식 (6)과 (7)은 제한 조건이다. 식 (6)은 공매도를 사용하지 않는 경우이며 개별 자산의 W 는 모두 0보다 커야 하고, W 의 총합은 1이 돼야 한다. 식 (7)은 공매도를 사용하는 경우이며 개별 자산의 W 는 음수가 허용되고 W 의 총합은 0이 돼야 한다 (zero-investment portfolio). 여기서는 공매도를 사용하지 않는 경우를 다룬다.

마코비츠의 포트폴리오 이론은 이론적으로는 완벽하다. 그러나 기대수익률과 기대분산을 추정해야 하는 문제 때문에 현실에 적용하기가 어렵다. 개별 자산들의 기대수익률과 기대분산을 예측하는 것은 어렵다. 어쩌면 불가능할지도 모른다. 게다가 금융 이론의 기본은 정규분포를 기반으로 한 랜덤워크이기 때문에 금융 이론과 예측은 서로 어울리지 않는다.

만약 실제 시장의 모습이 완전한 랜덤워크가 아니라면 과거 데이터와 미래의 최적 투자 비율인 W 는 어떤 형태로든 관련이 있기 때문에 W 를 추정할 수 있다. 물론 자산들의 개별 기대수익률이나 분산을 일일이 예측해서 W 를 추정하는 것은 어렵다. 그럼 어떻게 (미래의) 기대수익률을 추정하지 않고 W 를 추정할 수 있을까? 바로 이 부분에서 딥러닝과의 연결 고리를 맺어볼 수 있다.

딥러닝은 데이터를 학습해서 그 안에 존재하는 다양한 패턴들을 분석하는 데 뛰어난 능력을 발휘한다. 딥러닝은 학습을 통해 직접 경험한 것 뿐만 아니라, 경험해보지 못한 것들을 추론하는 능력도 뛰어나다. 아래 그림은 마코비츠의 이론을 딥러닝 네트워크로 구현해본 예시다. 아래 모델을 마코비츠-네트워크라 하고, 간단히 줄여서 MPN (Markowitz's Portfolio Network)이라 하겠다.



MPN은 기본적으로 비지도학습 (unsupervised learning)의 형태다. 입력과 출력, 그리고 loss function인 L 을 정의하고, 주어진 입력에 대해 L 이 극대화되도록 네트워크를 학습 시킨다. 입력은 일정 기간의 과거 데이터 (1)과, 그 시점 이후의 미래 데이터 (5)이다. 출력은 자산들의 최적 투자 비율인 W (4)이다. 세부 학습 절차는 아래와 같다.

Step-1. LSTM, CNN, Transformer 등의 딥러닝 네트워크를 구성하고, 파라미터 θ (세타)를 랜덤 초기화 한다.

Step-2. 입력 데이터 (과거)를 딥러닝 네트워크에 입력한다. 위 그림의 (1), (2) 과정.

Step-3. 딥러닝 네트워크는 파라미터 θ (세타)를 기반으로 W 를 출력한다. W 를 출력할 때 softmax 함수를 사용하면 마코비츠 이론의 제한조건을 충족 시킬 수 있다. 위 그림의 (3)과 (4) 과정.

Step-4. 입력 데이터 (미래)를 이용해서 평균 실현수익률 (R)과 공분산 행렬 (C)를 계산한다. 위 그림의 (5), (6), (7) 과정.

Step-5. 딥러닝 네트워크의 출력인 W 와, step-4에서 계산한 R 과 C , 그리고 사전에 설정한 γ (감마)와 λ (람다)를 이용해서 loss function인 L 을 계산한다. 위 그림의 (8) 과정.

Step-6. Loss function L 의 gradient를 계산해서 gradient ascent 알고리즘으로 딥러닝 네트워크의 파라미터 θ 를 업데이트 한다. 위 그림의 (9) 과정.

Step-6까지 과정을 거치면 파라미터 θ 는 의미가 있는 방향으로 조금 변하고, loss L 의 크기도 조금 커질 것이다. L 이 더 이상 커지지 않을 때까지 step-2부터 step-6까지 과정을 계속 반복한다. 그러면 loss L 을 극대화시키는 파라미터 θ 가 결정될 것이다 (파라미터 θ 를 결정하는 것이 목적이).

이 절차에는 (과거 시점 이후의) 미래 데이터가 사용됐다. 이런 구조를 teacher forcing이라 한다 (미래 데이터는 학습 시에만 사용된다). 학습이 완료된 후에는 좌측의 딥러닝 네트워크만 사용한다. 딥러닝 네트워크에 현재의 데이터를 입력하면 학습이 완료된 파라미터 θ 를 기반으로 어떤 W 를 출력하는데, 이것은 미래의 L 값을 극대화시키는 W 라는 의미다. 즉, 현재 시점에서 W 비율대로 분산 투자를 집행하면, 미래 일정 기간 후에 L 이 극대화되어 위험조정수익률이 가장 커진다는 의미다.

MPN을 이용하면 개별 자산들에 대한 미래의 기대수익률과 기대분산을 일일이 추정하지 않고도 포트폴리오 최적화의 목적을 달성할 수 있다. Keras를 이용하면 MPN을 비교적 쉽게 구현할 수 있는데, 실제 구현은 다음 글에 정리하기로 한다.

목차

- 1) 마코비츠-네트워크 (1) : MPN 개요
- 2) 마코비츠-네트워크 (2) : MPN 학습 (teacher forcing)
- 3) 마코비츠-네트워크 (3) : 성능 평가
- 4) 마코비츠-네트워크 (4) : Metric
- 5) 동영상 강의 (유료) : Insight Campus
- 6) 마코비츠-네트워크 (5) : 성능개선 (트랜스포머와 추가학습)

Keras를 이용해서 이전 시간에 살펴본 MPN (Markowitz's Portfolio Network)을 실제로 구현해 보자. 학습 데이터는 S&P500의 상위 50 종목의 일일 수익률을 사용했고, 2002년 5월 24일부터 2021년 8월 23일까지 총 5,022개 데이터를 사용했다. 일일 수익률은 금일 종가 (Pt)와 전일 종가 (Pt-1)를 이용해서 연속 수익률로 측정했다. 종가는 모두 수정 주가 (adjusted price)를 사용했으며, 주말을 제외한 공휴일의 종가는 모두 interpolation 방법으로 채워 넣었다. 이것은 장기 공휴일이 일일 수익률과 변동성 측정에 영향을 미치는 것을 방지하기 위함이다.

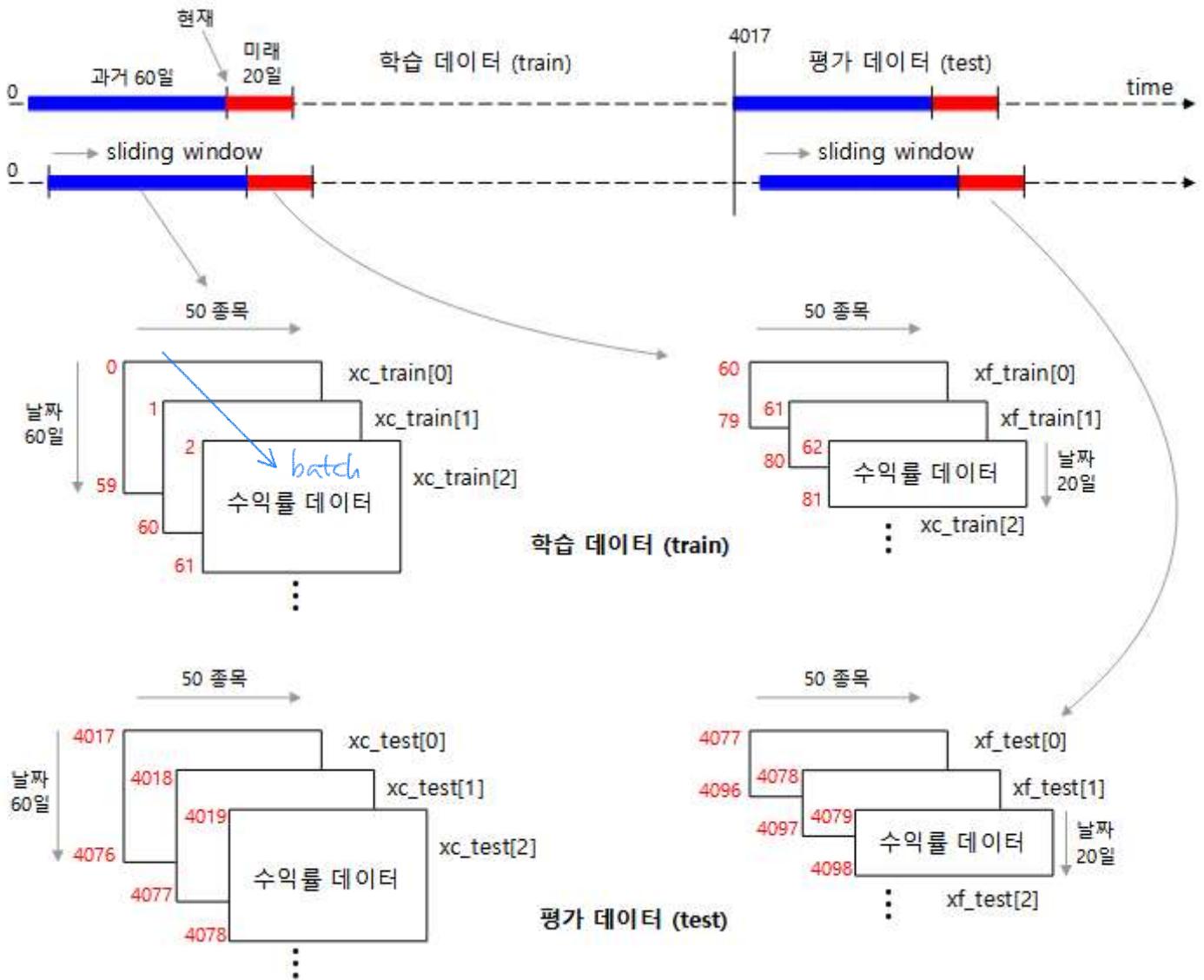
	A	B	C	D	E	F	G	H	I
1	Date	AAPL	MSFT	AMZN	NVDA	JPM	JNU	UNH	HD
2	2002-05-24	-4.177%	-2.887%	0.103%	-1.979%	-1.497%	-0.568%	0.510%	1.205%
3	2002-05-27	-0.353%	-0.886%	-1.058%	-2.004%	-0.797%	-0.155%	0.243%	-2.189%
4	2002-05-28	-0.354%	-0.894%	-1.070%	-2.045%	-0.803%	-0.155%	0.242%	-2.238%
5	2002-05-29	0.000%	-0.517%	-0.790%	3.706%	-0.620%	0.245%	1.130%	-1.431%
6	2002-05-30	0.913%	1.127%	-2.897%	-1.472%	-2.631%	0.016%	1.425%	-1.973%
7	2002-05-31	-3.790%	-3.342%	-0.765%	-2.771%	-0.167%	-0.033%	-0.418%	3.764%
8	2002-06-03	-1.688%	-2.970%	-1.492%	-6.068%	-3.654%	-2.142%	-0.475%	-2.847%
				⋮					
5020	2021-08-18	-2.583%	-0.614%	-1.265%	-2.172%	-0.915%	-1.279%	-1.063%	0.249%
5021	2021-08-19	0.232%	2.056%	-0.422%	3.904%	-0.839%	0.776%	2.501%	0.419%
5022	2021-08-20	1.011%	2.525%	0.382%	5.014%	0.285%	0.486%	0.424%	1.944%
5023	2021-08-23	1.020%	0.095%	2.039%	5.341%	1.272%	-0.433%	-1.017%	-0.457%
				⋮					

[그림-1] S&P500의 50 종목별 주가 수익률 데이터

[그림-1]의 주가 수익률 데이터를 이용해서 MPN을 학습시킬 데이터 세트를 [그림-2]와 같이 구성한다. 총 5,022개 데이터 중 앞부분의 4,017개 (80%)는 학습용으로 사용하고, 뒷부분 1,005개 (20%)는 평가용 (시험용)으로 사용한다. 정확한 시험을 위해서는 평가용 데이터 (evaluation dataset)와 시험용 데이터 (test dataset)를 분리해야 하지만 데이터 양이 적기 때문에 평가용과 시험용을 구분하지 않고 사용했다 (이렇게 해도 크게 문제될 것은 없다).

현재 시점에서 과거 60일 동안 주가 수익률의 흐름을 보고, 미래 20일 동안 위험조정수익률이 극대화될 최적 포트폴리오를 찾는 것을 목표로 한다. MPN에 입력할 학습 데이터는 두 개다. 한 개는 과거 60일 데이터이고, 다른 한 개는 미래 20일 데이터다. 과거 데이터인 xc_train은 (None, 60, 50) 차원을 갖고, 미래 데이터인 xf_train은 (None, 20, 50)이다. None은 batch size이고, 60은 과거의 기간, 20은 미래의 기간, 그리고 50은 종목의 개수다. 변수 이름의 c는 current, f는 future를 의미한다. xc_train과 xf_train은 모두 오른쪽으로 1일씩 sliding하면서 만들어진다.

평가용 데이터도 학습용과 동일하게 만든다. 전체 데이터 중 4,017번째 데이터부터 오른쪽으로 1일씩 sliding하면서 xc_test와 xf_test를 만들어 나간다. 차원은 학습용 데이터와 동일해야 한다.

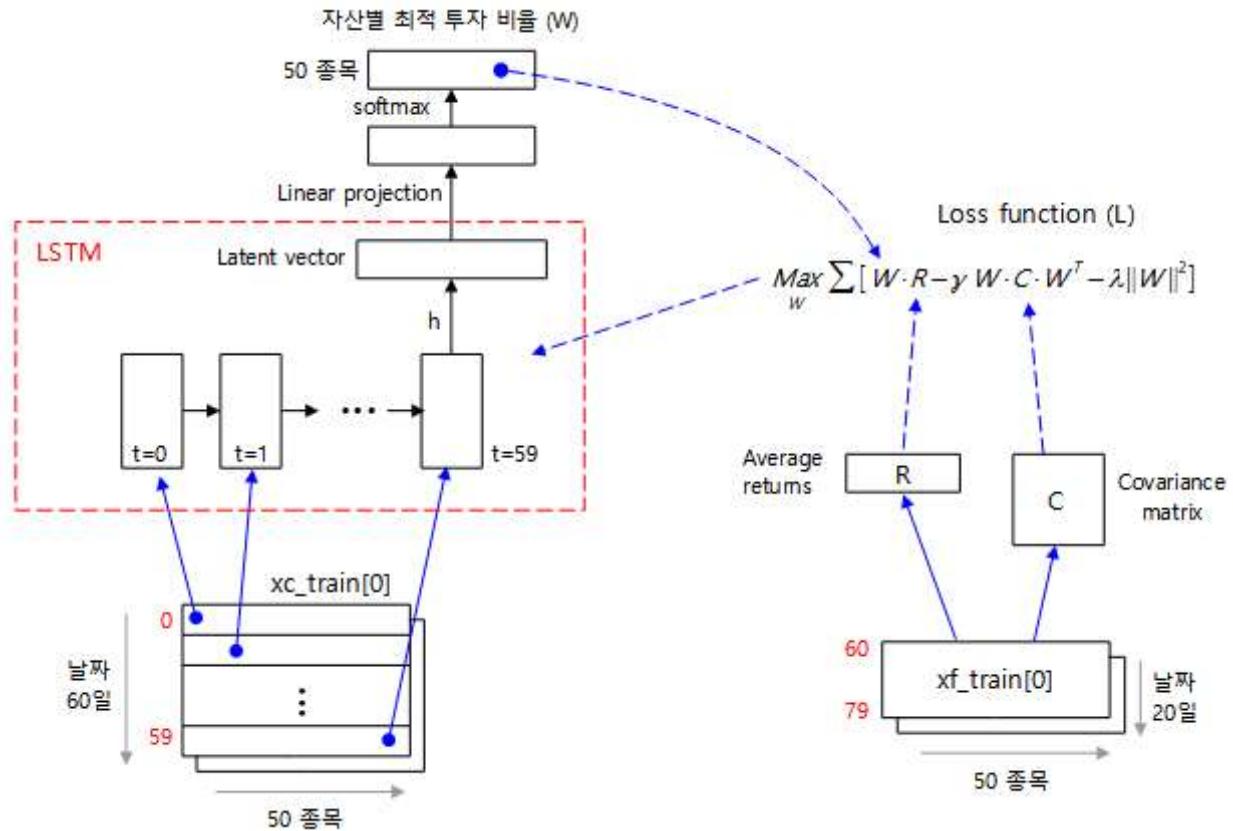


[그림-2] 학습 데이터와 평가 데이터

MPN의 세부 구조는 [그림-3]과 같다. 왼쪽에 LSTM을 구성하고 xc_{train} 을 입력한다. xc_{train} 은 60일의 time sequence를 가지고 있으므로 LSTM도 60번의 recurrent가 발생한다 ($t=0 \sim 59$). LSTM은 2D 구조의 xc_{train} 을 입력 받아서 (batch 차원까지 3D), 1D 구조의 latent vector를 출력한다 (batch 차원까지 2D). LSTM의 출력인 latent vector는 60일 동안 50 종목의 수익률 흐름에 대한 정보를 요약, 함축하고 있다. Latent vector의 크기는 분석자가 (실험을 통해) 적절한 값으로 설정한다 (ex : 64, 128, 등).

LSTM의 출력인 latent vector를 이용해서 종목별 투자 비율 (W)을 결정하기 위해 linear projection 과정을 거친다. Linear projection은 vector의 크기를 변환하기 위한 것이고, 여기서는 종목 개수인 50개로 변환한다. Linear projection은 단순히 선형 feed forward network를 의미하고 Keras에서는 Dense layer를 통과시키는 것을 말한다. Linear projection 결과에 마지막으로 softmax 함수를 적용하면 50 종목에 대한 투자 비율인 W 가 출력된다. Softmax 함수를 이용하면 공매도를 사용하지 않는 경우 마코비츠의 제한 조건을 만족시킬 수 있다.

한편 [그림-3]의 오른쪽은 마코비츠의 목표 함수를 계산하는 부분이다. 이 부분은 네트워크 구조는 아니고, 목표 함수를 계산하는 사용자 정의 함수 (custom loss function)이다. 이 함수에 xf_{train} 을 입력한다. xf_{train} 은 50 종목에 대한 20일 동안의 실현수익률 sequence를 가지고 있다. xf_{train} 을 이용해서 20일 동안의 평균 실현수익률 (R)과 종목 간 공분산 행렬 (C)를 계산하고, 왼쪽 네트워크의 출력인 W 와, 사전에 지정한 γ 와 λ 를 이용해서 목표 함수를 계산한다. 그리고 이 목표 함수가 점차 커지는 방향으로 왼쪽 네트워크의 파라미터들을 반복 학습 시킨다 (gradient ascent).



[그림-3] MPN의 세부 구조

Keras를 이용해서 [그림-3]의 MPN을 구현해 본다. Keras는 기본적으로 지도학습 (supervised learning)에 적합한 구조로 만들어 졌다. 그러나 MPN은 비지도학습 (unsupervised learning)의 형태이기 때문에 Keras에서 약간의 트릭을 써야 한다.

우선 학습에 필요한 사항들을 아래와 같이 설정해 주었다. 일일 수익률은 수치가 너무 작아 스케일을 월간 수익률 정도로 조절해 주었고, 필요한 상수 값들을 적당히 설정해 주었다. 학습 데이터인 xc_train 과 xf_train 은 입력 순서에 따른 상관성을 제거하기 위해 sklearn.utils에 있는 shuffling 함수를 이용해서 적당히 섞어 주었다. 그리고 성능을 평가할 프로그램에서 이 모델을 사용하기 위해서는 학습이 완료된 모델을 저장해 두어야 한다.

```
# 월간 수익률 정도의 스케일로 변환한다
xc_train = xc_train.astype('float32') * 20.0
xf_train = xf_train.astype('float32') * 20.0
xc_test = xc_test.astype('float32') * 20.0
xf_test = xf_test.astype('float32') * 20.0

N_TIME = xc_train.shape[1]
N_FUTURE = xf_train.shape[1]
N_STOCKS = xf_train.shape[2]

# 학습 데이터는 shuffling 한다.
xc_train, xf_train = shuffle(xc_train, xf_train)

# over confidence를 제어할 조절 변수 정의
GAMMA_CONST = 0.1
REG_CONST = 0.1
SAVE_MODEL = 'data/2-1.Markowitz_network.h5'
```

[그림-3]의 오른쪽 부분인 목표 함수는 아래와 같이 만든다. 이것은 Keras에서 제공하는 custom loss function의 형태다. Keras는 model.fit()에 지정된 target data를 loss function의 첫 번째 인자로 전달하고, 네트워크의 출력인 W 를 이 함수의 두 번째 인자로 전달한다. 즉, 첫 번째 인자인 y_true 는 model.fit()에서 지정한 xf_train 이고, 두 번째 인자인 y_pred 는 W 이다. 이 값을 이용해서 R 과 C 를 계산한 후 목표 함수를 계산한다. 우리는 이 목표 함수를 최대화 시켜야 한다. 그러나 Keras의 optimize는 목표 함수를 최소화 시키도록 만들어졌기 때문에 목표 함수의 부호를 바꿔 주어야 한다. 그러면 gradient ascent 문제를 gradient descent 문제로 바꿀 수 있다. 그럼 이 함수를 MPN의 loss function으로 사용할 수 있다.

```

# 최적 포트폴리오를 구축할 목표 함수를 정의한다.
# MPN에서는 이 함수를 loss로 이용한다. max(objective) = min(-objective)
# y_true = model.fit()에서 전달된 N_FUTURE일 후의 수익률 (xf_train)이 들어온다.
# y_pred = 마코비츠 네트워크의 출력이 전달된다. (keras 내부 기능)
def markowitz_objective(y_true, y_pred):
    W = y_pred      # 마코비츠 네트워크의 출력
    xf rtn = y_true
    W = tf.expand_dims(W, axis = 1)
    R = tf.expand_dims(tf.reduce_mean(xf rtn, axis = 1), axis = 2)
    C = tfp.stats.covariance(xf rtn, sample_axis=1)

    rtn = tf.matmul(W, R)
    vol = tf.matmul(W, tf.matmul(C, tf.transpose(W, perm = [0, 2, 1]))) * GAMMA_CONST
    reg = tf.reduce_sum(tf.square(W), axis = -1) * REG_CONST
    objective = rtn - vol - reg

    return -tf.reduce_mean(objective, axis=0)

```

[그림-3]의 왼쪽 부분은 아래와 같이 만든다. xc_train을 입력 받아 LSTM으로 전달하고, LSTM의 출력을 linear projection인 Dense로 전달한다. Dense의 activation function으로는 tanh를 사용했다. 이것은 아래 주석과 같이 over confidence를 방지하기 위함이다 (optional). 그리고 마지막으로 softmax 함수를 취해서 결과를 출력한다. 이 모델의 loss function에는 위에서 만든 markowitz_objective 함수를 사용한다.

```

# LSTM으로 Markowitz 모델을 생성한다.
xc_input = Input(batch_shape = (None, N_TIME, N_STOCKS))
h_lstm = LSTM(64, dropout = 0.5)(xc_input)
y_output = Dense(N_STOCKS, activation='tanh')(h_lstm) # linear projection

# 특정 종목을 과도하게 매수하는 것을 방지하기 위해 위에서 tanh를 사용했다.
# (over confidence 방지용). REG_CONST를 적용했기 때문에 이미 고려된 사항이지만,
# 안전을 위해 추가했다. ex : [-3, 0.4, 0.2, +20] → [-0.995, 0.380, 0.197, 1.0]

# 마코비츠의 최적 weights
y_output = Activation('softmax')(y_output)

model = Model(xc_input, y_output)
model.compile(loss = markowitz_objective, optimizer = Adam(learning_rate = 1e-5))

```

이제 모델이 완성되었으므로 아래와 같이 이 모델을 학습 시킨다. fit()함수의 첫 번째 인자인 xc_train은 MPN의 왼쪽 네트워크의 입력으로 들어가고, 두 번째 인자인 xf_train은 markowitz_objective() 함수의 첫 번째 인자로 전달된다. 반복 횟수와 mini-batch를 위한 batch_size는 적당히 설정해 주었고, validation_data는 나중에 평가 목적으로 만들어 둔 xc_test와 xf_test를 넣어 주었다. 정확한 시험을 위해서는 validation_data를 따로 준비해서 (ex : xc_eval, xf_eval) 적절한 반복 횟수 등을 결정해야 하지만, 전체 데이터가 적어서 그냥 xc_test와 xf_test를 사용했다.

```

# MPN을 학습하고 결과를 저장한다.
hist = model.fit(xc_train, xf_train, epochs=150, batch_size = 100, validation_data = (xc_test, xf_test))
model.save(SAVE_MODEL)

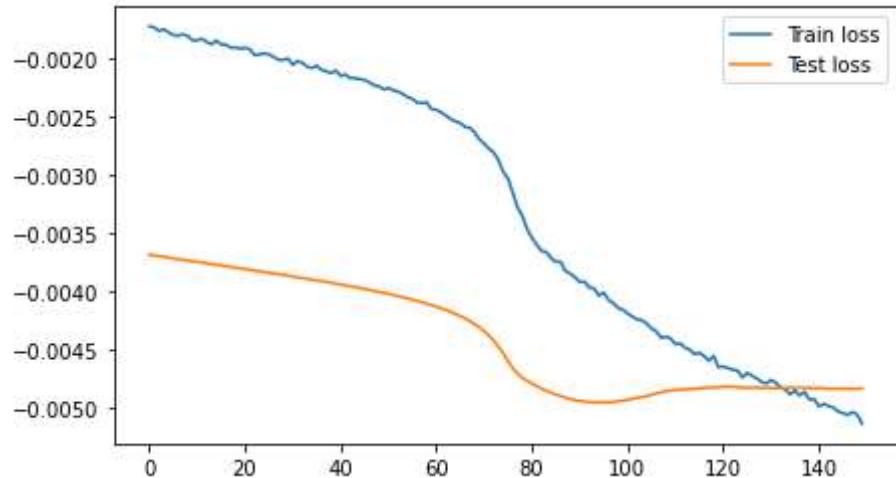
```

학습 과정에서 markowitz_objective()의 변화는 [그림-4]와 같다. x축은 반복 횟수인 epochs이고, y 축은 markowitz_objective 값이다. xc_train과 xf_train으로 측정한 train loss는 지속적으로 감소하고 있고 (파란색 커브), xc_test와 xf_test로 측정한 test loss는 점차 감소하다가 정체되고 있다 (오렌지색 커브).

```

# loss trajectory를 확인한다.
plt.figure(figsize=(6, 4))
plt.plot(hist.history['loss'], label='Train loss')
plt.plot(hist.history['val_loss'], label='Test loss')
plt.legend()
plt.show()

```

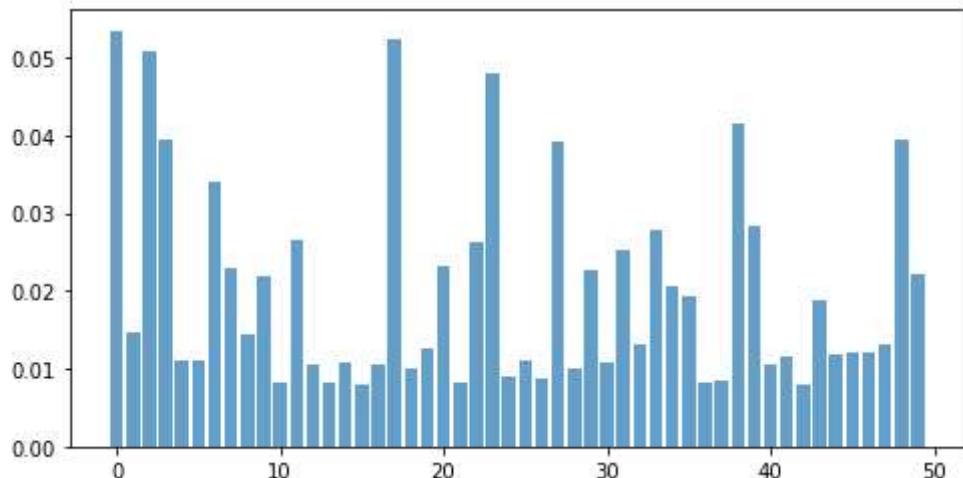


[그림-4] epochs에 따른 markowitz_objective() 값의 변화

마지막으로 test 데이터로 추정한 50 종목에 대한 최적 투자 비율인 W 의 분포를 확인해 본다. [그림-5]는 $xc_test[0]$ 데이터를 MPN의 LSTM에 입력하고 출력을 관찰한 것이다. 이 기간의 W 는 약 1%부터 5%까지의 분포를 보인다. 이것은 $xc_test[0]$ 의 마지막 시점에서 어떤 종목은 총 자본의 1% 정도 투자하고, 어떤 종목은 5% 정도 투자했을 때 향후 20일 후의 위험조정수익률이 극대화 된다는 것을 의미한다.

```
# 최적 포트폴리오 결과 조회용 코드
def check_w(n = 0):
    plt.figure(figsize=(8, 4))
    y_pred = model.predict(xc_test[n].reshape(1, N_TIME, N_STOCKS))[0]
    plt.bar(np.arange(N_STOCKS), y_pred, alpha = 0.7)
    plt.show()

check_w(0)
```



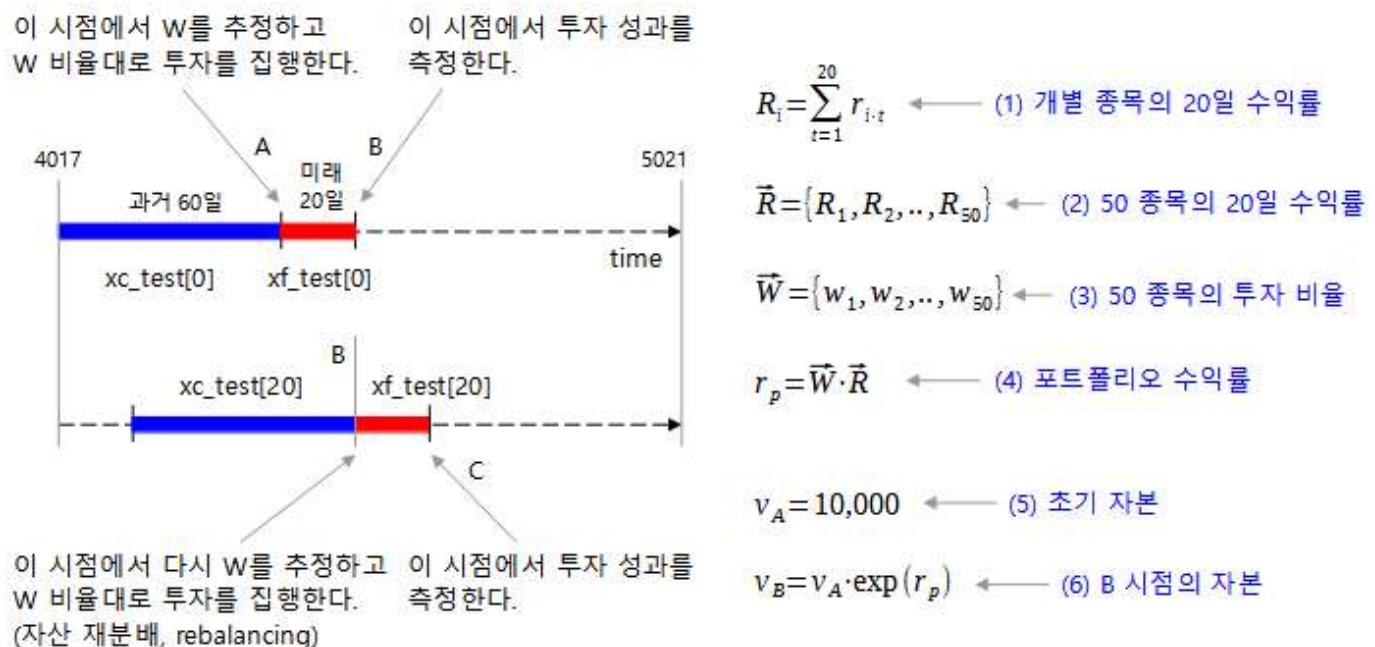
[그림-5] $xc_test[0]$ 로 추정한 $xf_test[0]$ 의 최적 투자 비율 (W) 조회 결과

그럼 평가 기간 동안 MPN이 출력하는 W 비율대로 투자한다면 좋은 성과를 얻을 수 있을까? 이 부분에 대한 검증은 다음 글에서 다루기로 한다.

목차

- 1) 마코비츠-네트워크 (1) : MPN 개요
- 2) 마코비츠-네트워크 (2) : MPN 학습 (teacher forcing)
- 3) 마코비츠-네트워크 (3) : 성능 평가
- 4) 마코비츠-네트워크 (4) : Metric
- 5) 동영상 강의 (유료) : [Insight Campus](#)
- 6) 마코비츠-네트워크 (5) : 성능개선 (트랜스포머와 추가학습)

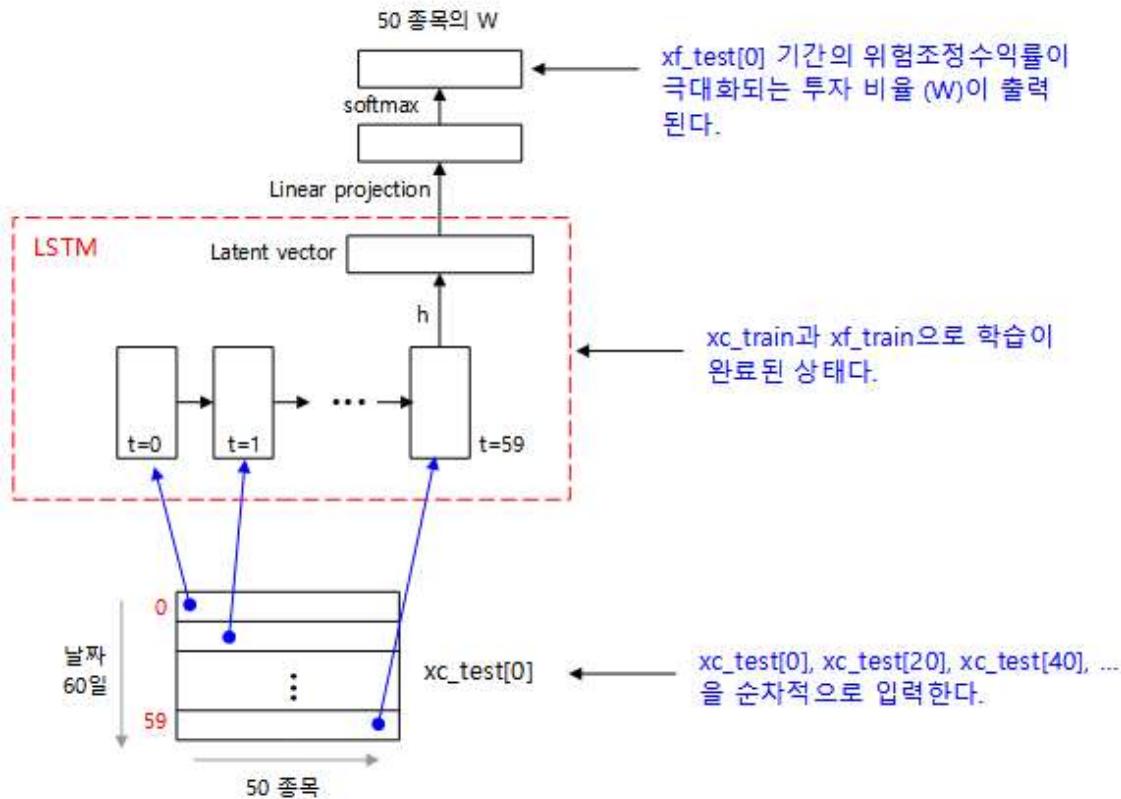
이전 시간에 학습시킨 MPN (Markowitz's Portfolio Network)의 출력대로 투자를 수행했을 때 어느 정도의 성과를 거둘 수 있는지 확인해 보자. MPN의 성과와 benchmark로 사용할 CRP (constant rebalanced portfolio)와의 성과를 비교해 보고, S&P500 index (시장 수익률)의 성과와도 비교해 본다. CRP란 종목 별 비중이 동일하게 유지되도록 (50 종목에 대해 2%씩) 포트폴리오를 재분배하는 것을 말한다.



[그림-1] MPN의 성능 평가 방법

성능 평가는 시험 데이터인 xc_test 와 xf_test 를 이용한다. xc_test 와 xf_test 는 학습에 사용되지 않은 데이터다. 사실, $fit()$ 함수에서 loss 측정용으로는 사용했기 때문에 간접적으로는 사용했다고 봐야 한다. 그러나 MPN의 파라미터 θ 를 업데이트하는 데는 전혀 사용되지 않았다. 이것은 데이터 양이 작아 evaluation dataset을 별도로 두지 않았기 때문이며, MPN의 성능을 확인하는 데 크게 문제가 되지는 않는다. 성능을 평가하는 방법은 [그림-1]과 같다. S&P500의 50 종목 데이터 중 4,017 번째부터 5,021 번째 까지가 시험용 데이터다 (이전 글 참조).

학습이 완료된 MPN으로 종목별 최적 투자 비율을 추정한다. MPN을 학습시킬 때는 loss function이 필요했지만, 투자 비율을 추정할 때는 loss function이 필요 없다. [그림-2]와 같이 MPN의 왼쪽에 있는 LSTM 네트워크만 이용하면 된다. LSTM에 원하는 기간의 수익률 데이터를 입력하면 다음 20일 동안 위험조정수익률이 극대화될 종목별 최적 투자비율 (W)이 출력된다.



[그림-2] 종목별 최적 투자 비율 추정

성능 평가의 세부 절차는 다음과 같다.

1. $xc_{test}[0]$ 의 마지막 날인 A 시점에 $xc_{test}[0]$ 데이터를 학습이 완료된 MPN에 입력한다. W 가 출력된다.
2. A 시점에 W 비율대로 50 종목에 분산 투자하고 (포트폴리오의 초기 자본 10,000원), $xf_{test}[0]$ 기간인 20일을 기다린다 (buy & hold).
3. $xf_{test}[0]$ 의 마지막인 B 시점에 포트폴리오 전체 value를 측정한다. 포트폴리오 value 측정 방법은 [그림-1]의 식 (1) ~ (6)과 같다.
4. B 시점에 다시 $xc_{test}[20]$ 데이터를 MPN에 입력하여 새로운 W 를 구한다.
5. B 시점에서 새로운 W 에 맞게 포트폴리오의 비중을 조절한다 (자산재분배 혹은 rebalancing)
6. $xf_{test}[20]$ 기간인 20일을 기다린 후 C 시점에 포트폴리오 전체 value를 측정한다.
7. 이 과정을 test 기간 끝까지 반복한다. 이렇게 하면 약 한 달에 (20 거래일) 한 번씩 자산재분배를 수행하는 것이다.
8. 포트폴리오의 최종 value를 측정하고, benchmark인 CRP value와 비교한다.

아래 코드는 위의 절차를 구현한 것이다.

```

N_TIME = xc_test.shape[1]
N_FUTURE = xf_test.shape[1]
N_STOCKS = xf_test.shape[2]

# 저장된 Markowitz 모델을 가져온다.
SAVE_MODEL = 'data/2-1.Markowitz_network.h5'
model = load_model(SAVE_MODEL, compile = False)
model.summary()

# 백 테스트를 수행한다.
prt_value = [10000] # portfolio의 초기 value
crp_value = [10000] # CRP의 초기 value
w_crp = np.ones(N_STOCKS) / N_STOCKS # CRP 비율 (균등 비율)

w_history = []
for i in range(0, xc_test.shape[0], N_FUTURE):
    # 이 시점에 각 종목을 w_port 비율대로 매수한다.
    # 학습할 때 월간 수익률로 변환했으므로, 여기서도 변환해야 한다.
    x = xc_test[i][np.newaxis,:,:] * 20.0
    w_port = model.predict(x)[0]
    w_history.append(w_port)

    # 다음 기의 누적 수익률
    m_rtn = np.sum(xf_test[i], axis = 0)

    # 누적 수익률과 w_port (W)로 포트폴리오의 수익률을 계산한다.

```

```

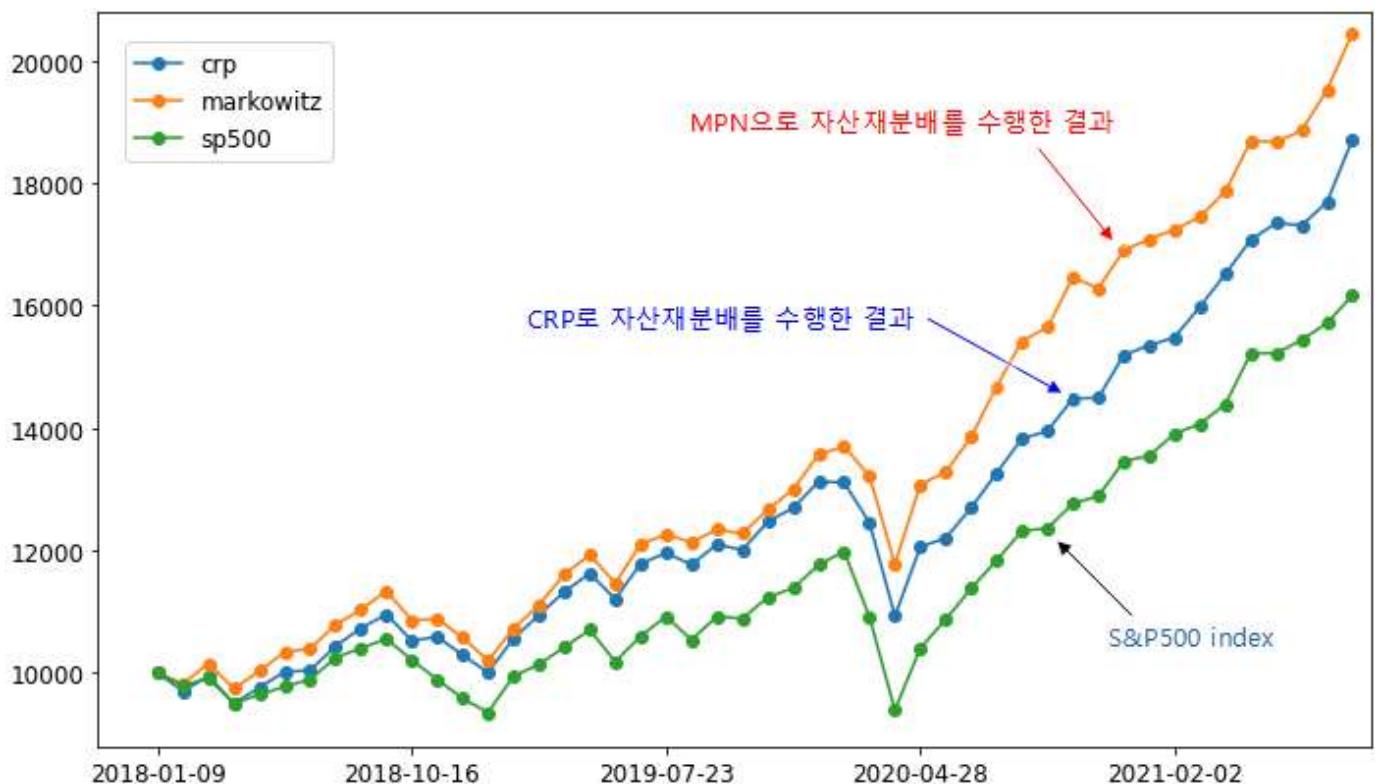
prt_value.append(prt_value[-1] * np.exp(np.dot(w_prt, m_rtn)))
crp_value.append(crp_value[-1] * np.exp(np.dot(w_crp, m_rtn)))

# 평가 시점의 날짜를 발췌한다.
idx = np.arange(0, len(test_date), N_FUTURE)

# Markowitz 성과와 CRP 성과를 데이터 프레임에 기록해 둔다.
perf_df = pd.DataFrame({'crp':crp_value, 'markowitz':prt_value},
index=test_date[idx])

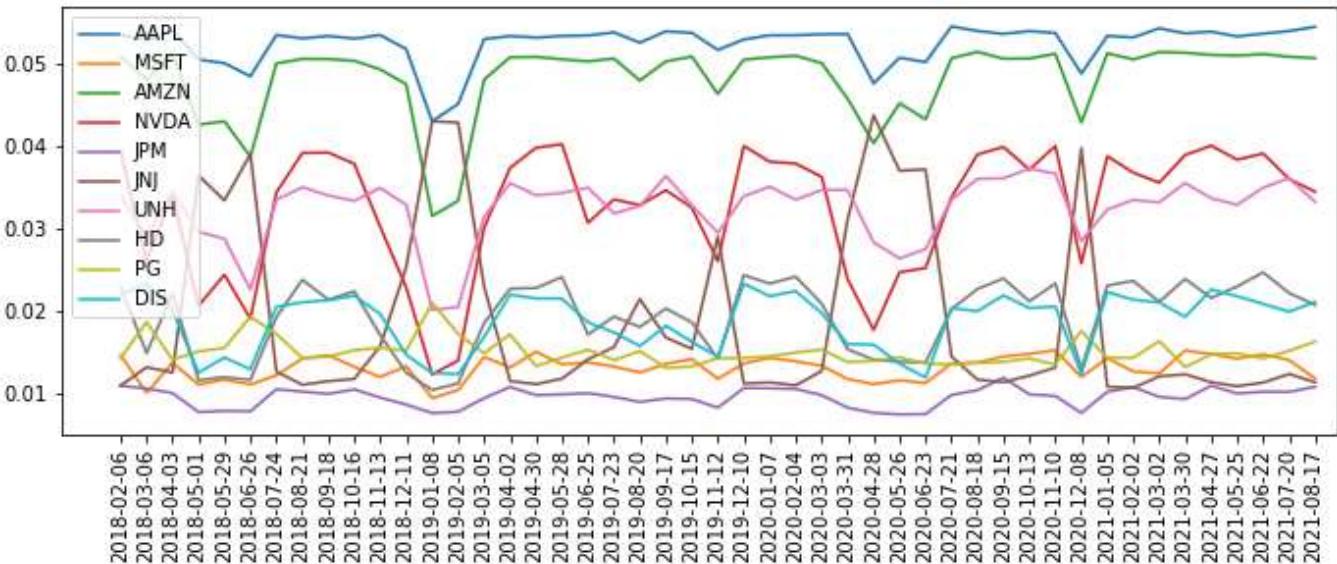
```

성능 평가 결과는 [그림-3]과 같다. 2018년 1월 9일부터 2021년 8월 23일까지 20 거래일 (약 1개월) 마다 자산재분배를 수행한 결과다. 오렌지색 커브는 위의 절차대로 MPN을 이용해서 얻은 성과이고, 파란색 커브는 각 종목에 균등하게 2% 씩 CRP를 적용한 결과다. 그리고 녹색 커브는 S&P500 지수이다. MPN의 성과가 가장 좋았고, MPN이나 CRP의 성과가 S&P500 지수보다 높았다. S&P500 지수는 시장의 기준으로 볼 수 있으며, MPN이나 CRP가 시장보다 높은 이유는 S&P500의 상위 50 종목의 실적이 다른 종목들 보다 높았기 때문일 것이다. 그리고 이 결과는 자산재분배에 따른 거래비용은 고려하지 않은 것이다.



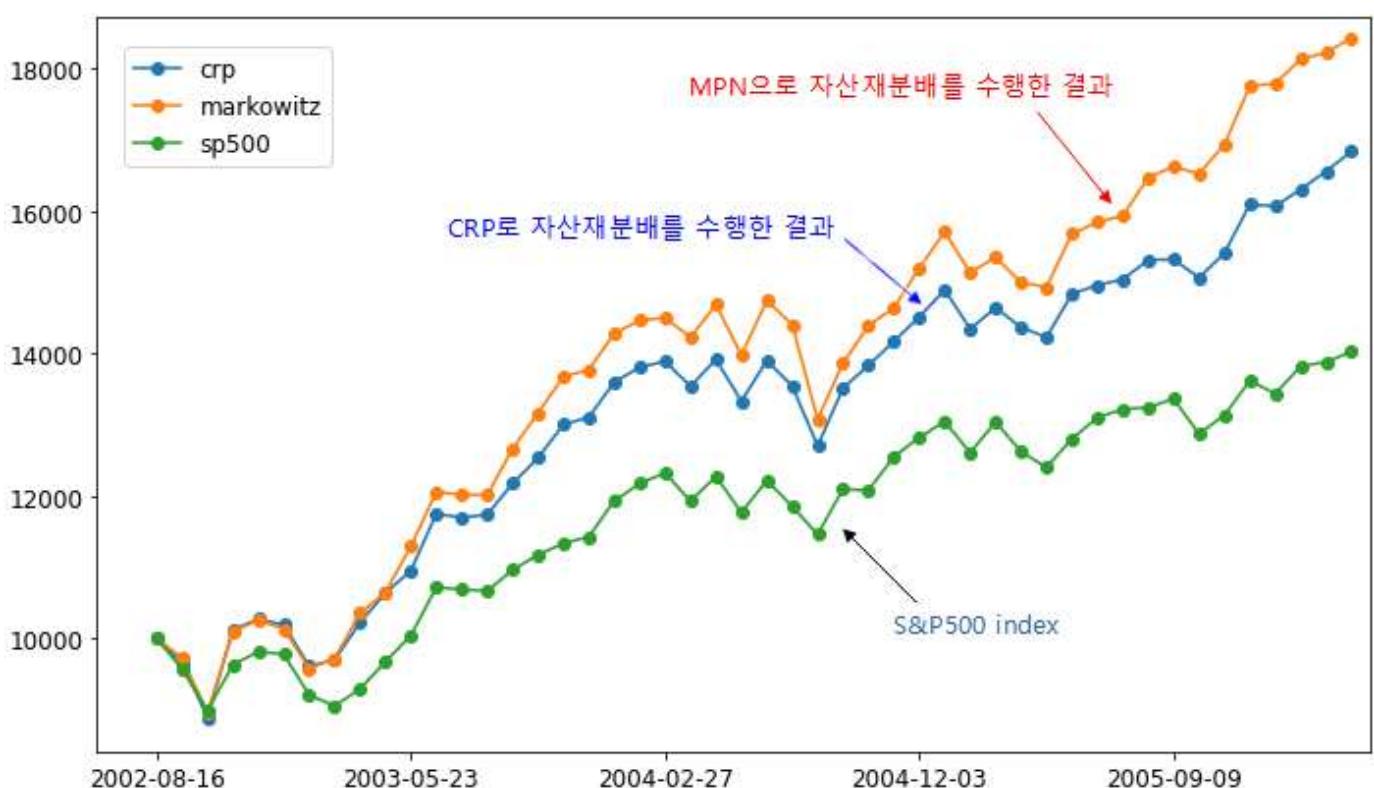
[그림-3] 성능 평가 결과

[그림-4]는 상위 10 종목의 투자 비율인 W 의 변화를 관찰한 것이다. 애플 (AAPL)과 아마존 (AMZN)의 비중이 가장 높았다. 애플의 경우 2019년 1월 8일에는 약 4%로 비중이 약간 축소되긴 했지만, 거의 5% 비중을 유지하고 있다. 엔비디아 (NVDA)의 경우는 비중이 약 1% ~ 4%로 변화가 많았다. 아래 부분에 비중이 1%로 유지되는 종목들도 많았다.



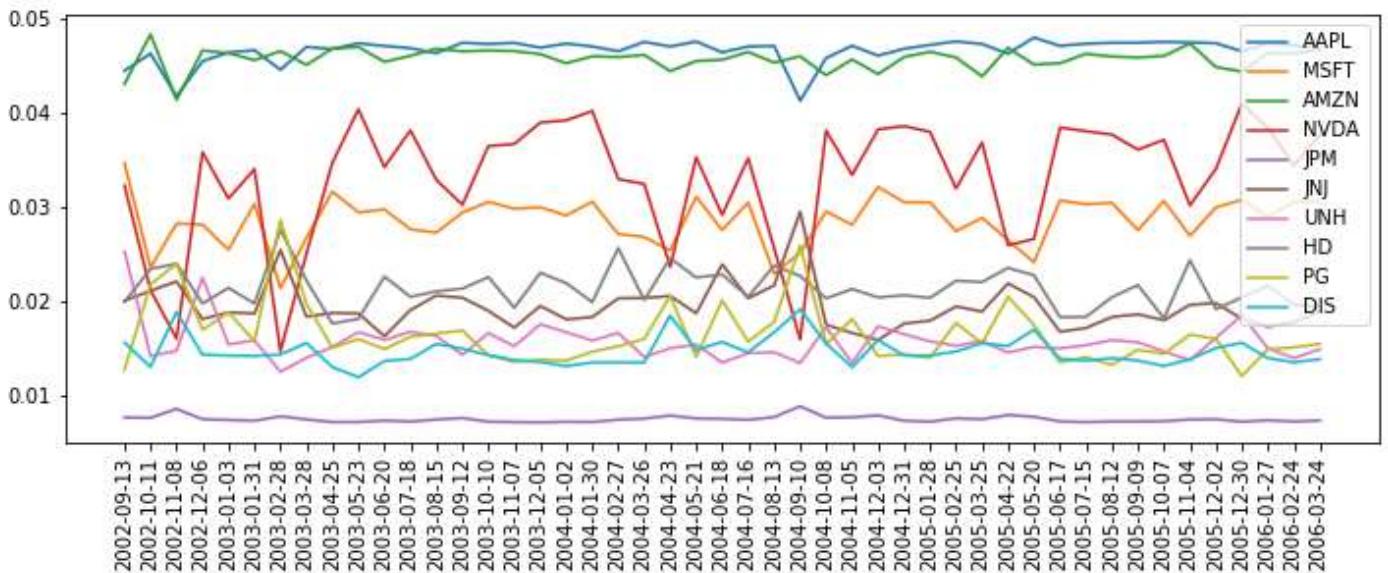
[그림-4] 투자 비율 (W)의 변화

정확한 평가를 위해서는 학습 기간과 평가 기간을 다르게 적용해 봐야 한다 (cross validation). [그림-5]는 S&P500의 앞 부분 1,005개를 평가 기간으로 설정하고, 뒷부분 4,017개를 학습 기간으로 설정해서 MPN을 다시 학습시키고, 평가한 결과다. 즉, 뒷부분의 4,017개 데이터를 학습해서, 앞 부분 1,005개를 평가했다. 이 기간에도 MPN의 성과가 가장 좋았다.



[그림-5] 다른 기간의 성능 평가 결과

[그림-6]과 같이 이 기간에도 애플 (AAPL)과 아마존 (AMZN)의 비중이 가장 높았다.



[그림-6] 다른 기간의 투자 비율 (W)의 변화

과거의 주가 수익률을 학습해서 미래의 최적 투자 비율을 추정해 봤고, 그 성과가 유의미 하다는 것도 살펴 봤다. 주가 수익률이 완전한 랜덤워크를 따른다면 과거 수익률과 미래 수익률은 서로 독립적이고, 아무런 상관이 없기 때문에 과거로부터 미래의 어떤 것도 추정할 수가 없다. 그러나 이 결과는 미래의 W가 과거 수익률에 (어느 정도는) 종속적이라는 것을 말해주고 있다.

여기까지 마코비츠의 포트폴리오 이론과 딥러닝 기술을 결합해서 최적 포트폴리오를 구축해 봤다. 미래의 기대 수익률과 분산을 알 수 없기 때문에, 마코비츠의 이론만으로는 이와 같은 전략을 만들 수 없다. 그러나 딥러닝 기술을 결합하면 이런 전략이 가능하고, 더 나아가 훨씬 더 많은 것들을 할 수 있다는 것을 예상해 볼 수 있다. 금융 이론과 딥러닝 기술을 결합하는 분야의 연구가 필요한 이유다.

목차

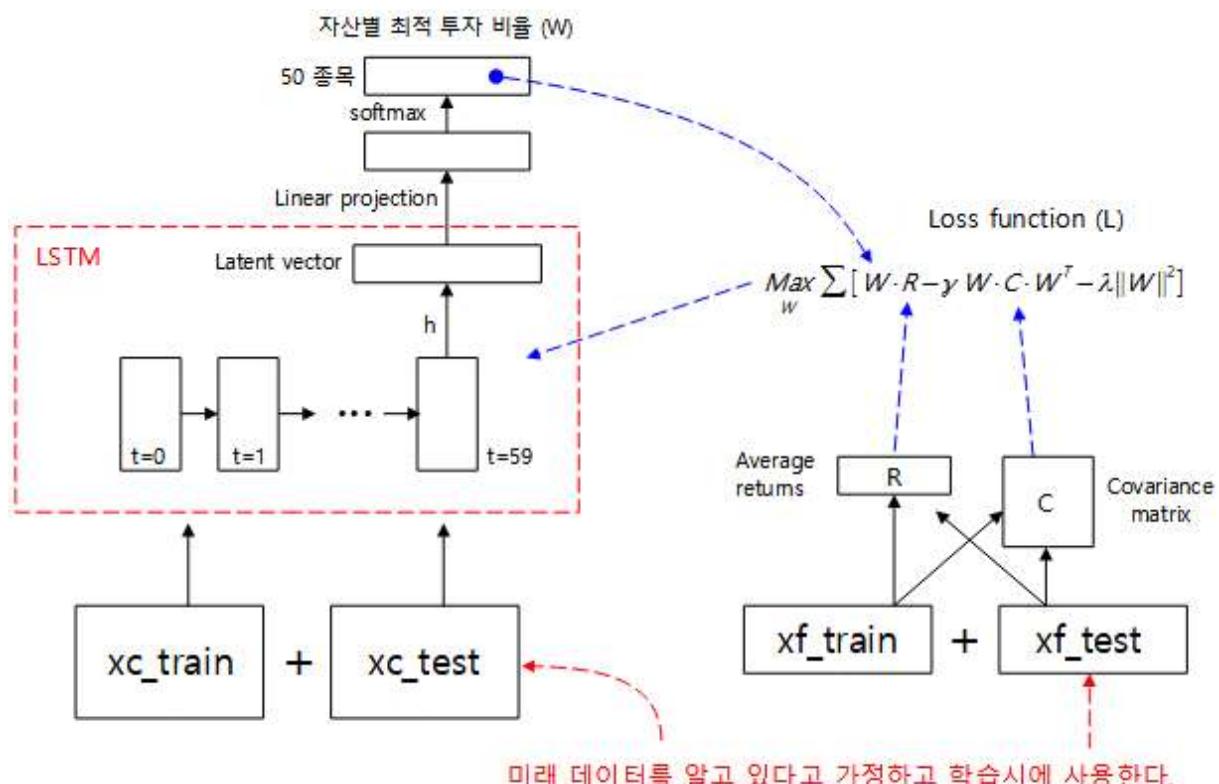
- 1) 마코비츠-네트워크 (1) : MPN 개요
- 2) 마코비츠-네트워크 (2) : MPN 학습 (teacher forcing)
- 3) 마코비츠-네트워크 (3) : 성능 평가
- 4) 마코비츠-네트워크 (4) : Metric
- 5) 동영상 강의 (유료) : [Insight Campus](#)
- 6) 마코비츠-네트워크 (5) : 성능개선 (트랜스포머와 추가학습)

MPN의 성능 척도 (metric)를 만들 수 없을까? 지도학습의 경우는 정답이 존재하기 때문에 정답과 추정치의 차이를 측정할 수 있다. Classification 문제에 대해서는 정확도 (accuracy), regression 문제에 대해서는 R-square 등으로 성능의 척도를 측정할 수 있다. 그러나 비지도학습의 경우는 정답 자체가 존재하지 않기 때문에 추정치와의 차이를 측정할 방법이 없다. MPN은 비지도학습이다. 따라서 MPN의 성능을 측정할 척도는 없다.

MPN에서 정답의 개념은 무엇인가? MPN에서 정답이란 미래의 위험조정수익률이 가장 커지는 투자 비율 (W)이다. 이것은 미래 시점에서 사후적으로만 측정이 가능하다. 현재 시점에서 정답을 미리 알 수는 없다. 사실은 미래에도 정답은 모른다. 미래에는 불확실성이 사라져서 포트폴리오와 W 라는 개념이 없어지기 때문이다.

여기서 한가지 가정을 해보자. 미래를 알 수 있는 존재가 있다고 하자. 예를 들어 영화 “컨택트” (원 제목 : Arrival)에 등장하는 헵타포드 (Heptapod)가 MPN으로 포트폴리오를 구축한다고 하자. 물론, 헵타포드가 미래를 정확히 안다면 포트폴리오를 구축할 필요도 없다. 그냥, 미래에 가장 수익률이 높을 종목에 집중 투자하면 된다. 그러나 헵타포드가 정확한 미래는 모르고 확률적으로 불확실성이 존재하는 상태로 미래를 추정한다면, 헵타포드도 불확실성을 줄이기 위해 포트폴리오를 구축할 것이다.

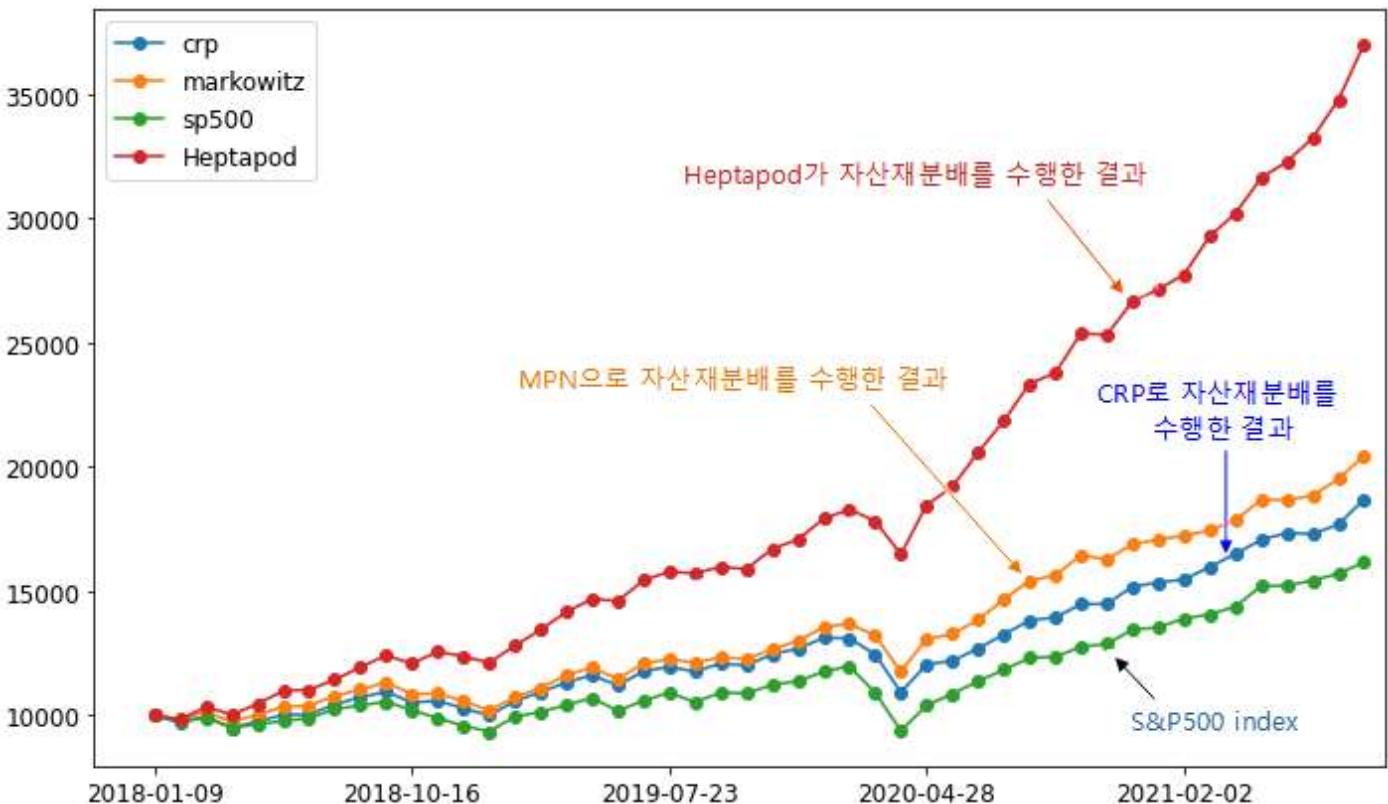
헵타포드는 [그림-1]처럼 미래 데이터인 xc_{test} 와 xf_{test} 를 모두 사용해서 MPN을 학습시킬 수 있다. 이것을 Heptapod's Portfolio Network (HPN)이라 하자. 물론, 우리는 미래를 모르기 때문에 이렇게 할 수는 없다. [그림-1]과 같이 학습할 때 xc_{test} 와 xf_{test} 를 사용하면 미래 상황이 학습에 반영된다. 그리고 평가할 때 LSTM에 xc_{test} 를 입력하면 미래의 최적 투자 비율 (W)이 출력된다. 이 값을 정답으로 간주하고 MPN의 출력과 비교하면 성능 척도를 측정해 볼 수 있다. 즉 HPN으로 정답을 출력하고, MPN의 추정치와 비교해보면 MPN의 출력이 정답에 어느 정도 근접한지 측정해 볼 수 있다.



[그림-1] Heptapod's Markowitz Network (HPN)

헵타포드가 HPN을 이용해서 포트폴리오를 구축할 때의 성과와 우리가 사용할 MPN의 성과를 비교해보면 [그림-2]와 같다. HPN의 성능이 훨씬 우수하다. 학습할 때 미래의 xc_{test} 와 xf_{test} 를 미리 반영했기 때문에 이는 당연한 결과다. 우리는 HPN을 마냥 부러워만하는 게 아니라 MPN의 결과와 정답이라고 간주한 HPN의 차이가 얼마나 큰가에 관심이 있다. [그림-2]의 결과는 HPN과 MPN의 차이가 상당히 크다는 것을 보여주고 있다. MPN의 성과가 HPN에 비해 낫다.

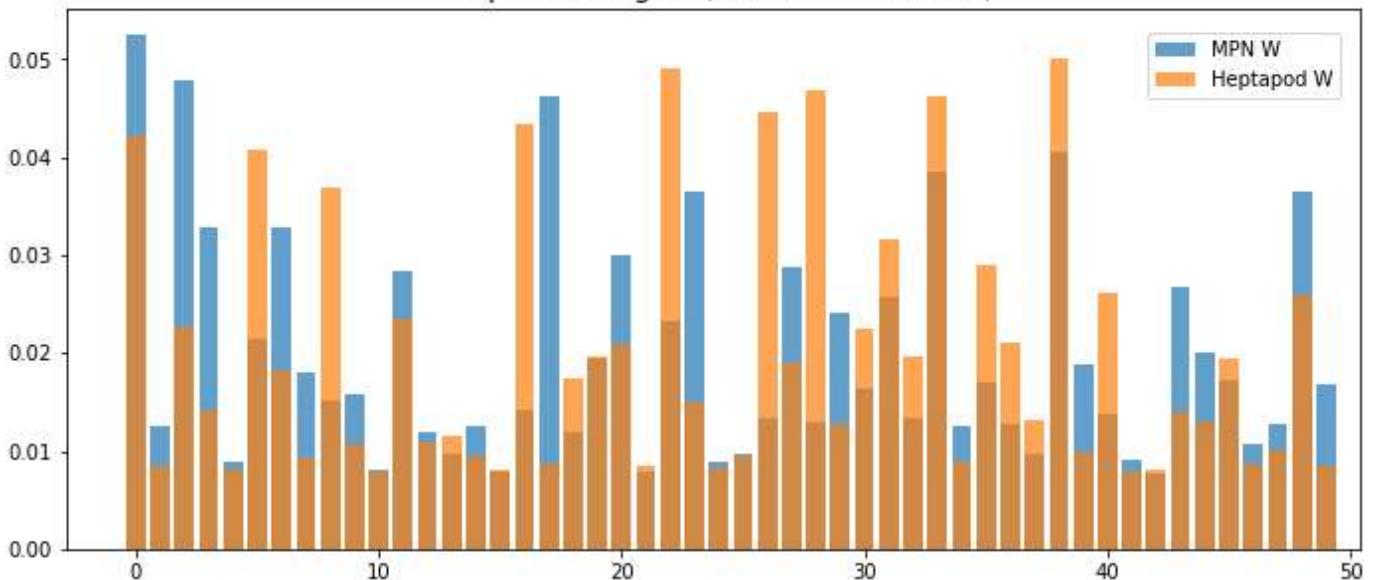
이것은 순수 과거 데이터만을 사용해서 미래를 추정하는 것이 그만큼 어렵다는 것을 말해 주는 것이기도 하다. 어쨌든 이제 MPN의 성과와 비교할 대상이 생겼다.



[그림-2] HPN의 성과

HPN을 사용하는 데 한가지 제약이 더 있다. HPN은 평가용 데이터까지 모두 사용하기 때문에 모든 데이터를 완전히 암기할 때까지 loss를 줄인다. 즉, 학습의 반복 횟수인 epochs를 늘릴수록 [그림-2]의 HPN 성과는 계속 더 올라간다. 그럼 어디까지 정답으로 인정해야 하는지에 대한 문제가 발생한다. 이 문제를 해결하려면 epochs의 상한선을 한정하거나, xc_test와 xf_test 데이터의 사용량 (예를 들어 랜덤 50%만 사용)을 한정하는 기준이 필요하다. 분석이 나름대로 이 기준을 정해서 HPN을 사용하면 MPN의 성능 척도 (metric)를 측정할 수 있다. [그림-2]는 HPN을 학습시킬 때 MPN의 조건과 동일한 상태에서 epochs를 500번으로 한정한 결과다.

Optimal weights (correlation = 0.386)



[그림-3] 특정 기간에 HPN의 W와 MPN의 W를 비교한 결과

[그림-3]은 특정 기간에 (ex : xc_test[10]) HPN으로 추정한 투자 비율인 Wh와 MPN으로 추정한 Wm을 비교한 결과다. Wh는 정답으로 간주한 최적 투자 비율이고, Wm은 과거 데이터로 추정한 미래의 최적 투자 비율이다. 두 결과에 대해 유사도나 상관계수 혹은 R-square 같은 측정치를 이용하면 MPN에 대한 metric을 측정할 수 있다. [그림-3] 기간에는 두 결과의 상관계수가 0.386으로 측정됐다. 상관계수가 높은 편은 아니지만 미래를 추정하는 게 어렵다는 것을 감안하면 그런대로 높은 수치라 할 수 있다. xc_test 전체 기간에 대해 상관계수를 측정하고 평균을 계산했더니 0.384가

나왔다. 이 수치를 MPN의 metric으로 사용할 수 있다. 그럼 모델의 형태를 변경하거나, 다른 모델을 사용할 경우 (CNN, Transformer 등)에 대해 이 metric을 이용해서 모델별 성능을 평가할 수도 있다.

MPN의 metric으로 사용한 평균 상관계수가 0.384라는 것에 또 다른 의미를 부여해 보자. 이는 과거 데이터로 미래를 추정한 결과이므로, 과거와 미래가 0.384 정도의 상관성을 갖는다고 볼 수 있다. 즉, 이 정도의 종속성이 존재한다고 볼 수 있다 (이론적 엄밀성은 없지만 ...). 개별 주가 시계열이나 수익률 시계열에 대해서는 과거를 이용해서 미래를 추정하는 것은 가능해 보이지 않는다. 즉 개별 종목의 시계열은 과거와 미래가 서로 독립적이다. 그러나 여러 종목의 수익률 데이터를 종합하면 (약간의) 종속성이 발생한다고 볼 수도 있다 (한 종류의 실험으로 단언할 수는 없지만...).

여기까지 MPN에 대한 metric을 측정할 방법에 대해 알아봤다. 다음 시간에는 LSTM이 아닌 다른 모델을 이용해서 MPN을 구성해 보기로 하겠다.

목차

- 1) 마코비츠–네트워크 (1) : MPN 개요
- 2) 마코비츠–네트워크 (2) : MPN 학습 (teacher forcing)
- 3) 마코비츠–네트워크 (3) : 성능 평가
- 4) 마코비츠–네트워크 (4) : Metric
- 5) 동영상 강의 (유료) : [Insight Campus](#)
- 6) 마코비츠–네트워크 (5) : 성능개선 (트랜스포머와 추가학습)

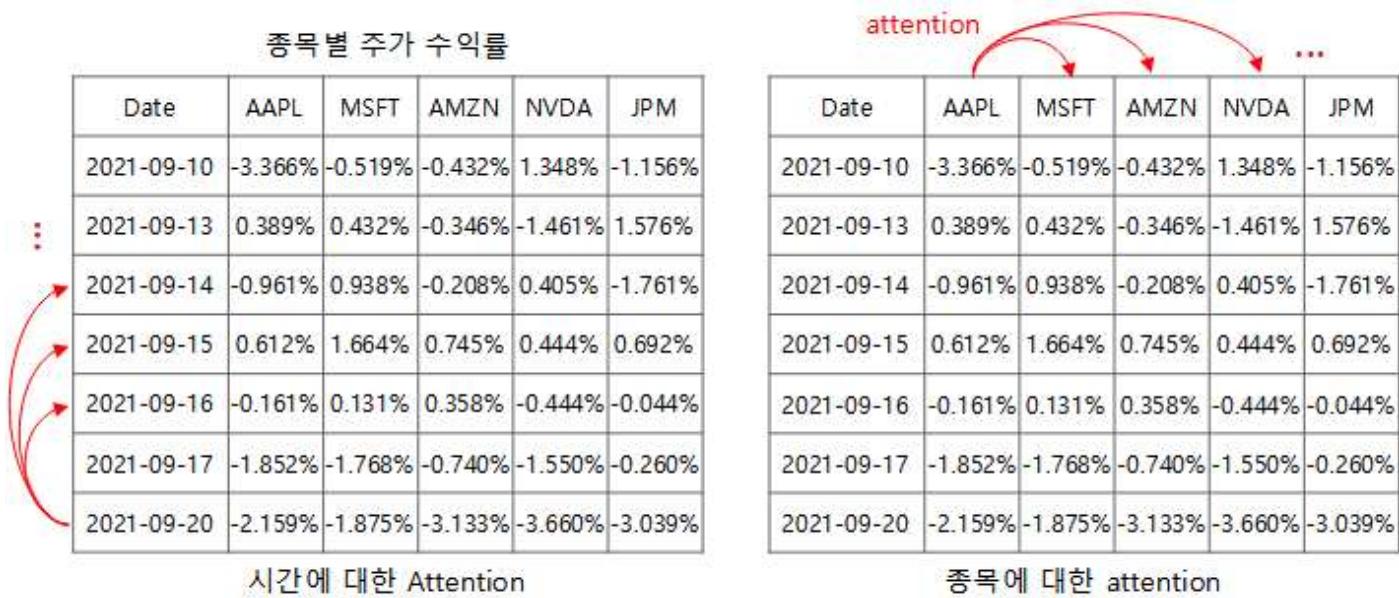
MPN의 성능을 개선할 방법은 없을까? 이전 시간에 다룬 MPN은 LSTM을 사용했다. LSTM을 사용한 이유는 주가 수익률 데이터의 시간에 대한 흐름 (sequence)을 분석하기 위함이었다. 그리고 시험 기간 동안에는 학습이 완료된 MPN을 그대로 사용했다. MPN의 성능을 개선하기 위해 두 가지 방안을 생각해 본다.

첫째, LSTM 보다 더 효율적인 네트워크를 생각해 본다. 자연어처리 (NLP) 분야에서 기계번역 (machine translation)이나 챗봇 (chatbot)을 만들 때 과거에는 주로 LSTM을 사용하다가 sequence-to-sequence, attention 같은 개념이 등장했고, 최근에는 2017년 구글에서 제안한 트랜스포머 (Transformer, self-attention이라고도 함) 네트워크를 주로 사용하고 있다. 게다가 사전 학습이라는 개념을 도입해서 트랜스포머를 이용한 BERT나 GPT 같은 기술을 사용하고 있다. 우리도 MPN에 LSTM 대신 트랜스포머를 사용하기로 한다.

둘째, 시험 기간 동안 기 학습된 MPN을 이용해서 리밸런싱을 수행해 나갈 때, 이미 지나간 시험 데이터는 알려진 데이터이므로 학습에 사용할 수 있다. 그러면 학습 양도 많아지고 최근의 상황이 반영되므로 MPN의 성능은 더 좋아질 것으로 기대된다.

Attention

LSTM은 데이터의 시간에 따른 흐름 (sequence)을 분석한다. 과거와 현재의 패턴을 학습하고 미래의 패턴을 추정한다. Attention은 과거와 현재의 패턴을 학습할 때 현재가 과거의 어느 시점과 더 유사한지를 판단한다. 현재 상황을 판단할 때 과거 흐름 중에 어느 시점에 더 집중 (attention의 의미)할 것인지를 판단하는 것이다. LSTM은 과거의 상황을 균등하게 취급하는데 반해 attention은 중요한 과거에 더 높은 가중치를 두는 방식이다. 따라서 attention은 흐름 분석에 있어서 LSTM보다 더 개선된 방법이라 할 수 있다.



주가 수익률 데이터에 attention을 적용하는 방식은 위의 그림과 같이 두 가지 방식이 있다. 왼쪽 그림은 과거와 현재의 attention이고 위에서 설명한 방식이다. 오른쪽 그림은 시간의 흐름이 아닌 종목 간의 attention이다. 종목의 수익률을 분석할 때 어느 종목의 수익률이 더 중요한지에 대해 가중치를 주는 방식이다. 물론 이 두 가지 방법을 혼합해서 사용할 수도 있다. 여기에서는 시간에 대한 attention을 살펴보기로 한다.

트랜스포머 (Transformer)

트랜스포머는 2017년 구글의 자연어처리 연구팀이 “Attention is all you need”라는 논문으로 발표한 알고리즘이다. 이 논문이 발표되기 전까지는 자연어처리를 위해 LSTM + attention 형태가 많이 사용됐는데, 여기서 LSTM을 제거하고 attention만을 이용한 것이다. 물론 LSTM을 단순히 제거한 것은 아니다. LSTM 대신 데이터의 흐름을 분석하는 부분은 추가됐다 (positional encoding이라는 부분이 추가됨). 아래의 왼쪽 그림은 트랜스포머의 네트워크 구조다.

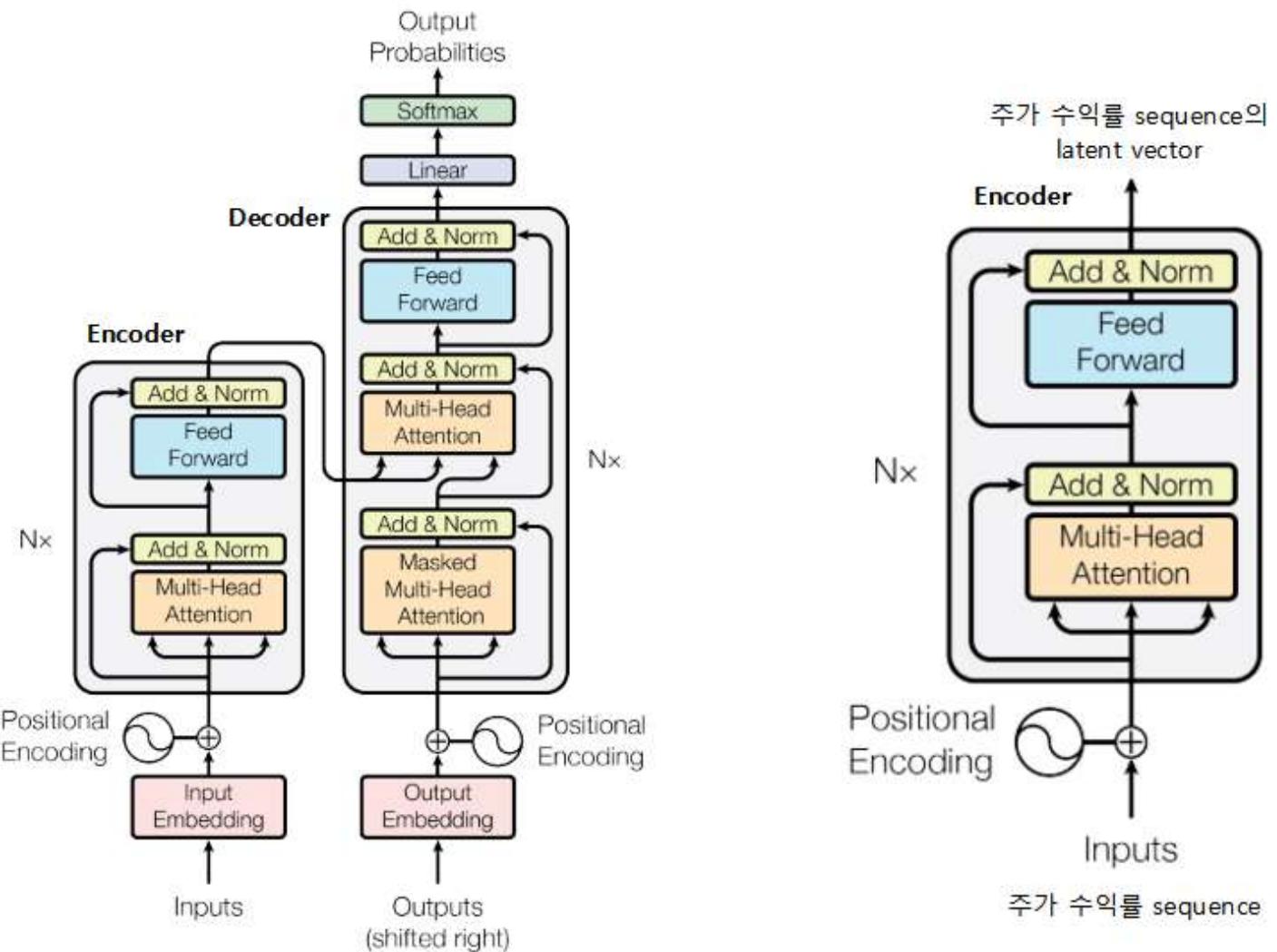


Figure 1: The Transformer - model architecture.

트랜스포머는 encoder와 decoder로 구성된다. 자연어처리에서 한글 → 영어 번역기를 만들 때 encoder에서는 한글 문장을 입력 받아 latent vector로 변환한 다음 decoder에서 latent vector로 영어 문장을 생성한다. 시계열 분석에 트랜스포머를 적용하려면 encoder 부분만 사용한다. Encoder만 사용하면 LSTM과 같이 시계열 데이터에 대한 latent vector를 생성할 수 있다. 단, 자연어처리용으로 사용되는 “Input Embedding” 부분은 제거한다. Encoder 내부의 “Multi-Head Attention” 부분에서 시간 순서에 대한 수익률 흐름의 attention이 적용된다.

트랜스포머로 구성한 MPN 코드는 아래와 같다. 트랜스포머 코드는 <https://github.com/suyash/transformer>를 이용했다. 이 코드에서 Encoding class 부분만 사용했고, “Input Embedding” 부분은 제거했다. Encoder는 3D 데이터를 입력 받아 3D 데이터를 출력한다 (batch 차원 포함). 3D 출력에 GlobalAveragePooling1D를 적용해서 2D 형태의 latent vector를 생성했다 (batch 차원을 빼면 1D 출력임). 나머지 부분은 LSTM의 MPN과 동일하다.

```

from transformer import Encoder

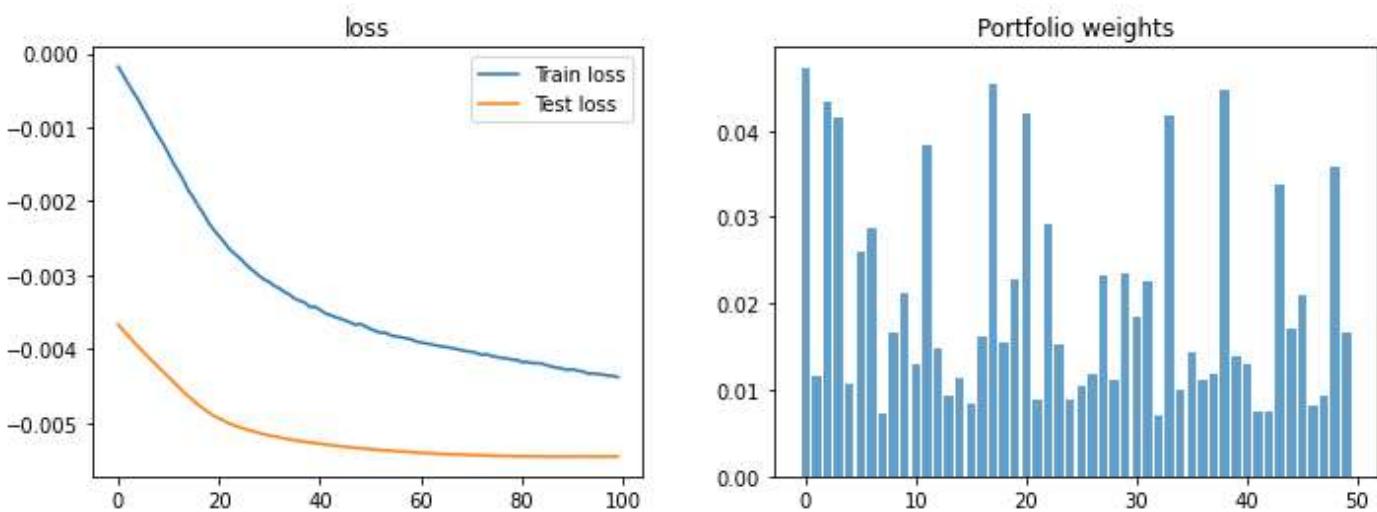
xc_input = Input(batch_shape = (None, N_TIME, N_STOCKS))

# 트랜스포머의 Encoder
enc_output, _ = Encoder(num_layers=1, d_model=N_STOCKS, num_heads=5, d_ff=100, dropout_rate=0.3)(xc_input)
y_output = GlobalAveragePooling1D()(enc_output)
y_output = Activation('tanh')(y_output)

# 마코비츠의 최적 weights
y_output = Activation('softmax')(y_output)

model = Model(xc_input, y_output)
model.compile(loss = markowitz_objective, optimizer = Adam(learning_rate = 1e-5))
model.summary()

```



학습 결과는 위 그림과 같다. Train과 test 데이터의 loss가 잘 감소하고 있고, 포트폴리오의 최적 비율 (weights)도 (육안 상으로) 잘 나오고 있다.

이번에는 두 번째 개선 사항인 시험 기간 동안의 추가 학습에 대해 살펴본다. 기존 MPN의 성능평가 프로그램에 시험 기간 동안 추가로 발생한 시장 데이터를 추가로 학습하는 부분을 추가했다. 추가 학습은 지나간 시험 데이터와 기존 학습 데이터의 일부를 사용했다. 지나간 시험 데이터만 사용하면 MPN이 최근의 상황만 지나치게 반영할 것이기 때문에 과거 데이터인 기존 학습 데이터도 같이 사용했다.

```
# MPN을 이용해서 백 테스트를 수행한다.
mpn_value = [10000]    # portfolio의 초기 value
crp_value = [10000]    # CRP의 초기 value
w_crp = np.ones(N_STOCKS) / N_STOCKS    # CRP 비율 (균등 비율)

w_hist = []
for i in range(0, xc_test.shape[0], N_FUTURE):
    # MPN으로 W를 추정한다.
    x = xc_test[i][np.newaxis, :, :] * 20.0
    w_prt = model.predict(x)[0]
    w_hist.append(w_prt)

# 다음 기의 누적 수익률
m_rtn = np.sum(xf_test[i], axis = 0)

# 누적 수익률과 w_prt (W)로 포트폴리오의 수익률을 계산한다.
mpn_value.append(mpn_value[-1] * np.exp(np.dot(w_prt, m_rtn)))
crp_value.append(crp_value[-1] * np.exp(np.dot(w_crp, m_rtn)))

# 추가로 발생한 시장 데이터로 MPN을 추가 학습시킨다.
# xc_test[0] ~ xc_test[19].
# xf_test[0] ~ xf_test[19]를 추가로 학습시킬 수 있다.
xc_new = xc_test[i:(i+N_FUTURE), :, :]
xf_new = xf_test[i:(i+N_FUTURE), :, :]

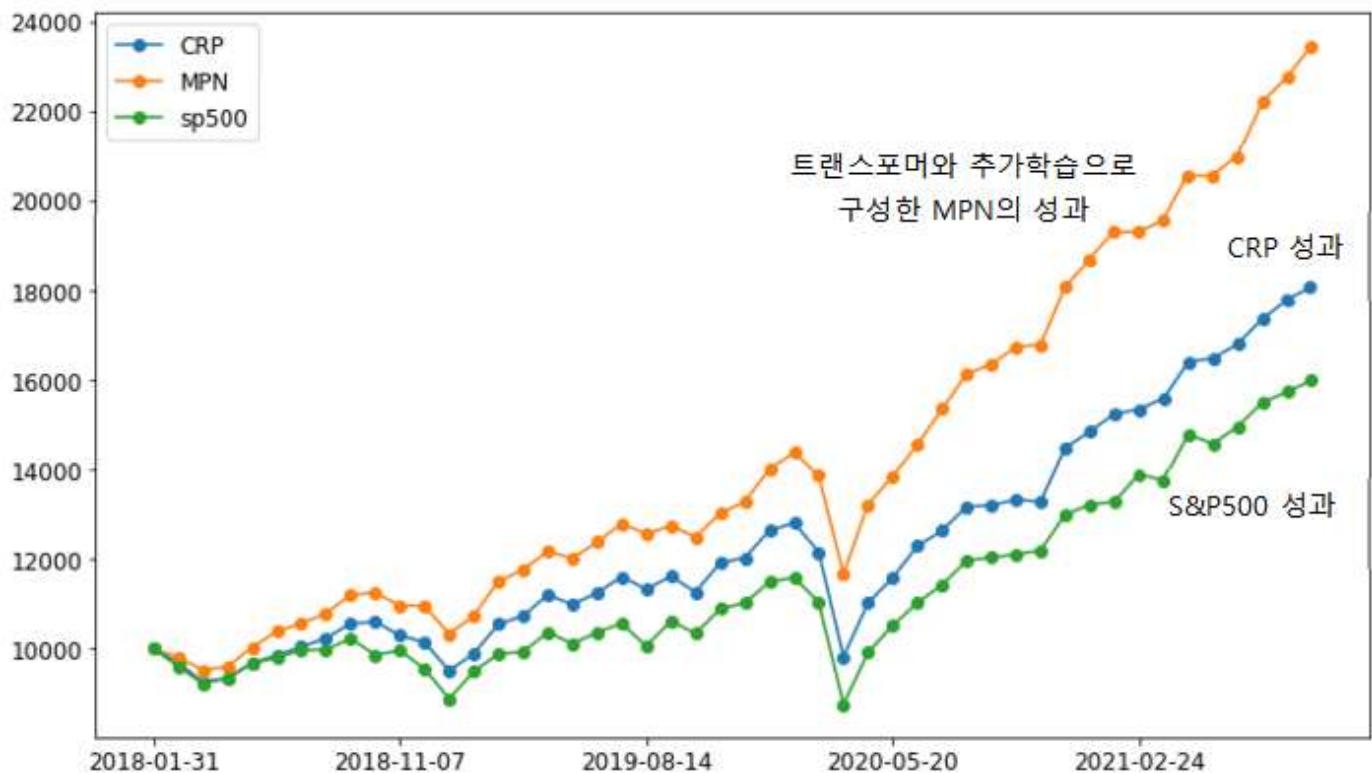
# xc_train 데이터에서 80개를 random sampling 한다.
idx = np.random.randint(0, xc_train.shape[0], 80)

# 추가 학습 데이터를 생성한다.
x = np.vstack([xc_new, xc_train[idx]])
y = np.vstack([xf_new, xf_train[idx]])
x, y = shuffle(x, y)

# 추가 학습한다.
x *= 20.0
y *= 20.0
model.fit(x, y, epochs=50, batch_size=10, verbose=0)

# 추가로 발생한 데이터도 이후에 sampling될 수 있도록 보관해 둔다.
xc_train = np.vstack([xc_train, xc_new])
xf_train = np.vstack([xf_train, xf_new])
```

성능 평가 결과는 아래와 같다. 기존의 MPN보다 성능이 개선됐다. 기존 MPN에서는 초기 자본이 약 1.8배 상승했으나, 이 결과에서는 약 2.4배 상승했다.



이번 글에서는 트랜스포머와 추가학습을 통해 MPN을 개선해 보았다. 사실, 성능이 개선된 영향은 추가학습 부분이 더 컸다. 트랜스포머를 사용하지 않고 LSTM과 추가학습을 사용해도 위와 유사한 정도의 결과가 나온다.

여기까지 MPN에 대해 모두 살펴 보았고, 마코비츠의 포트폴리오 이론에 딥러닝을 적용해 보았다. 다음 시간부터는 요인모형 (Factor Model in finance)에 딥러닝을 적용하는 방법을 살펴보기로 하겠다.