

RNN, LSTM, GRU로 예측해 본 삼성전자 주가

2023 Q1-2024 Q4를 바탕으로

4조_여태양, 한수빈, 김나현

목차

① 주제 소개

삼성전자 주가 분석을 통한 LSTM과 RNN의 비교

② 사용 알고리즘

RNN, LSTM, RNN-GRU 알고리즘 소개

③ 데이터셋

사용 데이터셋 소개

④ 코드

사용한 코드 소개

⑤ 분석 결과 시각화

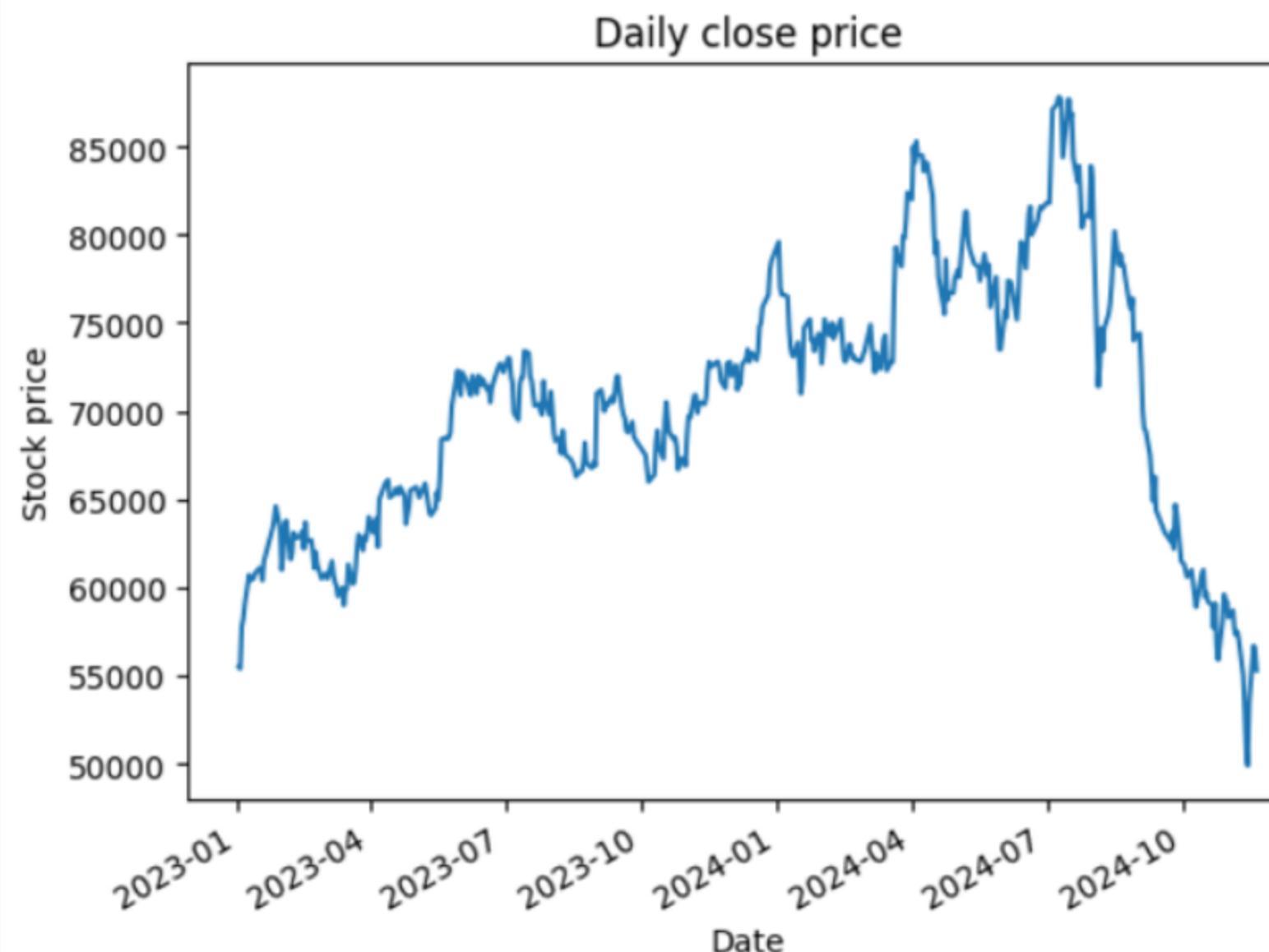
알고리즘 시각화

⑥ INSIGHT

해당 프로젝트를 통해 도출한 insight 도출

01 주제소개

삼성전자 주가 분석 및 예측을 통한 LSTM과 RNN의 비교



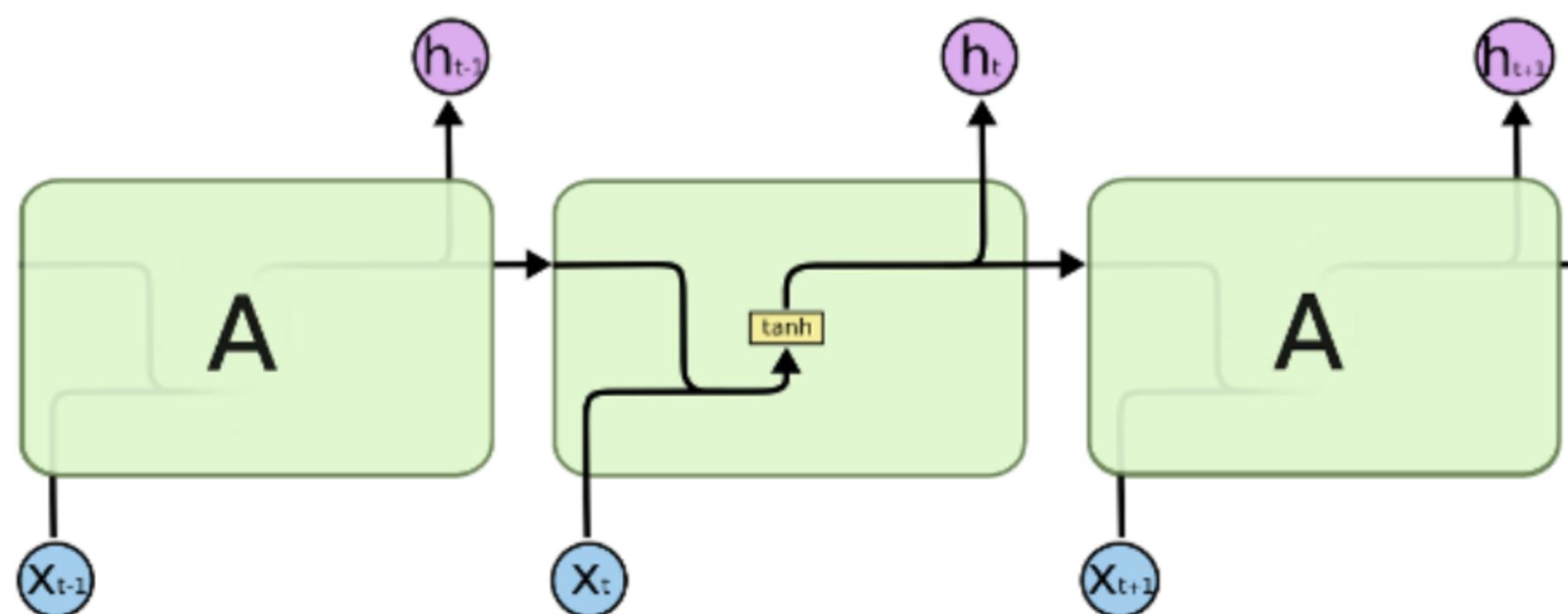
종목명 005930/삼성전자

조회기간 20230102 ~ 20241125

2023 1분기-2024 4분기까지의 삼성전자 주가

2023 1분기-2024 1분기 주가 데이터를 바탕으로 RNN, LSTM 알고리즘의 성과 파악, 실제로 LSTM이 RNN보다 우수한 성능을 보이는가를 중점적으로 탐구

RNN과 LSTM, GRU

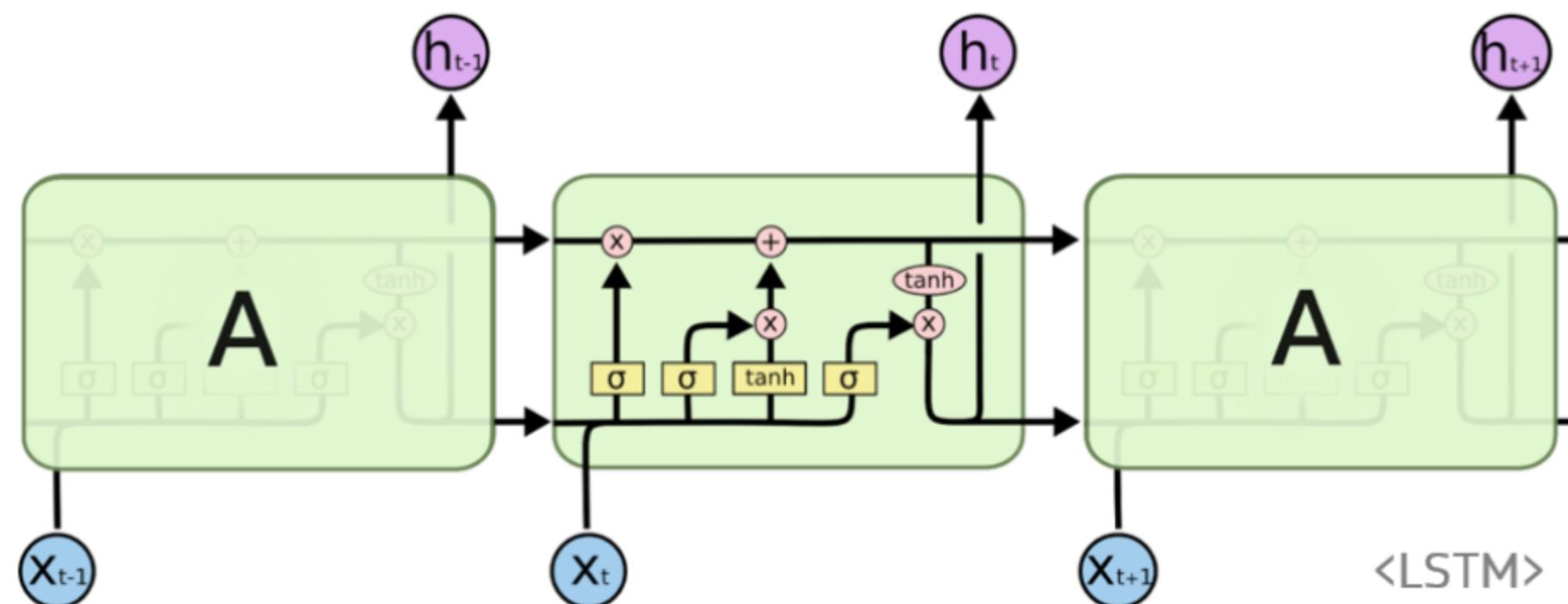


RNN

신호를 순환하여 상호 관계가 있는, 시계열 신호와 같은 것들을 처리하는 인경신공망

1. 기존 Neural Network와는 달리 기억을 가지고 있다는 것이 그 특징
2. 은닉 계층의 연산을 바로 출력 계층으로 전달하지 않고 데이터를 한번 더 사용
3. 오래된 데이터는 소멸되어 장기 기억이 필요한 부분에서 효력을 발휘할 수 없는 문제가 발생

RNN과 LSTM, GRU

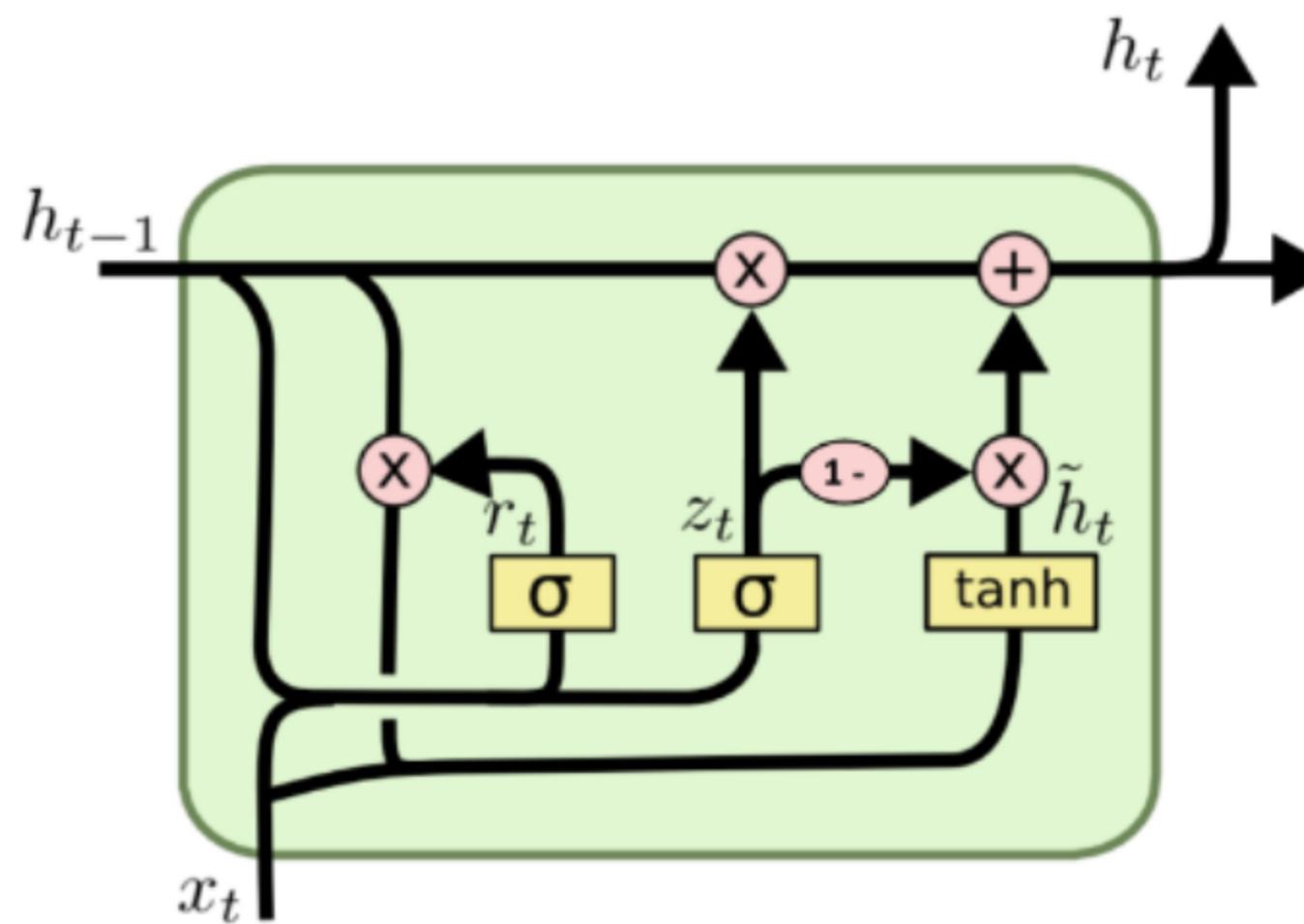


LSTM

RNN에서 forget, input, output 3가지 게이트를 추가

- 각각 불필요한 정보 잊음, 새 정보 선택적 추가, 필요한 정보 출력
- 기존 RNN의 한계인 오랜 데이터 기억 손실 문제를 개선
→ 장기의존성 문제 해결

RNN과 LSTM, GRU



RNN - GRUCell

LSTM을 간소화 시킨 것

1. LSTM의 forgrt, input, output gate 3개의 게이트를 update, reset 2개의 gate로 줄인 모델
2. LSTM보다 파라미터 수가 적고 복잡하지 않아 연산이 빠르다는 장점

03 데이터셋

데이터셋

일자	시가	고가	저가	거래량	거래대금	시가총액	상장주식수	종가
2023-01-02	55500	56100	55200	10031448	5.58E+11	3.31E+14	5.97E+09	55500

•
•
•

2024-11-20	56100	56500	54800	20864667	1.16E+12	3.30E+14	5.97E+09	55300
------------	-------	-------	-------	----------	----------	----------	----------	-------

- 데이터 구성: 2023년 1분기-2024년 11월까지의 삼성전자 주가
- train: 2023.01.02 ~ 2024.06.28
- test: 2024.07.01 ~ 2024.11.20



훈련 데이터로 훈련한 알고리즘의 예측 결과를 24.07.01~24.11.20의
실제 주가와 비교, 얼마나 예측 성과가 우수한지 비교하려 함

03 데이터셋

	시가	고가	저가	거래량	거래대금	시가총액	상장주식수	종가
count	462.000000	462.000000	462.000000	4.620000e+02	4.620000e+02	4.620000e+02	4.620000e+02	462.000000
mean	70502.597403	71095.887446	69809.090909	1.749686e+07	1.233468e+12	4.202641e+14	5.969783e+09	70399.567100
std	7447.726273	7458.376500	7355.580144	8.404638e+06	6.043214e+11	4.440715e+13	0.000000e+00	7435.515978
min	50200.000000	51800.000000	49900.000000	5.824628e+06	3.890000e+11	2.980000e+14	5.969783e+09	49900.000000
25%	65000.000000	65425.000000	64225.000000	1.174975e+07	8.155000e+11	3.862500e+14	5.969783e+09	64750.000000
50%	71200.000000	71650.000000	70500.000000	1.492119e+07	1.050000e+12	4.240000e+14	5.969783e+09	71050.000000
75%	75000.000000	75875.000000	74200.000000	2.034296e+07	1.517500e+12	4.480000e+14	5.969783e+09	75075.000000
max	88500.000000	88800.000000	87100.000000	5.769127e+07	4.210000e+12	5.240000e+14	5.969783e+09	87800.000000

처음
데이터셋

- 처음 데이터셋 attribute: 일자, 시가, 고가, 저가, 거래량, 거래대금, 시가총액, 상장주식수, 종가
- 상장주식수는 min=max, 정규화 시 문제가 되기 때문에 drop

03 데이터셋

일자	시가	고가	저가	종가	일자	시가	고가	저가	거래량	거래대금	시가총액	종가
	55500	56100	55200	55500		55400	56000	54500	55400	10031448	5.580000e+11	3.310000e+14
2023-01-02	55500	56100	55200	55500	2023-01-02	55500	56100	55200	10031448	5.580000e+11	3.310000e+14	55500
2023-01-03	55400	56000	54500	55400	2023-01-03	55400	56000	54500	13547030	7.480000e+11	3.310000e+14	55400
2023-01-04	55700	58000	55600	57800	2023-01-04	55700	58000	55600	20188071	1.150000e+12	3.450000e+14	57800
2023-01-05	58200	58800	57600	58200	2023-01-05	58200	58800	57600	15682826	9.120000e+11	3.470000e+14	58200
2023-01-06	58300	59400	57900	59000	2023-01-06	58300	59400	57900	17334989	1.020000e+12	3.520000e+14	59000

```
stock_array = stock.to_numpy() # numpy 배열로 변환  
stock_array
```

```
array([[5.5500000e+04, 5.6100000e+04, 5.5200000e+04, ...,  
       3.3100000e+14, 5.96978255e+09, 5.5500000e+04],  
      [5.5400000e+04, 5.6000000e+04, 5.4500000e+04, ...,
```

전처리

- 시가, 고가, 저가, 거래량, 거래대금, 시가총액, 종가 7개 변수로 분석
- 시가, 고가, 저가, 종가 4개 변수로 분석
- 딥러닝 모델 입력을 위해 numpy 배열로 변환

04 코드

```
seq_len = 10 # 10개씩 넣기
data_dim = 4 # 컬럼수
hidden_dim = 10
output_dim = 1
l_rate = 0.01
epochs = 300
```

```
x_data = []
y_data = []
total_len = len(y)-seq_len

for i in range(total_len):
    x1 = x[i:i+seq_len]
    y1 = y[i+seq_len]

    x_data.append(x1)
    y_data.append(y1)

x_data = np.array(x_data)
y_data = np.array(y_data)

# train / test 분리
split_index = stock.index.get_loc('2024-06-28') # 끝짜 기준으로 행 번호 찾기
x_train, x_test = x_data[:split_index], x_data[split_index:] # 2019~2024.2분기
y_train, y_test = y_data[:split_index], y_data[split_index:] # 2024.3분기~
```

```
model_lstm.compile(optimizer=tf.keras.optimizers.Adam(l_rate), loss='mse')

model_rnn_gru.compile(optimizer=tf.keras.optimizers.Adam(l_rate), loss='mse')

from tensorflow.keras.layers import RNN, SimpleRNNCell

model_rnn = tf.keras.Sequential([
    RNN(SimpleRNNCell(hidden_dim), return_sequences=True, input_shape=(seq_len, data_dim)),
    RNN(SimpleRNNCell(hidden_dim)),
    tf.keras.layers.Dense(output_dim)
])

model_rnn.compile(optimizer=tf.keras.optimizers.Adam(l_rate), loss='mse')

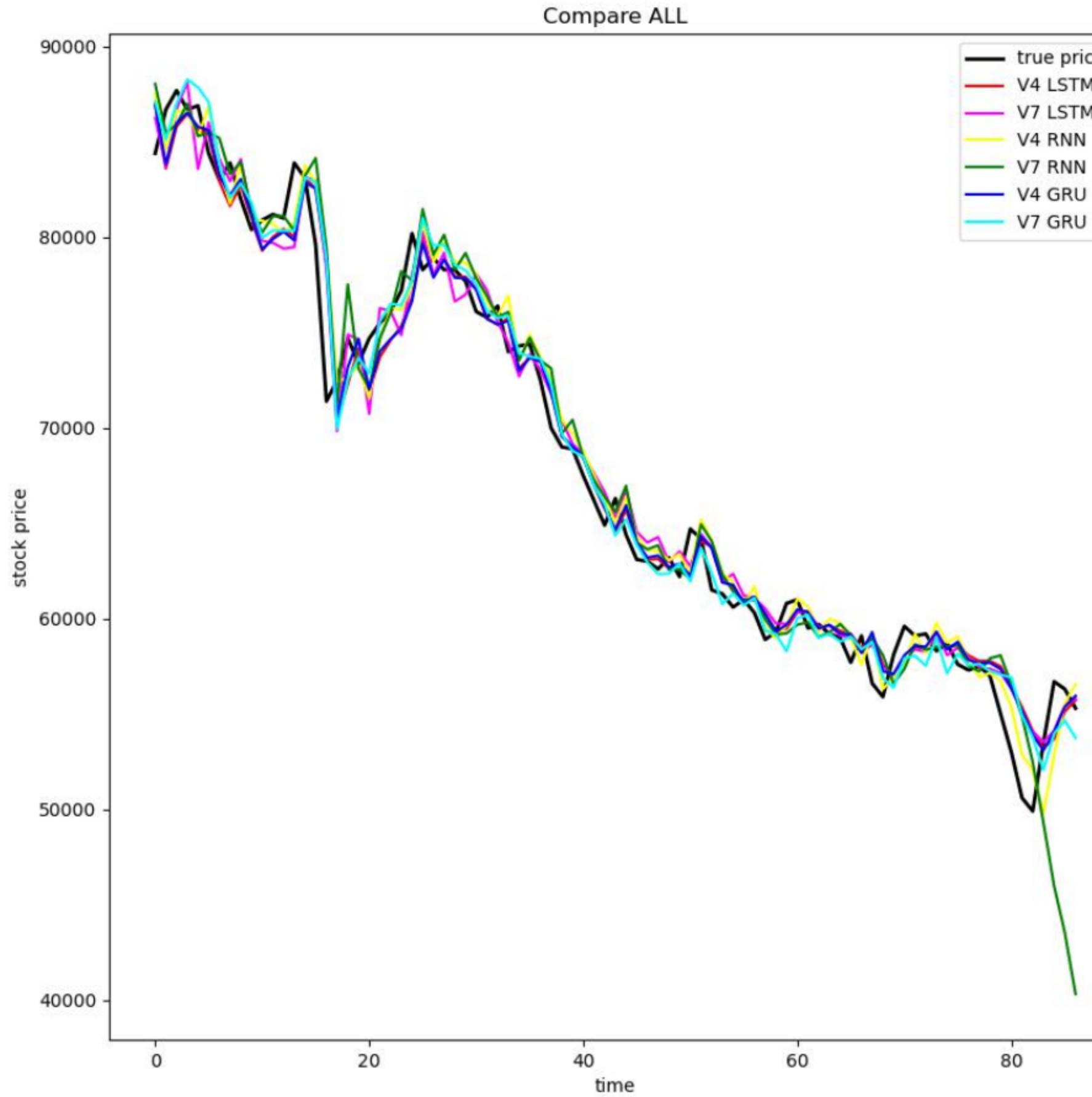
# 학습
history_rnn = model_rnn.fit(x_train, y_train,
                             epochs=epochs,
                             batch_size=32,
                             verbose=2)

# 예측
test_predict_rnn = model_rnn.predict(x_test)

test_predict_rescaled_rnn = test_predict_rnn*(S_max[-1]-S_min[-1])+S_min[-1]
y_test_rescaled_rnn = y_test*(S_max[-1]-S_min[-1])+S_min[-1]

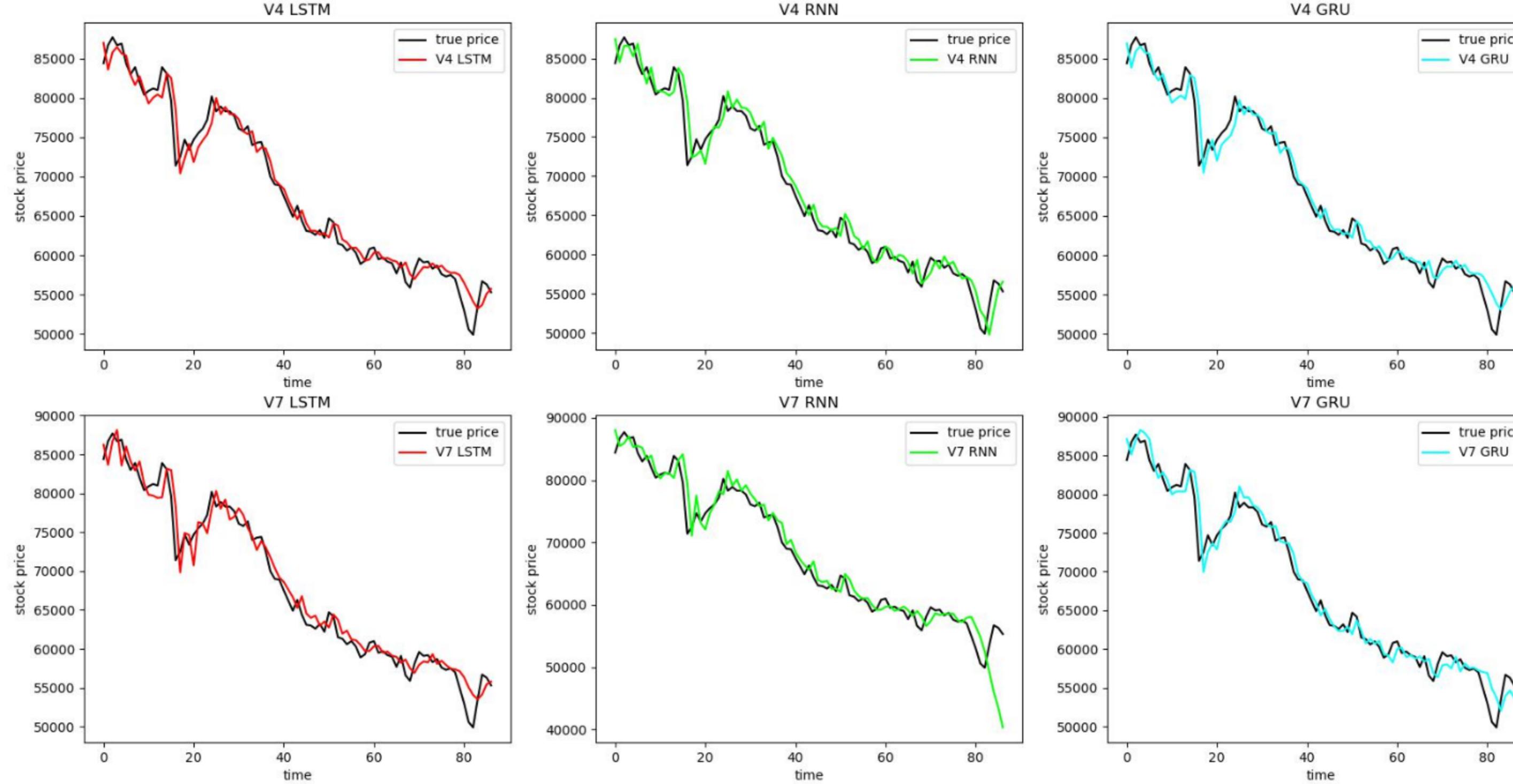
test_error_rnn = np.sqrt(np.mean(np.square(y_test_rescaled_rnn - test_predict_rescaled_rnn)))
print("test_error: {}".format(test_error_rnn))
```

05 시각화



- 4개 변수, 7개 변수 Compare ALL
- V7 Simple-RNN의 예측이 크게 어긋남(test_error가 큼)
- 다른 알고리즘은 true price와 비슷한 움직임을 보임

05 시각화

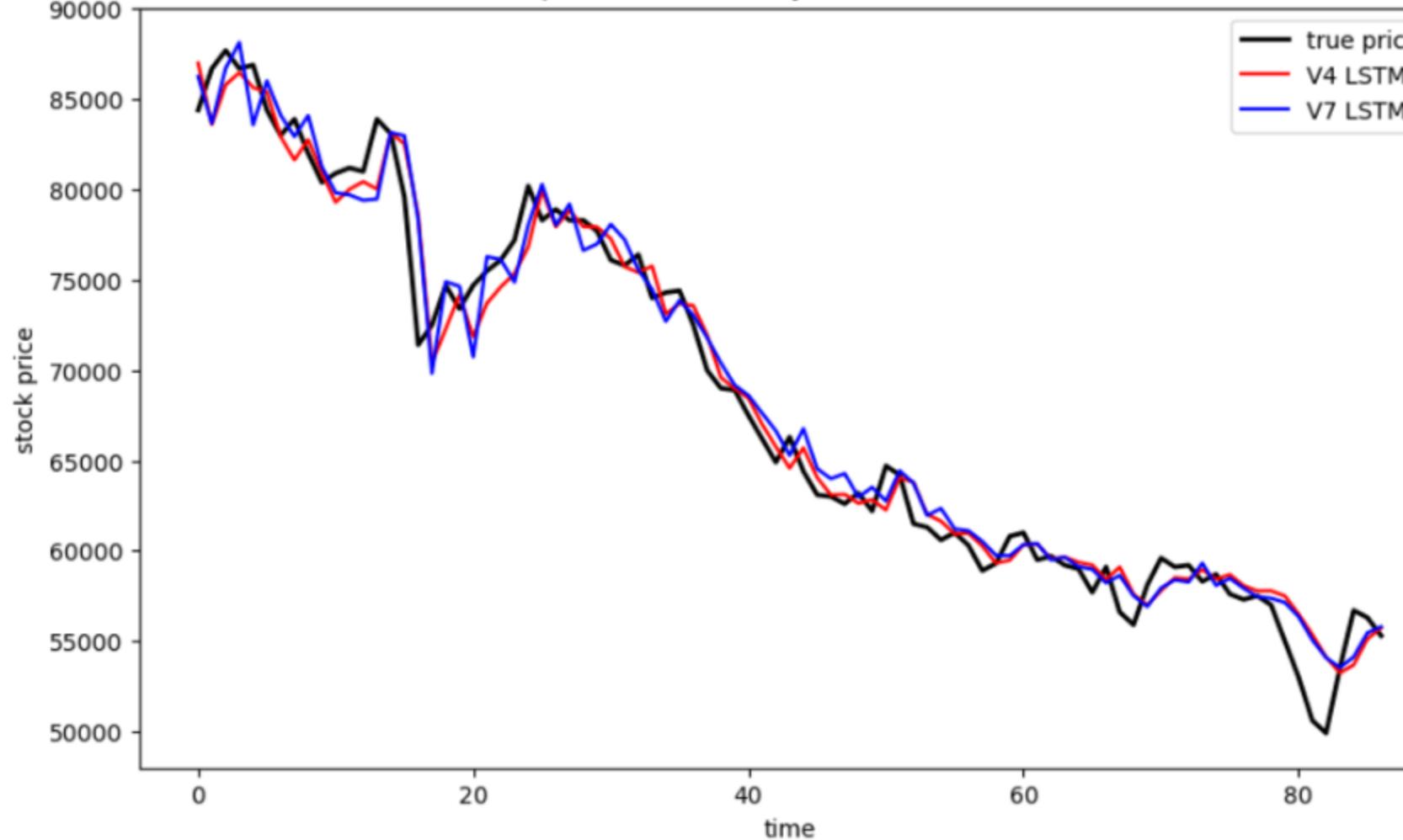


시각화

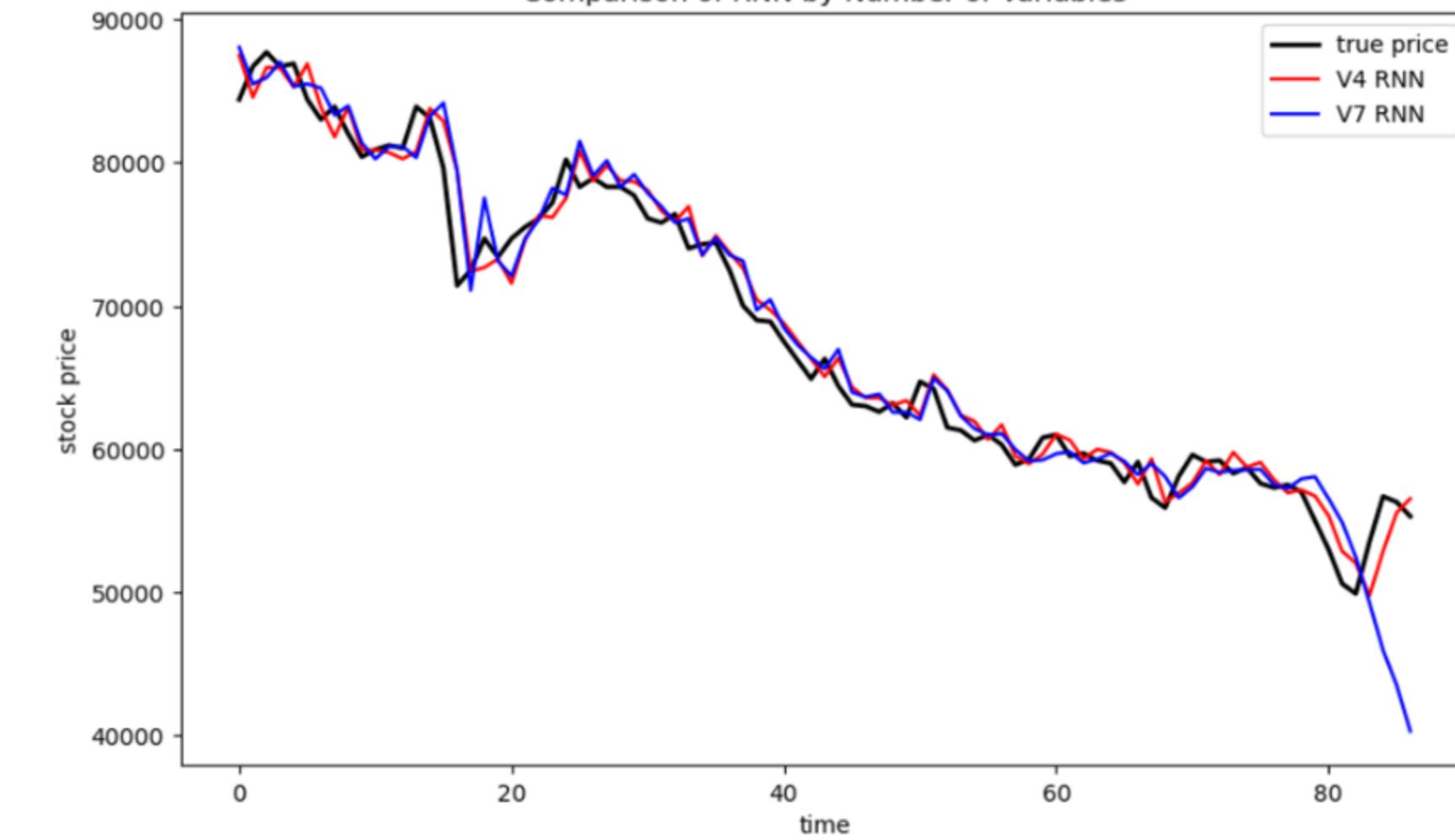
1. 시가, 고가, 저가, 종가 4개 변수로 돌려본 LSTM, RNN, GRU
2. 시가, 고가, 저가, 거래량, 거래대금, 시가총액, 종가 7개 변수로 돌려본 LSTM, RNN, GRU

05 시각화

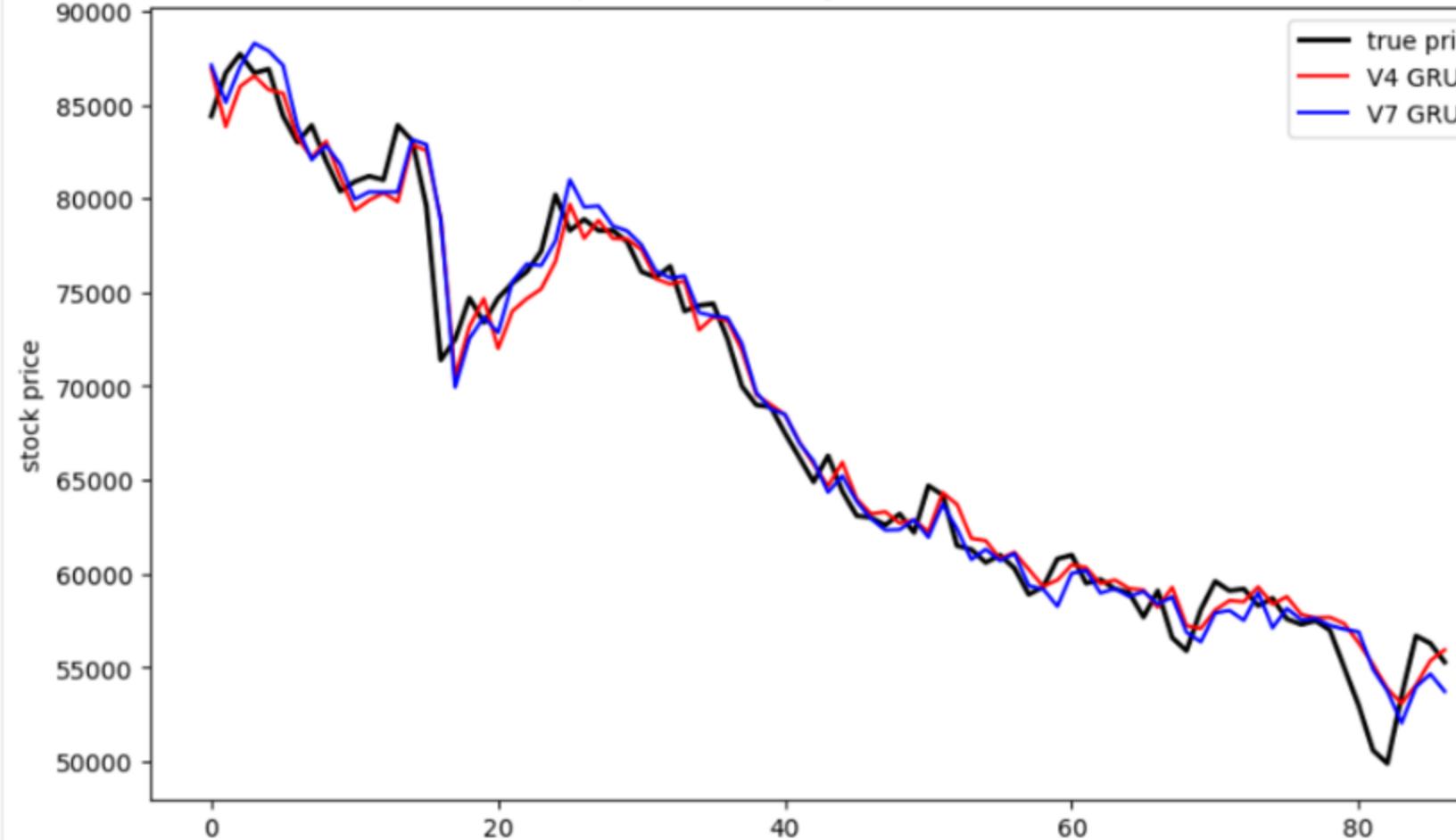
Comparison of LSTM by Number of Variables



Comparison of RNN by Number of Variables



Comparison of GRU by Number of Variables

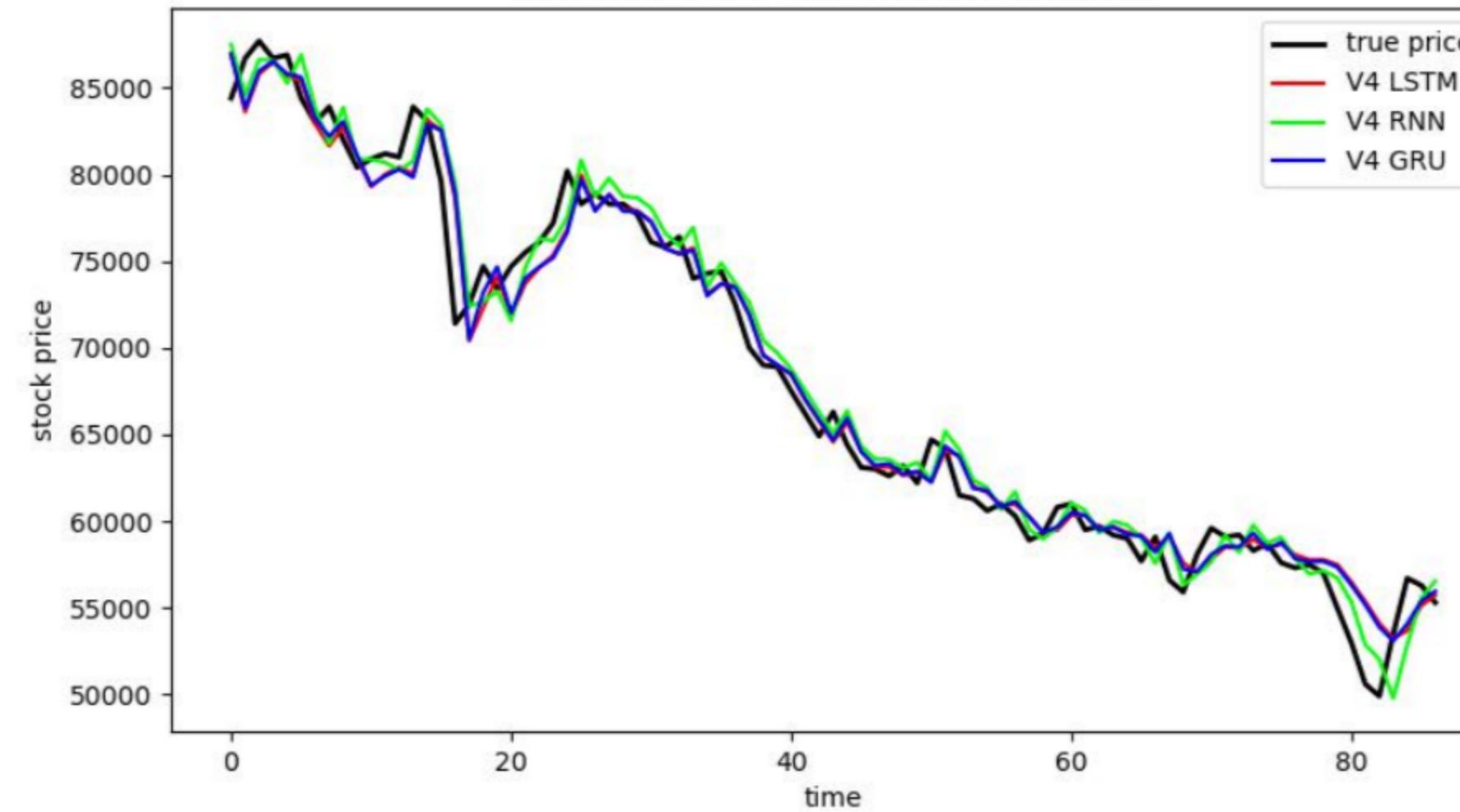


- 4개 변수, 7개 변수에 따른 알고리즘 결과의 차이점
- V4, V7 LSTM 의 예측 차이 거의 X
- V4, V7 RNN의 예측 차이 큼. V7 RNN의 예측이 갈수록 크게 틀림
- V4, V7 GRU의 예측 차이 거의 X

RNN 제외, 변수 간 차이는 크게 없음

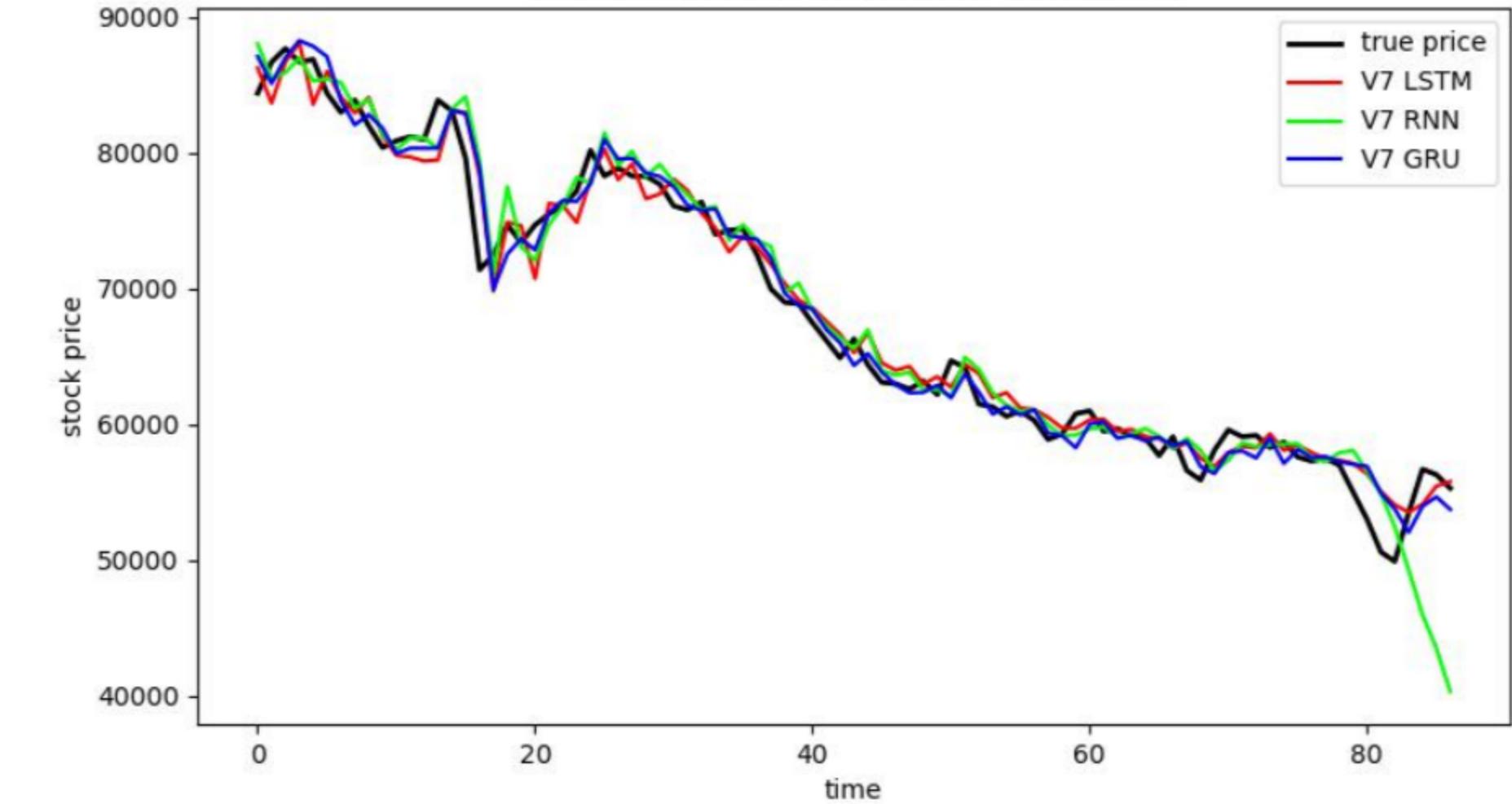
05 시각화

Comparison of 4 Variables by Algorithms



4개 변수의 알고리즘 간 차이점

Comparison of 7 Variables by Algorithms



7개 변수의 알고리즘 간 차이점

- 4개 변수에서 알고리즘 간 유사한 모습을 보임
- 7개 변수에서 LSTM과 GRU는 유사한 모습, SimpleRNN은 예측에서 크게 벗어난 모습을 보임

06 INSIGHT

V4 LSTM

2024.11.21

실제 종가: 56400

예측 종가: 55726.656 (-673.344)

test_error: 1797.16020347792

4개 변수

V4 RNN

2024.11.21

실제 종가: 56400

예측 종가: 56540.15 (+140.15)

test_error: 1817.439594594754

V4 GRU

2024.11.21

실제 종가: 56400

예측 종가: 55949.42 (-450.58)

test_error: 1765.7412408948464

V7 LSTM

2024.11.21

실제 종가: 56400

예측 종가: 55789.113 (-610.887)

test_error: 1850.7587495375221

7개 변수

V7 RNN

2024.11.21

실제 종가: 56400

예측 종가: 40315.74 (-16,084.26)

test_error: 3074.61688647654362

V7 GRU

2024.11.21

실제 종가: 56400

예측 종가: 53739.766 (-2,660.234)

test_error: 1761.87569779883877

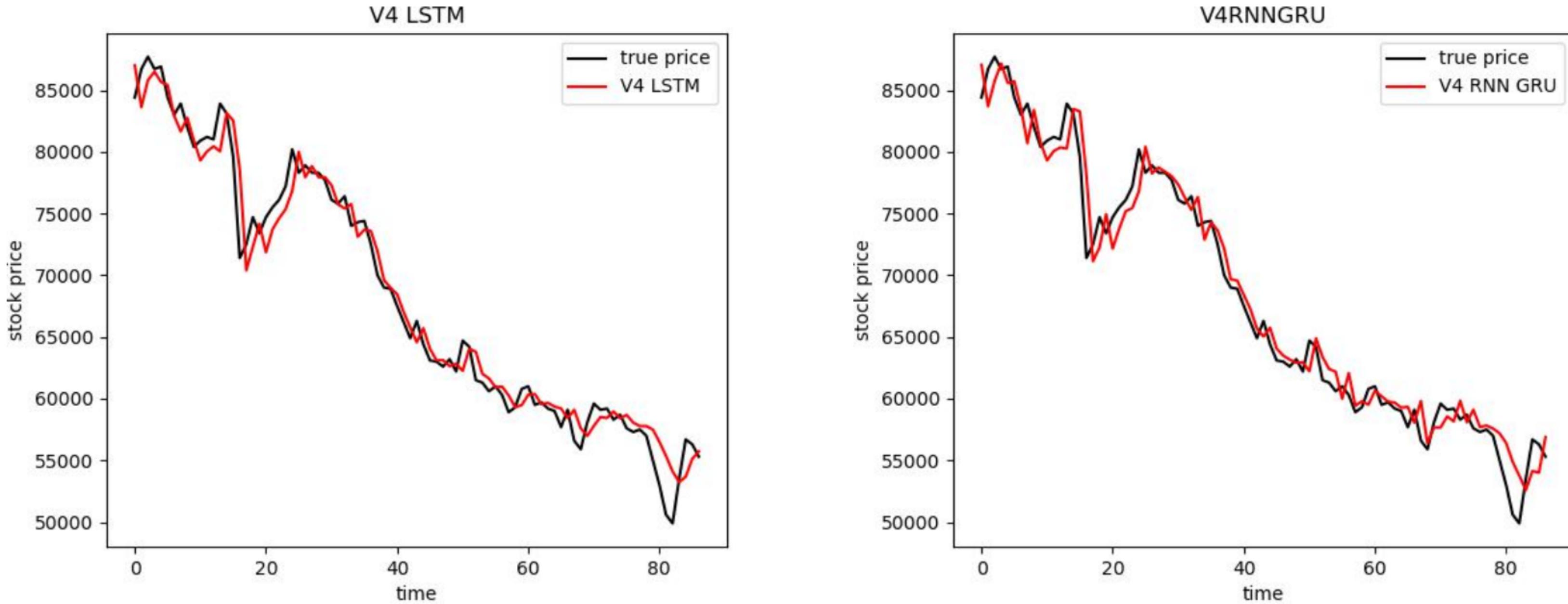
1 LSTM과 RNN-GRUCell은 유의미한 성능차이를 보이지 않으며
둘 모두 우수한 성과, SimpleRNN은 상대적으로 test_error가 큼

2 4개 변수, 7개 변수에서 LSTM과 RNN-GRUCell은 test_error의
차이가 미미함. 하지만 RNN은 7개 변수에서 test_error 커짐

3 7개 변수의 SimpleRNN은 장기의존성 문제로 데이터가 복잡해
지므로, 모델이 효율적으로 정보를 처리하지 못하고 성능이 저하
된 것으로 보임

4 SimpleRNN < LSTM < RNN-GRUCell 순으로 test_error가
낮음
V7<V4, 전반적으로 변수가 적은 것이 성능이 좋음

06 INSIGHT



LSTM과 RNN-GRUCell

- LSTM과 GRU는 성능 측면에서 큰 차이 X
- 데이터가 많을수록 LSTM이, 데이터가 적을수록 GRU의 성능이 더 좋을 것으로 예측했지만 4개 변수, 7개 변수 모두 GRU의 성능이 소폭 우세
- LSTM과 GRU는 Task에 따라 어떤 것이 성능이 좋을지 모르기 때문에 두 개 모두 돌려보고 결과가 좋은 cell을 사용하는 것이 바람직하다는 결론

결론

● SimpleRNN<LSTM<=GRU

- Simple RNN의 한계를 직접 확인, 특히 변수가 많아질수록 성능이 떨어지는 것이 확연함
- 데이터가 많을수록 LSTM이, 데이터가 적으면 GRU의 성능이 더 좋은 경향이 있다고 해 4개 변수에서는 LSTM이 7개 변수에서는 GRU가 우수할 것으로 예측했지만 모두 GRU가 성능 우수->직접 돌려서 결과 확인 전까지는 알 수 없음
- 전반적으로 4개 변수의 예측 성능이 우수했으며
훈련 데이터로 예측한 price와 true price가 거의 유사하여 생각했던 것보다 예측 성능이 뛰어났음

감사합니다

RNN, LSTM, GRU로 예측해 본 삼성전자 주가