RNN을 통한 주가예측

20212245 김희서
2021.12.18

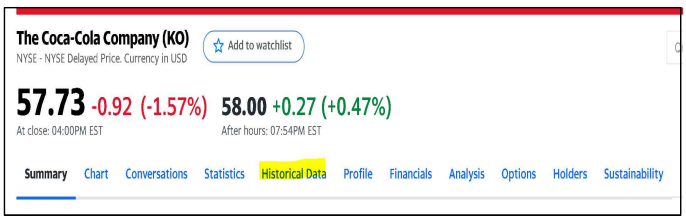
github-code
https://github.com/kimheeseo/deeplearning_project2_RNN

data : coca 주가

- 데이터 개수의 조건이 있기에, 그 조건을 갖춘 주가 데이터를 위해서는 오랜 기간 상장 된 기업이어야 하는 데, 코카콜라가 그 조건을 성립하기에 선정하였습니다.

주가 데이터는 야후 파이낸스에서 추출하였습니다.

방법 : https://finance.yahoo.com/



- 왼쪽 사진처럼 야후 파이낸스에서 주가 데이터를 추출하기 위해서는 “Historical Data”를 클릭합니다. 그리고 조건을 설정합니다. 조건은 “time Period”을 설정하고, Apply 후 다운로드합니다.

그러면 주가 데이터를 추출할 수 있습니다. 다음과 같이“.CSV”파일이 추출됩니다.

	A	B	C	D	E	F	G
1	Date	Open	High	Low	Close	Adj Close	Volume
2	2020-12-18	53.08	53.84	52.62	53.74	52.09757	35480100
3	2020-12-21	52.68	53	51.98	52.81	51.19599	14926600
4	2020-12-22	52.81	53.18	52.39	52.76	51.14751	9689500
5	2020-12-23	52.95	53.39	52.94	53.08	51.45774	7040300
6	2020-12-24	53.02	53.55	53.02	53.44	51.80673	3265500
7	2020-12-28	53.85	54.44	53.73	54.16	52.50473	9020500
8	2020-12-29	54.45	54.49	54.02	54.13	52.47564	8320600
9	2020-12-30	54.05	54.63	54.03	54.44	52.77617	8142700
10	2020-12-31	54.45	54.93	54.27	54.84	53.16394	8495000
11	2021-01-04	54.27	54.63	52.03	52.76	51.14751	25611100
12	2021-01-05	52.33	52.62	52.03	52.18	50.58524	20323800
13	2021-01-06	51.97	52.02	50.19	50.52	48.97598	38724500

<코드분석>

```
import numpy as np
import pandas as pd
import math
import sklearn
from keras.models import Sequential
from sklearn.model_selection import train_test_split
from keras.layers import Dense, LSTM
from sklearn.preprocessing import MinMaxScaler#MinMaxScaler 표준화
import matplotlib.pyplot as plt
from google.colab import drive
drive.mount('/content/gdrive')#구글드라이브에서 파일 열기
```

```
Mounted at /content/gdrive
```

위의 "coca.scv"을 실행하기 위해서,
"from google.colab import drive"와
"drive.mount('/content/gdrive')" 작성

```
df=pd.read_csv('/content/gdrive/MyDrive/code/coca.csv',encoding='utf8')#df = 코카콜라주식
print('Wndf.info()')
df.info()
print('Wndf.head()')
df.head()#1962년부터 주가를 yahoo finance에서 조회가 가능한데, 가격이 저렴한 것이지, 아래의 값은
#MinMaxScaler의 값이 아닙니다.
```

위의 주식 가격을 나타내는 "open", "high", "low", "Adj Close", "Volume", "Close"의 경우, 소수점인 것을 확인할 수 있습니다.

df.head()

	Date	Open	High	Low	Adj Close	Volume	Close
0	1962-01-03	0.259115	0.259115	0.253255	0.049994	1574400	0.257161
1	1962-01-04	0.257813	0.261068	0.257813	0.050374	844800	0.259115
2	1962-01-05	0.259115	0.262370	0.252604	0.049234	1420800	0.253255
3	1962-01-08	0.251302	0.251302	0.245768	0.048728	2035200	0.250651
4	1962-01-09	0.250651	0.256510	0.248698	0.049614	960000	0.255208

이는 'MinMaxScaler'를 한 것이 아닌 주식 상장 초기인 1962년의 가격이기에 1달러 미만임을 나타냅니다.

df.tail()

	Date	Open	High	Low	Adj Close	Volume	Close
15081	2021-11-30	53.599998	53.630001	52.439999	52.450001	30485200	52.450001
15082	2021-12-01	52.980000	53.520000	52.279999	52.299999	18719600	52.299999
15083	2021-12-02	52.599998	53.340000	52.509998	53.070000	17074200	53.070000
15084	2021-12-03	53.330002	53.610001	52.980000	53.540001	21062400	53.540001
15085	2021-12-06	54.310001	55.250000	54.139999	54.910000	26622900	54.910000

```
print('describe값')
df.describe()
```

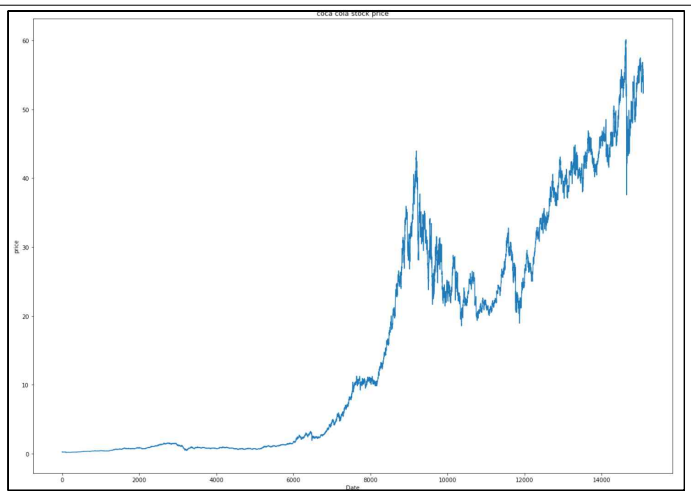
describe값

	Open	High	Low	Adj Close	Volume	Close
count	15086.000000	15086.000000	15086.000000	15086.000000	1.508600e+04	15086.000000
mean	16.033093	16.165045	15.898840	11.341124	9.023160e+06	16.037467
std	16.915488	17.040094	16.787879	14.214715	7.923026e+06	16.917524
min	0.192708	0.193359	0.182292	0.037855	7.680000e+04	0.192057
25%	0.859700	0.869792	0.854167	0.242306	2.812800e+06	0.859375
50%	9.226562	9.367188	9.187500	4.611509	7.575200e+06	9.250000
75%	28.844687	29.143750	28.534063	17.220101	1.290605e+07	28.873750
max	59.810001	60.130001	59.619999	56.610435	1.241690e+08	60.130001

데이터의 세부사항을 확인하기 위해서, tail과 describe를 사용하였습니다.

그래프 출력하기

```
plt.figure(figsize=(20,15))
plt.title('coca cola stock price')
plt.plot(df['Close'])#종가를 기준으로 주가 그래프 그리기
plt.xlabel('Date')
plt.ylabel('price')
plt.show()
```



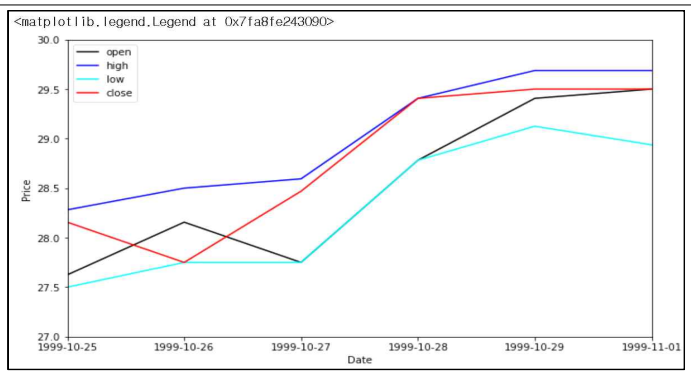
사용한 "coca.csv" 파일의 값을 출력 했습니다.
x축의 값은 date이고, y축은 주가를 나타내는 price값.

open, high, low, close마다 값차이 존재 확인

```
plt.figure(figsize=(10,6));
plt.plot(df.Date.values, df.Open.values, color='black',label='open')
plt.plot(df.Date.values, df.High.values, color='blue',label='high')
plt.plot(df.Date.values, df.Low.values, color='cyan',label='low')
plt.plot(df.Date.values, df.Close.values, color='red',label='close')

plt.xlabel('Date')
plt.xlim(9520,9525)
plt.ylim(27,30)
plt.ylabel('Price')
plt.legend(loc='best')
```

각 요소 : open, high, low, close



각 요소마다 주가가 다르다는 것을 확인했습니다.

MinMaxScaler화 : 데이터 정규화

```
scaler=MinMaxScaler()
df=df.iloc[:,1:]
scale_df=scaler.fit_transform(df)
print('scale_df값',scale_df)
```

```
scale_df값 [[1.11388821e-03 1.09709183e-03 1.19390541e-03 2.14573915e-04
  1.20684459e-02 1.08619008e-03]
 [1.09204891e-03 1.12967623e-03 1.27059074e-03 2.21290950e-04
  6.18894661e-03 1.11879046e-03]
 [1.11388821e-03 1.15139917e-03 1.18295277e-03 2.01139845e-04
  1.08306566e-02 1.02102268e-03]
 ...
 [8.79061886e-01 8.86713690e-01 8.80378949e-01 9.37417827e-01
  1.36973960e-01 8.82211492e-01]
 [8.91306722e-01 8.91218464e-01 8.88286421e-01 9.45725756e-01
  1.69112966e-01 8.90052952e-01]
 [9.07744889e-01 9.18580674e-01 9.07802634e-01 9.69942417e-01
  2.13922390e-01 9.12909909e-01]]
```

df=df.iloc[:,1:] :

"Date"값 제외 값을 'MinMaxScaler' 하기 위해서 설정.

1)Pandas DataFrame에서 특정 행, 열을 선택하는 방법이 여러 가지 있는데,

1) 행 번호(row number)로 선택하는 방법(.iloc),

2) label이나 조건 표현으로 선택하는 방법(.loc)

```
data.iloc[0:5] # 첫 5개행만
data.iloc[:, 0:2] # 첫 2개열만
data.iloc[[0,3,6,24], [0,5,6]] # 1st, 4th, 7th, 25th 행과 + 1st 6th 7th 열만
data.iloc[0:5, 5:8] # 첫 5개 행과 5th, 6th, 7th 열만
```

```
print('df값',df)
```

df값

	Open	High	Low	Adj Close	Volume	Close
0	0.259115	0.259115	0.253255	0.049994	1574400	0.257161
1	0.257813	0.261068	0.257813	0.050374	844800	0.259115
2	0.259115	0.262370	0.252604	0.049234	1420800	0.253255
3	0.251302	0.251302	0.245768	0.048728	2035200	0.250651
4	0.250651	0.256510	0.248698	0.049614	960000	0.255208
...	...	...	...	...	...	...
15081	53.599998	53.630001	52.439999	52.450001	30485200	52.450001
15082	52.980000	53.520000	52.279999	52.299999	18719600	52.299999
15083	52.599998	53.340000	52.509998	53.070000	17074200	53.070000
15084	53.330002	53.610001	52.980000	53.540001	21062400	53.540001
15085	54.310001	55.250000	54.139999	54.910000	26622900	54.910000

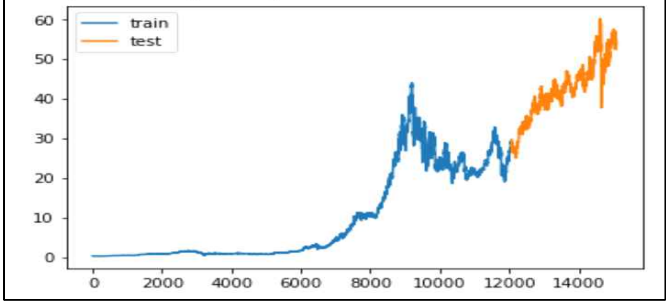
[15086 rows x 6 columns]

- Date가 없는 것을 확인할 수 있습니다.

train값과 test값 구분

```
close1=df.iloc[:,5]
train_close1=pd.DataFrame(close1[0:12069])
test_close1=pd.DataFrame(close1[12069:])
print('test_close.shape값',test_close1.shape)
ax=train_close1.plot()
test_close1.plot(ax=ax)
plt.legend(['train','test'])
```

```
test_close.shape값 (3017, 1)
<matplotlib.legend.Legend at 0x7fa91932eed0>
```



close1은 close값만을 train와 test로 구분하기 위해서 설정하였고, 위의 12069는 전체 15086*0.8을 통해 계산. 그리고 "ax=train_close1.plot()"와 "test_close1.plot(ax=ax)를 통해, 각각 trian과 test로 그래프를 그렸습니다.

```
df.head()#df에는 open, high, low, Adj close, volume, close값 존재.
```

	Open	High	Low	Adj Close	Volume	Close
0	0.259115	0.259115	0.253255	0.049994	1574400	0.257161
1	0.257813	0.261068	0.257813	0.050374	844800	0.259115
2	0.259115	0.262370	0.252604	0.049234	1420800	0.253255
3	0.251302	0.251302	0.245768	0.048728	2035200	0.250651
4	0.250651	0.256510	0.248698	0.049614	960000	0.255208

```
x=scale_df[:,0:5]
print('x값',x[0:4]) #open, high, low, adj close, volume
print('x.shape값',x.shape)
y=scale_df[:,5]
print('y값',y[0:4]) #close값
```

```
x값 [[0.00111389 0.00109709 0.00119391 0.00021457 0.01206845]
 [0.00109205 0.00112968 0.00127059 0.00022129 0.00618895]
 [0.00111389 0.0011514  0.00118295 0.00020114 0.01083066]
 [0.00098284 0.00096674 0.00106794 0.0001922  0.01578181]]
x.shape값 (15086, 5)
y값 [0.00108619 0.00111879 0.00102102 0.00097758]
```

y값은 close값만 출력한 것으로, scale_df[:,5]를 사용.

```
y=y.reshape(-1,1)
print('y.shape값',y.shape)
print('y값 일부출력₩n',y[0:3])
```

```
y.shape값 (15086, 1)
y값 일부출력
 [[0.00108619]
 [0.00111879]
 [0.00102102]]
```

위의 y값으로 구한 MinMaxScaler값과 단독으로 Close값만 뽑아서 MinMaxScaler를 구한 값이 같다는 것을 확인가능.

```
close=df['Close']#close값만 추출
print('close값',close)
train_close1=close[0:12069]
test_close1=close[12069:]

scaler=MinMaxScaler(feature_range=(0,1))
scale_close=scaler.fit_transform(np.array(close).reshape(-1,1))#close값 MinMaxScaler 사용
print('scale_close값',scale_close)
```

- "close=df['Close']"를 통해, close만 MinMaxScaler 값 수행. train_close1, test_close1은 MinMaxScaler 하지 않은 값만을 출력.

MinMaxScaler한 train, test 구분하기 (8:2)

```
train_close=scale_close[0:12069]  #close값 train : 0.8만큼 추출
test_close=scale_close[12069:]

print('test_close값',test_close.shape)
print('train_close값',train_close.shape)
```

```
test_close값 (3017, 1)
train_close값 (12069, 1)
```

```
train_sc_df = pd.DataFrame(train_close, columns=['Close'], index=train_close1.index)
test_sc_df = pd.DataFrame(test_close, columns=['Close'], index=test_close1.index) #train_sc_df

print('train_close1값',train_close1)
print('₩n')
print('test_close1값',test_close1) #minmaxscaler하지 않은 test, train값 구분
```

"pd.DataFrame"을 사용.

1) 출처 : https://azanewta.tistory.com/34

```
train_close1값 0          0.257161
1          0.259115
2          0.253255
3          0.250651
4          0.255208
           ...
12064     28.745001
12065     28.934999
12066     28.840000
12067     28.915001
12068     29.290001
Name: Close, Length: 12069, dtype: float64

test_close1값 12069     29.555000
12070     29.520000
12071     29.530001
12072     29.209999
12073     28.490000
           ...
15081     52.450001
15082     52.299999
15083     53.070000
15084     53.540001
15085     54.910000
Name: Close, Length: 3017, dtype: float64
```

```
train_sc_df값              Close
0          0.001086
1          0.001119
2          0.001021
3          0.000978
4          0.001054
...             ...
12064   0.476375
12065   0.479545
12066   0.477960
12067   0.479211
12068   0.485468

[12069 rows x 1 columns]
test_sc_df값              Close
12069   0.489889
12070   0.489305
12071   0.489472
12072   0.484133
12073   0.472121
...             ...
15081   0.871867
15082   0.869365
15083   0.882211
15084   0.890053
15085   0.912910

[3017 rows x 1 columns]
```

왼쪽 값은 data의 close값, 오른쪽은 scaler한 값을 출력하였습니다.

```
for i in range(1, 22):
    print(i)
    train_DataFrame['Close_{}'.format(i)]=train_DataFrame ['Close'].shift(i)
    test_DataFrame['Close_{}'.format(i)]=test_DataFrame ['Close'].shift(i)
    print(test_DataFrame)
#과거의데이터 21일치를 이용하여, 예측하는 데, 사용.
```

2) Pandas.DataFrame.shift 활용

DataFrame.shift(periods=1, freq=None, axis=0, fill_value=<object object>)[source]

선택적 시간으로 원하는 기간 수만큼 인덱스 이동freq.

2

	Close	Close_1	Close_2
12069	0.489889	NaN	NaN
12070	0.489305	0.489889	NaN
12071	0.489472	0.489305	0.489889
12072	0.484133	0.489472	0.489305
12073	0.472121	0.484133	0.489472
...	...	...	...
15081	0.871867	0.907404	0.893223
15082	0.869365	0.871867	0.907404
15083	0.882211	0.869365	0.871867
15084	0.890053	0.882211	0.869365
15085	0.912910	0.890053	0.882211

[3017 rows x 3 columns]

4

	Close	Close_1	Close_2	Close_3	Close_4
12069	0.489889	NaN	NaN	NaN	NaN
12070	0.489305	0.489889	NaN	NaN	NaN
12071	0.489472	0.489305	0.489889	NaN	NaN
12072	0.484133	0.489472	0.489305	0.489889	NaN
12073	0.472121	0.484133	0.489472	0.489305	0.489889
...	...	...	...	...	...
15081	0.871867	0.907404	0.893223	0.921586	0.929093
15082	0.869365	0.871867	0.907404	0.893223	0.921586
15083	0.882211	0.869365	0.871867	0.907404	0.893223
15084	0.890053	0.882211	0.869365	0.871867	0.907404
15085	0.912910	0.890053	0.882211	0.869365	0.871867

21

	Close	Close_1	Close_2	...	Close_19	Close_20	Close_21
12069	0.489889	NaN	NaN	...	NaN	NaN	NaN
12070	0.489305	0.489889	NaN	...	NaN	NaN	NaN
12071	0.489472	0.489305	0.489889	...	NaN	NaN	NaN
12072	0.484133	0.489472	0.489305	...	NaN	NaN	NaN
12073	0.472121	0.484133	0.489472	...	NaN	NaN	NaN
...	...	...	...	...	...	...	...
15081	0.871867	0.907404	0.893223	...	0.932764	0.933932	0.937268
15082	0.869365	0.871867	0.907404	...	0.935934	0.932764	0.933932
15083	0.882211	0.869365	0.871867	...	0.941106	0.935934	0.932764
15084	0.890053	0.882211	0.869365	...	0.945110	0.941106	0.935934
15085	0.912910	0.890053	0.882211	...	0.936601	0.945110	0.941106

[3017 rows x 22 columns]

for문에 따라 Close_1, 4, 21이 나오는 것을 확인 할 수 있고, "NaN"이 나왔습니다. 이것은 향후 제거할 계획.
이를 통해 데이터 윈도우를 생성하였습니다.

RNN의 아키텍처인 LSTM은 과거의 데이터를 기반으로 미래를 예측하는 모델이기에, 주식에 가장 적합하다고 생각하여, 프로젝트 주제로 결정하였습니다. 따라서 RNN의 경우, 과거의 주가 데이터 몇 개를 기준으로 미래의 주가를 예측할 지를 결정해야합니다.

이때 위의 경우 "range(1,22)"를 통해 21개(3주)를 기준으로 데이터를 입력하도록 설정하였습니다.

+) pandas.DataFrame.shift 예제 코드

```
>>> df = pd.DataFrame({"Col1": [10, 20, 15, 30, 45],
...                    "Col2": [13, 23, 18, 33, 48],
...                    "Col3": [17, 27, 22, 37, 52]},
...                   index=pd.date_range("2020-01-01", "2020-01-05"))
>>> df
            Col1  Col2  Col3
2020-01-01    10    13    17
2020-01-02    20    23    27
2020-01-03    15    18    22
2020-01-04    30    33    37
2020-01-05    45    48    52
```

```
>>> df.shift(periods=3)
            Col1  Col2  Col3
2020-01-01   NaN   NaN   NaN
2020-01-02   NaN   NaN   NaN
2020-01-03   NaN   NaN   NaN
2020-01-04  10.0  13.0  17.0
2020-01-05  20.0  23.0  27.0
```

위의 코드를 보면, periods값이 3이기에, (10, 13, 17)이 3칸 이동한 2020.01.04.로 데이터가 이동했음을 확인할 수 있습니다.

2) 출처 : https://runebook.dev/ko/docs/pandas/reference/api/pandas.dataframe.shift
https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.shift.html

```
x_train=train_sc_df.dropna().drop('Close', axis=1)
y_train=train_sc_df.dropna()[['Close']]

x_test=test_sc_df.dropna().drop('Close', axis=1)
y_test=test_sc_df.dropna()[['Close']]

x_train=x_train.values;  x_test=x_test.values;

y_train=y_train.values;  y_test=y_test.values;
print('x_test.shape값',x_test.shape); print('x_test값',x_test);
print('x_train.shape값',x_train.shape); print('x_train값',x_train);
```

```
x_test.shape값 (2996, 21)
x_test값 [[0.4712865  0.46619789 0.45685491 ... 0.48947198 0.48930512 0.48988906]
 [0.47253776 0.4712865  0.46619789 ... 0.48413309 0.48947198 0.48930512]
 [0.47337199 0.47253776 0.4712865  ... 0.47212068 0.48413309 0.48947198]
 ...
 [0.86936486 0.87186748 0.90740425 ... 0.9411057  0.93593374 0.93276374]
 [0.88221149 0.86936486 0.87186748 ... 0.94510988 0.9411057  0.93593374]
 [0.89005295 0.88221149 0.86936486 ... 0.93660111 0.94510988 0.9411057 ]]
x_train.shape값 (12048, 21)
x_train값 [[0.00084724 0.00068973 0.00070603 ... 0.00102102 0.00111879 0.00108619]
 [0.00087983 0.00084724 0.00068973 ... 0.00097758 0.00102102 0.00111879]
 [0.00092327 0.00087983 0.00084724 ... 0.00105361 0.00097758 0.00102102]
 ...
 [0.47954501 0.4763751  0.47453985 ... 0.4623606  0.45960774 0.45134922]
 [0.47796005 0.47954501 0.4763751  ... 0.46503003 0.4623606  0.45960774]
 [0.47921137 0.47796005 0.47954501 ... 0.46344503 0.46503003 0.4623606 ]]
```

3)위에서 언급했던 21일의 값을 보면 “NaN”이 있는 것을 볼 수 있습니다. 이 값은 제거해야하기 때문에, ”dropna()“함수로 missing value를 제거합니다.

“dropna()” : pandas “NaN” 값 확인 및 처리 : 결측값(NaN)은 데이터분석 시 성능에 영향을 줄 수 있습니다. 따라서 “NaN”을 제거하거나 특정 값으로 채워주는 식으로 처리합니다.

- “NaN” 있는 행, 열 제거 : dropna()

“NaN” 있는 행이나 열을 제거하기 위해서는 “dropna()”를 사용합니다. dropna함수의 axis인자의 값으로 0을 넣어주면 행을 제거해주고, 열은 axis인자에 1을 넣어주면 됩니다.

ex) “NaN” 있는 열 제거 : df.dropna(axis=1), “NaN” 있는 행 제거 : df.dropna() or df.dropna(axis=0)

```
print('train_close값',train_close.shape)
print('x_train.shape값',x_train.shape); print('x_test.shape값',x_test.shape);
#print('x_val.shape값',x_val.shape)
print('y_train.shape값',y_train.shape); print('y_test.shape값',y_test.shape);

train_close값 (12069, 1)
x_train.shape값 (12048, 21)
x_test.shape값 (2996, 21)
y_train.shape값 (12048, 1)
y_test.shape값 (2996, 1)


x_train=np.reshape(x_train, (x_train.shape[0], x_train.shape[1],1))
x_test=np.reshape(x_test, (x_test.shape[0], x_test.shape[1],1))

print('x_train.shape값',x_train.shape); print('x_test.shape값',x_test.shape);
print('y_train.shape값',y_train.shape); print('y_test.shape값',y_test.shape);

x_train.shape값 (12048, 21, 1)
x_test.shape값 (2996, 21, 1)
y_train.shape값 (12048, 1)
y_test.shape값 (2996, 1)
```

Model

```
model=Sequential()
model.add(LSTM(21,return_sequences=True, input_shape=(21,1)))
model.add(LSTM(20,return_sequences=False))
model.add(Dense(1,activation='linear'))
model.compile(loss='mse',optimizer='adam')
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_4 (LSTM)                (None, 21, 21)            1932

lstm_5 (LSTM)                (None, 20)                3360

dense_2 (Dense)              (None, 1)                 21

=================================================================
Total params: 5,313
Trainable params: 5,313
Non-trainable params: 0
_________________________________________________________________
```

4)왼쪽 : “x_train=np.reshape(x_train,(x_train.shape[0], x_train.shape[1],1))”을 보면, shape값이 **(12048, 21, 1)**이 되는데, **“21”**의 값을 통해, 21일 간격으로 데이터를 예측하도록 설정하였다는 것을 확인할 수 있습니다.

그것을 위해서 reshape 함수를 통해, 차원 변경을 하였습니다.

Model은 다음과 같이 구현하였습니다. 이때 day에 따라 “21”로 설정 된 값을 변경하였습니다.

5)Model의 값을 보면 “Param”이 있는데, 이 값은 다음 식을 통해 구해집니다.

$lstm_4 : 1932 = 4*(2*21+21^2)$=(input_dim)*((size of input+1)*(size of output)+$(size\,of\,output)^2$)

이때, input_dim이 4인 이유는 “High, Low, Volume, Adj close”값으로 input을 설정하였기 때문입니다.

---

3) 출처 : https://yganalyst.github.io/data_handling/Pd_6/
https://computer-science-student.tistory.com/306
4) 출처 : https://yganalyst.github.io/data_handling/memo_5/
5) 출처 : https://stackoverflow.com/questions/38080035/how-to-calculate-the-number-of-parameters-of-an-lstm-network
https://medium.com/deep-learning-with-keras/lstm-understanding-the-number-of-parameters-c4e087575756

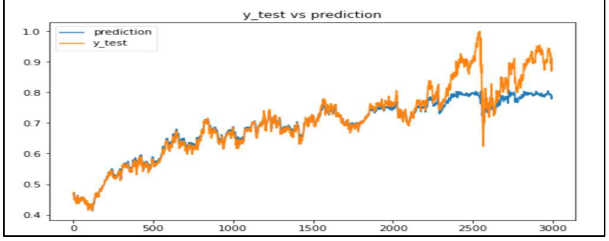
```
model.fit(x_train,y_train, verbose=1,batch_size=10, epochs=30)

1205/1205 [==============================] - 22s 18ms/step - loss: 1.9790e-04
Epoch 3/30
1205/1205 [==============================] - 20s 17ms/step - loss: 1.5930e-04
Epoch 4/30
1205/1205 [==============================] - 20s 17ms/step - loss: 1.3282e-04
Epoch 5/30
1205/1205 [==============================] - 21s 17ms/step - loss: 1.1510e-04
Epoch 6/30
1205/1205 [==============================] - 21s 18ms/step - loss: 1.0588e-04
Epoch 7/30
1205/1205 [==============================] - 21s 17ms/step - loss: 9.4383e-05
Epoch 8/30
1205/1205 [==============================] - 21s 17ms/step - loss: 8.0025e-05
Epoch 9/30
```

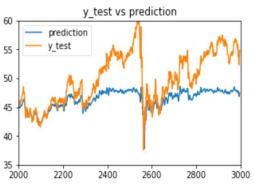
```
Epoch 26/30
1205/1205 [==============================] - 21s 18ms/step - loss: 5.3568e-05
Epoch 27/30
1205/1205 [==============================] - 21s 17ms/step - loss: 5.0701e-05
Epoch 28/30
1205/1205 [==============================] - 21s 17ms/step - loss: 5.2314e-05
Epoch 29/30
1205/1205 [==============================] - 21s 17ms/step - loss: 5.7458e-05
Epoch 30/30
1205/1205 [==============================] - 21s 18ms/step - loss: 5.1243e-05
<keras.callbacks.History at 0x7f823a537310>
```

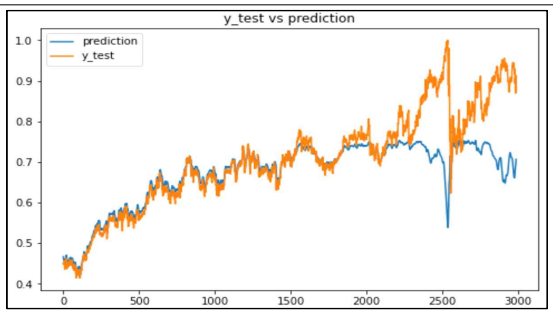
```
plt.figure(figsize=(8,5))
y_pred = model.predict(x_test)
plt.plot(y_pred,label='prediction')
plt.plot(y_test,label='y_test')
plt.title('y_test vs prediction')

plt.legend()
plt.show()   #이를 통해 model값 자체가 x_test로 구한 값과 y_test의 오차
```

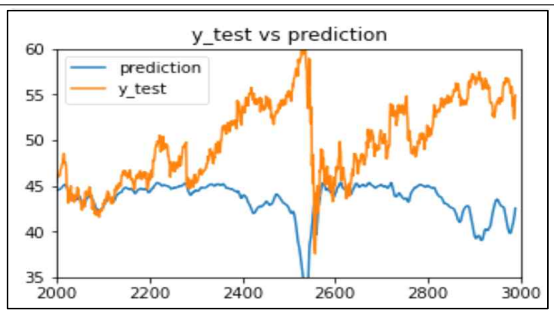



```
plt.figure(figsize=(5,3))
y_pred = model.predict(x_test)
plt.xlim([2000,3000])
plt.ylim([35,60])
y_pred=scaler.inverse_transform(y_pred)
y_test=scaler.inverse_transform(y_test)
plt.plot(y_pred,label='prediction'); plt.plot(y_test,label='y_test'); plt.title('y_test vs prediction');
plt.legend()
plt.show()
```





위의 그래프는 “MinMaxScaler”한 값.

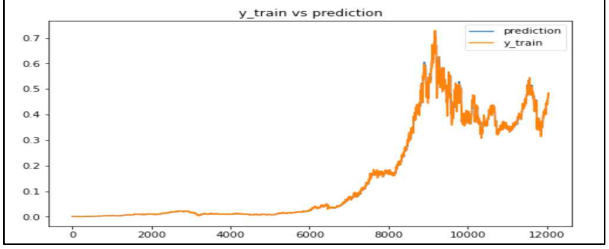


위의 그래프는 실제 주가 그래프 표현.

y_pred=scaler.inverse_transform(y_pred)

따라서, 다음 코드와 같이 “scaler.inverse_transform” 사용

```
plt.figure(figsize=(8,5))
prediction = model.predict(x_train)
plt.plot(prediction,label='prediction')
plt.plot(y_train,label='y_train')
plt.title('y_train vs prediction')
plt.legend()
plt.show() #이를 통해 model값 자체가 x_train로 구한 값과 y_train의 오차
```




```
print('x_train.shape값',x_train.shape)
print('x_test.shape값',x_test.shape)
train_predict=model.predict(x_train)
test_predict=model.predict(x_test)
print('train_predict.shape값',train_predict.shape)
print('test_predict.shape값',test_predict.shape)

x_train.shape값 (12048, 21, 1)
x_test.shape값 (2996, 21, 1)
train_predict.shape값 (12048, 1)
test_predict.shape값 (2996, 1)
```

train값을 출력한 결과 거의 동일하다고 보이는 데, 이를 RMSE로 출력하면, train의 경우, RMSE값이 매우 작은 것을 확인할 수 있습니다.

RMSE값

```
[84] import math #RMSE : y_test,train값과 model에 넣은 값인 predict값의 RMSE값 계산
     from sklearn.metrics import mean_squared_error
     print('test_predict.shape값',test_predict.shape); print('test_RMSE값');
     testRMSE=math.sqrt(mean_squared_error(y_test, test_predict))
     print(testRMSE)
     print('train_predict.shape값',train_predict.shape); print('train_RMSE값');
     math.sqrt(mean_squared_error(y_train, train_predict))
     #train_RMSE값 : 0.006754792341325634, train_predict.shape값 (12048, 1), test_RMSE값 : 42.41810931937551

     test_predict.shape값 (2996, 1)
     test_RMSE값
     42.269629117432764
     train_predict.shape값 (12048, 1)
     train_RMSE값
     0.006135223557482414
```

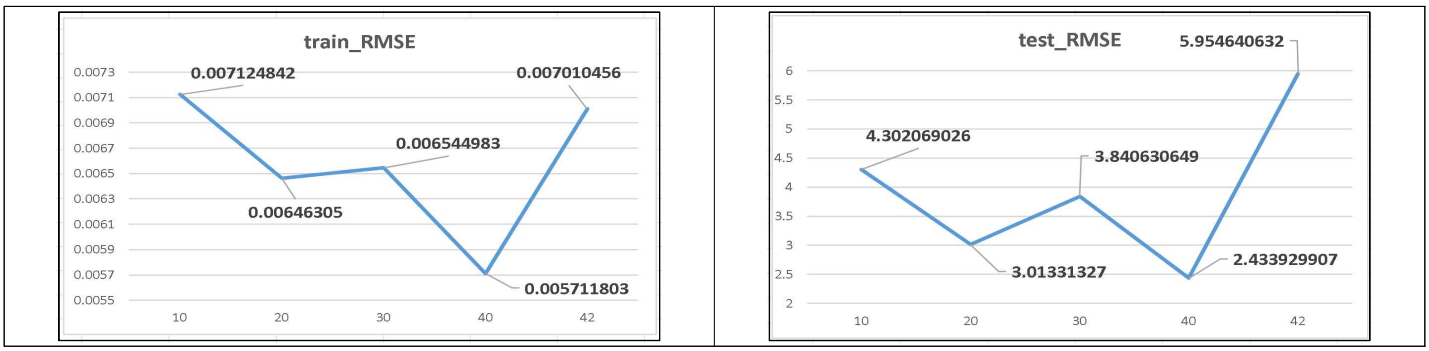
6)회귀 평가지표인 RMSE는 값이 작을수록 회귀 성능이 좋은 것으로 해석됩니다. 향후 test, train값의 RMSE를 통해 성능을 분석할 때 사용합니다.

7) math = math.sqrt(x) : x의 제곱근을 반환합니다.

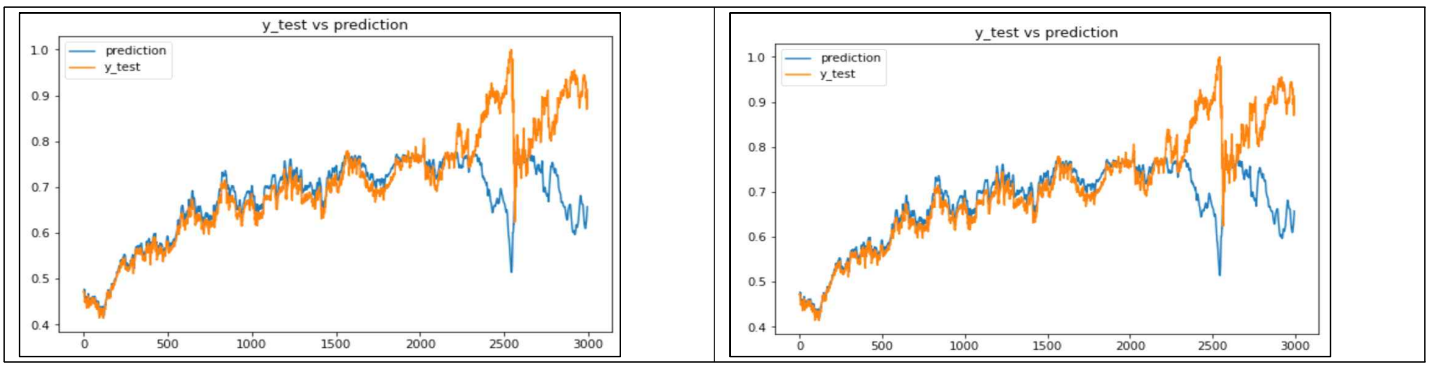
8)mean_squared_error : $$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2.$$

Show your more than two efforts to improve the inference accuracy of the deep learning approach.

1. epoch 값 바꾸기



- 이전에 한 project에서도 epoch를 늘리면, 시간이 오래 걸리지만 성능적인 면에서 향상이 된다는 것을 확인할 수 있었습니다.

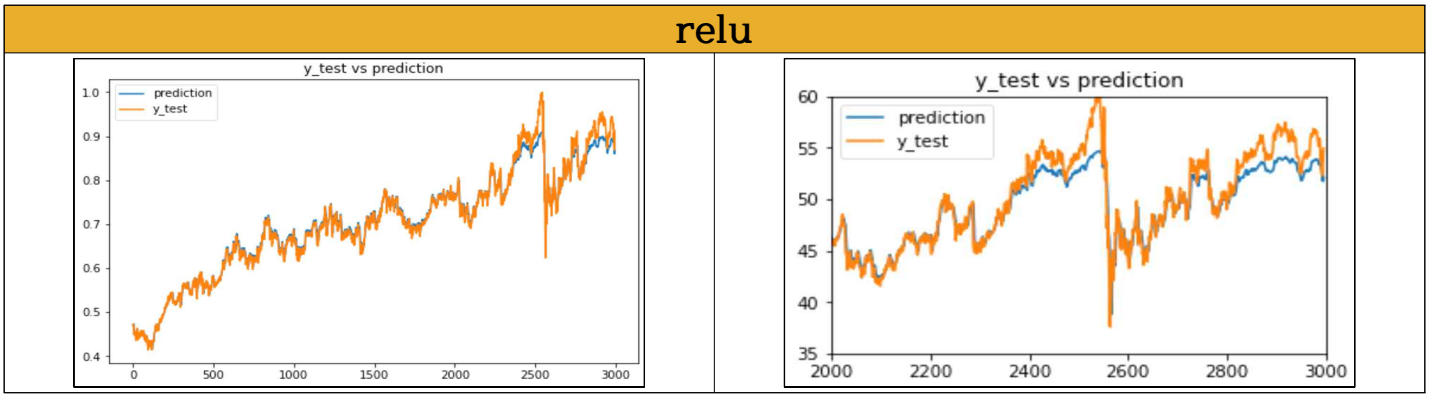


- 특히 Epoch 횟수를 너무 크게 하면 over-fitting이 발생할 수 있는데, epoch값이 42정도로 설정하니 위의 그래프와 표의 값을 통해 over-fitting이 발생한 것을 확인할 수 있었습니다.

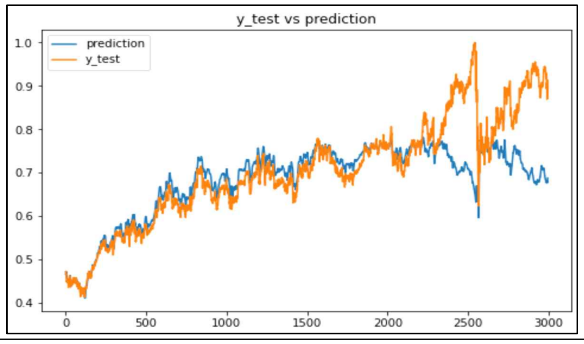
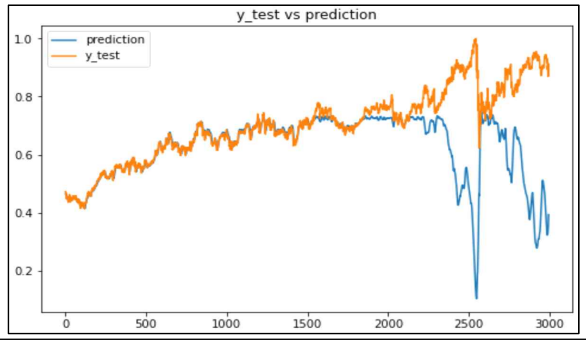
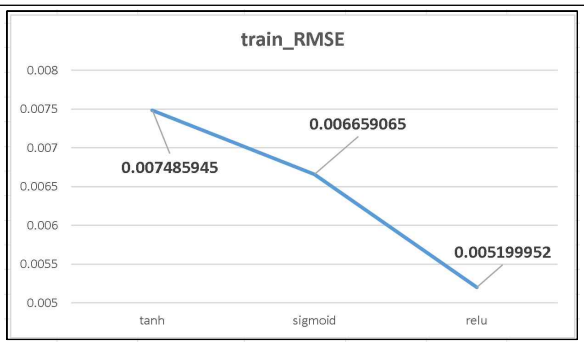
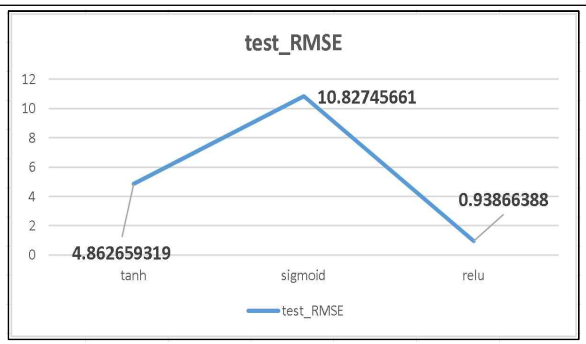
2. activation function 바꾸기

- 여러 활성화 함수를 바꿔서 구현해 봤고, 성능이 좋은 function을 찾았습니다.

relu



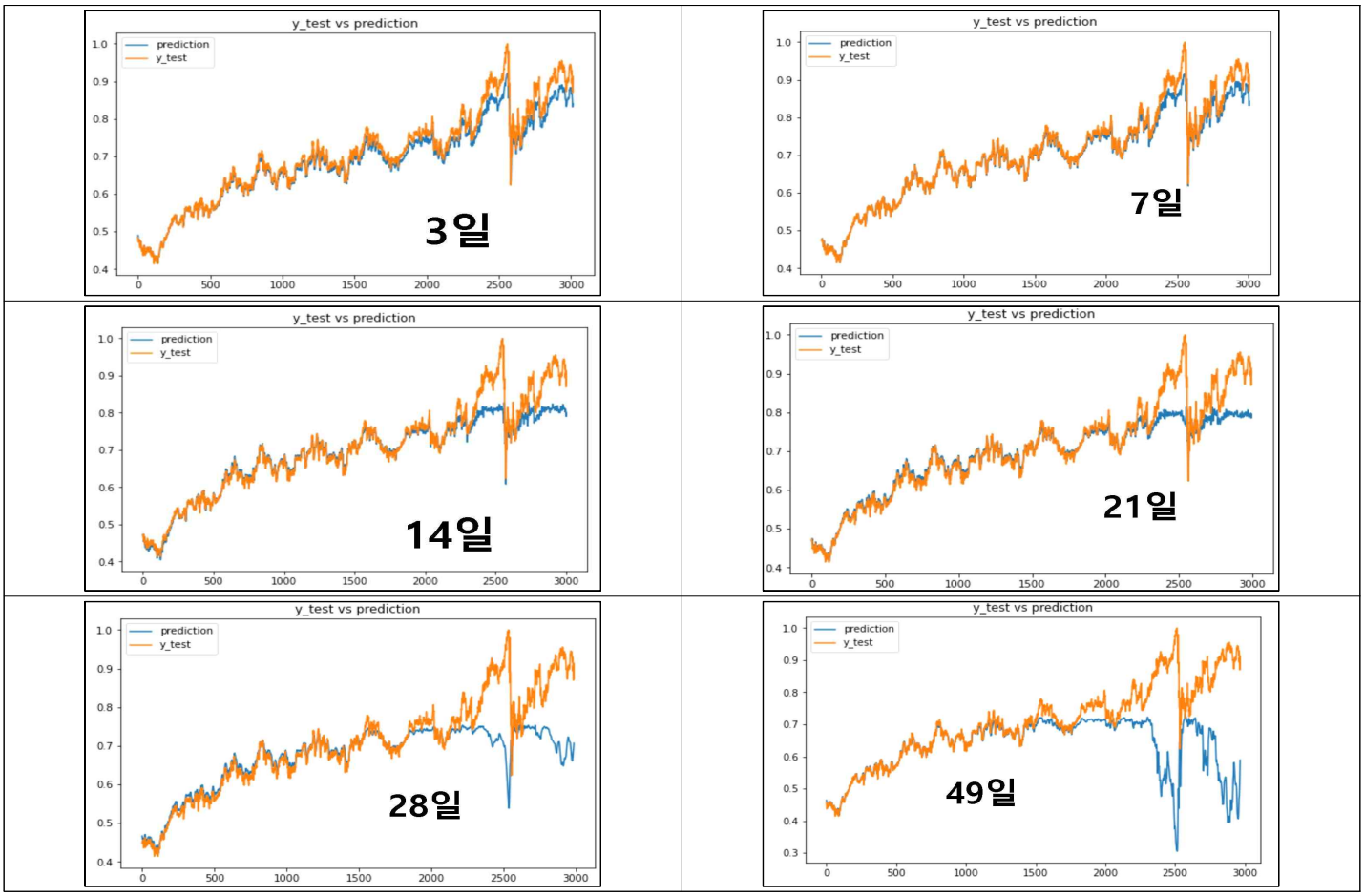
6) 출처 : https://pro-jm.tistory.com/31
7) 출처 : https://docs.python.org/ko/3/library/math.html
8) 출처 : https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html

tanh	sigmoid
	
	

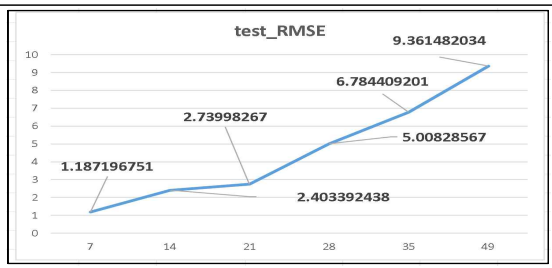
- 그래프를 보면, "ReLu" 함수가 가장 성능이 좋은 활성화함수였습니다.

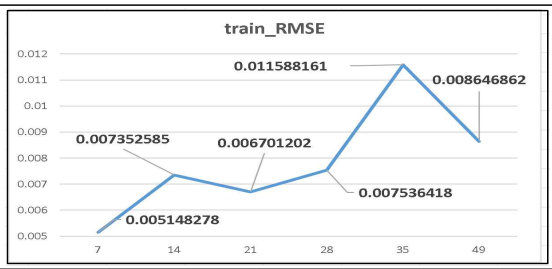
3. day 변경

- 기존 코드는 21일을 기준으로 작성한 것이지만, 3일, 7일, 14일, 21일, 28일, 35일 기준으로 성능비교를 해봤습니다.



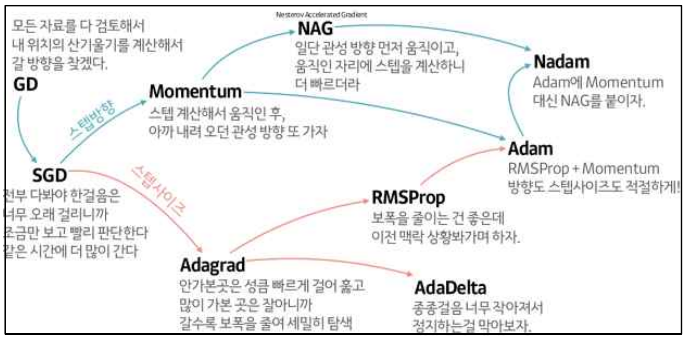
그래프를 통해, 날짜를 길게 잡을수록 RMSE 값이 점점 커지는 것을 확인할 수 있습니다. 시뮬레이션 돌리기 전에는 데이터 값이 많을수록 주가에 대한 데이터가 많이 쌓여 예측하기 좋을 것이라고 생각했는데, 오히려 기간이 길어지면, 데이터의 변동에 의해 예측에 오류가 생긴다는 것을 확인할 수 있습니다.

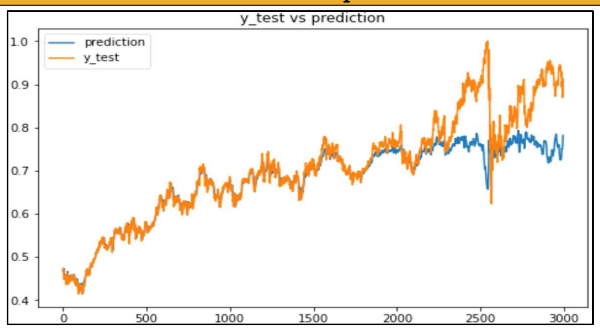
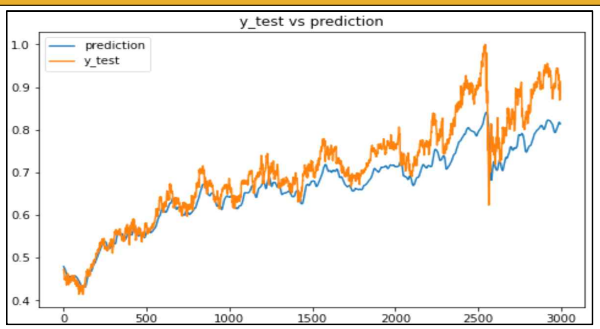
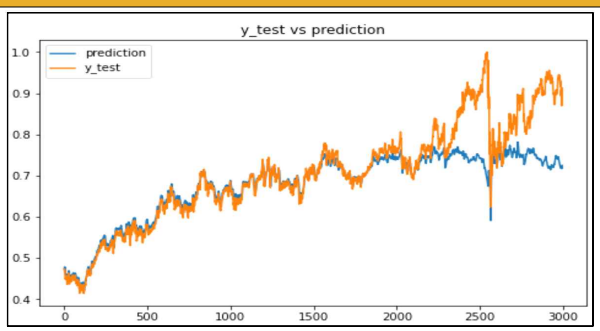
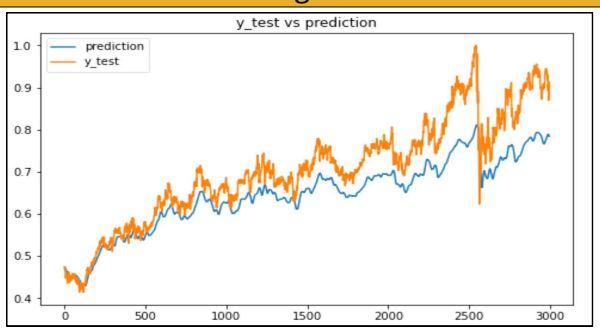




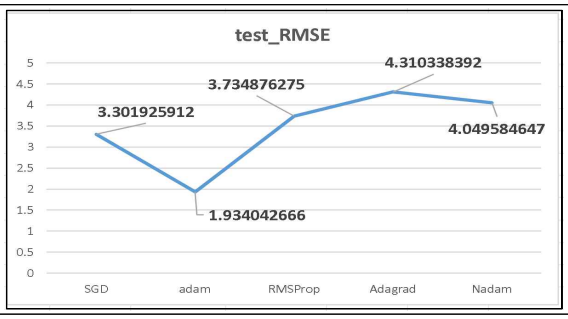
- 위의 그래프를 통해, "test_RMSE"의 값이 "7일" 간격으로 할 때 가장 성능이 좋았습니다.

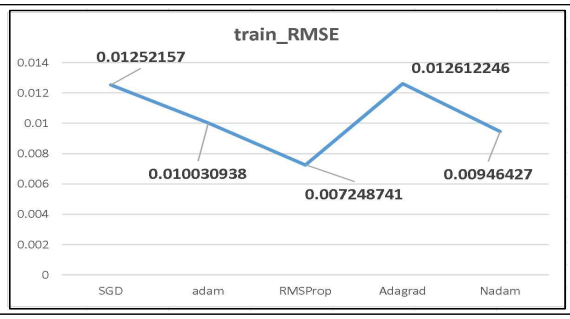
4. [9]optimizer 종류



RMSProp	SGD
	
Nadam	**Adagrad**
	

- 위의 그래프와 그래프를 통해, train에서는 RMSE의 값이 큰 차이는 없지만, "test_RMSE"의 경우, "adam"이 가장 좋은 성능을 보여주었습니다.





9) 출처 : https://ganghee-lee.tistory.com/24

[Rules for determining your dataset]

Date	Open	High	Low	Adj Close	Volume	Close
1962-01-03	0.259115	0.259115	0.253255	0.049994	1574400	0.257161
1962-01-04	0.257813	0.261068	0.257813	0.050374	844800	0.259115
1962-01-05	0.259115	0.26237	0.252604	0.049234	1420800	0.253255
1962-01-08	0.251302	0.251302	0.245768	0.048728	2035200	0.250651
1962-01-09	0.250651	0.25651	0.248698	0.049614	960000	0.255208
1962-01-10	0.255208	0.260091	0.252604	0.049867	1612800	0.25651
1962-01-11	0.25651	0.259115	0.255208	0.050374	614400	0.259115
1962-01-12	0.259115	0.259115	0.254557	0.049994	883200	0.257161
1962-01-15	0.25651	0.25651	0.253906	0.049488	614400	0.254557
1962-01-16	0.253906	0.253906	0.249674	0.048602	1113600	0.25
1962-01-17	0.25	0.25	0.242188	0.047083	1574400	0.242188
1962-01-18	0.242188	0.24349	0.236328	0.046134	2649600	0.237305
1962-01-19	0.239583	0.244792	0.239583	0.047589	1459200	0.244792
1962-01-22	0.244792	0.247396	0.242188	0.047969	1075200	0.246745
1962-01-23	0.246094	0.246094	0.241536	0.046956	768000	0.241536

Make a nontrivial input-output relationship with time dependency

1) the dimensions of the input should be **larger than 5**.

"coca.csv"파일안에 **"Date", "Open", "High", "Low", "Adj Close", "Volume", "Close"**값이 있기에, 성립하였습니다.

15081	2021-11-26	54.59	54.75	53.58	53.31654	14754300	53.73
15082	2021-11-29	54.05	54.73	53.93	54.16	22712500	54.58
15083	2021-11-30	53.6	53.63	52.44	52.45	30485200	52.45
15084	2021-12-01	52.98	53.52	52.28	52.3	18719600	52.3
15085	2021-12-02	52.6	53.34	52.51	53.07	17074200	53.07
15086	2021-12-03	53.33	53.61	52.98	53.54	21062400	53.54
15087	2021-12-06	54.31	55.25	54.14	54.91	26622900	54.91

2) Generate training **(more than 10,000 samples)** and testing datasets **(more than 2,000 samples)** based on your function.

- 데이터 총 수가 15087개이기에, 다음 조건을 성립.

3) Input and output can be limited/unlimited, discrete/continuous, whatever.

4) Your output should also be time dependent

- 다음 조건은 시계열 데이터이기에, 성립합니다.

[What you need to do]

1) Write a code for regression to approximate your function by designing custom RNNs. Whatever RNN architecture is ok.

- LSTM으로 구현하였기에, 성립

2) Show your more than two efforts to improve the inference accuracy of the deep learning approach.

- controlling your learning rate - modifying your neural network architecture.

<결론>

- 딥러닝 수업을 처음으로 들었을 때, 모의투자대회에서 입상했을 정도로 관심 있던 주식분야를 분석해보고 싶었는데, LSTM이라는 것을 알게 되어 주제로 한 번 다뤄봐서 재미있는 프로젝트였습니다.

특히 RMSE로 성능 평가했을 때, 약간의 오차가 발생하는 것을 볼 수 있지만, 이 오차가 아무리 작더라도 실제로 투자로 옮길 경우, 손해가 발생할 가능성이 존재하기에 실제 투자로 옮기기는 어렵다는 것을 결론을 내릴 수 있습니다.

그래서 "Twitter mood predicts the stock market- J.Bollen", "Stock Prediction Using Twitter Sentiment Analysis - anshul mittal" 등의 논문에서는 twitter의 언급되는 메시지에 따라 호재인가 악재인지를 파라미터로 추가 설정하여 주가를 예측하였습니다. 즉, 기존의 재무제표와 주가 데이터만으로는 주가를 예측하기 어렵기에 여러 파라미터를 추가하여, 주가의 변동성을 예측하고 있습니다.

만일 가능하다면 기존의 증권사가 운영하고 있는 로봇 어드바이저는 어떤 파라미터 요소로 주가를 예측하는 것인지와 100% AI가 예측하는 자동매매 프로그램인지, 코로나나 블랙스완와 같은 이슈를 대처할 수 있는 현 자율주행차 기술처럼 반자동기술인지에 대해 알아보고 싶습니다.