LSTM - Time Serise Dataset

- 서울시 기후 데이터 : 2011년 01월 01일 ~ 2019년 12월 31일
- https://data.kma.go.kr/cmmn/main.do
- 기후통계분석 -> 기온분석 -> 기간(20110101~20191231) -> 검색 -> CSV 다운로드
- Seoul_Temp.csv

```
import warnings
warnings.filterwarnings('ignore')
```

Import Packages

- Packages

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt

from sklearn.preprocessing import MinMaxScaler

from keras.models import Sequential
from keras.layers import Dense, LSTM
```

I. Colab File Upload

1) 'Seoul_temp.csv' 파일을 Colab에 업로드 후 진행

```
url = 'https://raw.githubusercontent.com/rusita-ai/pyData/master/Seoul_Temp.csv'
temp = pd.read_csv(url)

temp.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3287 entries, 0 to 3286
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   date    3287 non-null   object
 1   avg     3287 non-null   float64
 2   min     3287 non-null   float64
 3   max     3287 non-null   float64
dtypes: float64(3), object(1)
memory usage: 102.8+ KB
```

```
temp.head()
```

	date	avg	min	max
0	2011-01-01	-6.8	-10.4	-2.9
1	2011-01-02	-5.4	-8.5	-1.2
2	2011-01-03	-4.5	-8.5	-0.3
3	2011-01-04	-3.9	-7.4	-1.7
4	2011-01-05	-4.0	-7.7	-1.8

II. Data Preprocessing

1) 일일 평균온도('avg') 변화 시각화

- 일일 평균온도 변화에 일정한 패턴 확인

```
temp_data = temp[['avg']]

plt.figure(figsize = (12, 5))
plt.plot(temp_data)
plt.show()
```

2) Normalization

- tanh Activation 적용을 위해 -1 ~ 1 범위로 정규화

```
scaler = MinMaxScaler(feature_range = (-1, 1))

temp_data = scaler.fit_transform(temp_data)
```

3) Train vs. Test Split

- Train_Dataset : 2011년 01월 01일 ~ 2017년 12월 31일
- Test_Dataset : 2018년 01월 01일 ~ 2019년 12월 31일

```
train = temp_data[0:2557]
test  = temp_data[2557:]
```

III. 시계열 데이터 처리 함수

1) 시계열 학습용 데이터 생성 함수 정의

- X : 학습 평균온도 데이터
- y : 정답 평균온도 데이터
- 일정 기간의 X로 y를 예측하도록 학습

```
def create_dataset(time_data, look_back = 1):
    data_X, data_y = [], []

    for i in range(len(time_data) - look_back):
        data_X.append(time_data[i:(i + look_back), 0])
        data_y.append(time_data[i + look_back, 0])

    return np.array(data_X), np.array(data_y)
```

2) loop_back 기간 설정 후 학습데이터 생성

- 180일 기간 평균온도로 다음날 평균온도 예측 데이터 생성

```
look_back = 180

train_X, train_y = create_dataset(train, look_back)
test_X,  test_y  = create_dataset(test, look_back)

train_X.shape, train_y.shape, test_X.shape, test_y.shape
```

```
((2377, 180), (2377,), (550, 180), (550,))
```

3) Tensor Reshape

```
train_X = np.reshape(train_X, (train_X.shape[0], train_X.shape[1], 1))
test_X = np.reshape(test_X, (test_X.shape[0], test_X.shape[1], 1))

train_X.shape, train_y.shape, test_X.shape, test_y.shape
```

```
((2377, 180, 1), (2377,), (550, 180, 1), (550,))
```

IV. LSTM Modeling

1) Model Define

```
model = Sequential()
```

```
model = Sequential()
model.add(LSTM(64,
               input_shape = (None, 1)))
model.add(Dense(1, activation = 'tanh'))
```

- Model Summary

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 64)                16896
_________________________________________________________________
dense (Dense)                (None, 1)                 65
=================================================================
Total params: 16,961
Trainable params: 16,961
Non-trainable params: 0
_________________________________________________________________
```

2) Model Compile

```
model.compile(loss = 'mean_squared_error',
              optimizer = 'adam')
```

3) Model Fit

- 약 5분

```
%%time

hist = model.fit(train_X, train_y,
                 epochs = 200,
                 batch_size = 16,
                 validation_data = (test_X, test_y))
```

```
149/149 [==============================] - 1s 7ms/step - loss: 0.0052 - val_loss: 0.0095
Epoch 173/200
149/149 [==============================] - 1s 8ms/step - loss: 0.0058 - val_loss: 0.0096
Epoch 174/200
149/149 [==============================] - 1s 8ms/step - loss: 0.0058 - val_loss: 0.0094
Epoch 175/200
149/149 [==============================] - 1s 8ms/step - loss: 0.0051 - val_loss: 0.0087
Epoch 176/200
149/149 [==============================] - 1s 8ms/step - loss: 0.0052 - val_loss: 0.0096
Epoch 177/200
149/149 [==============================] - 1s 8ms/step - loss: 0.0056 - val_loss: 0.0088
Epoch 178/200
149/149 [==============================] - 1s 8ms/step - loss: 0.0056 - val_loss: 0.0092
Epoch 179/200
```

```
149/149 [==============================] - 1s 8ms/step - loss: 0.0052 - val_loss: 0.0090
Epoch 180/200
149/149 [==============================] - 1s 8ms/step - loss: 0.0052 - val_loss: 0.0091
Epoch 181/200
149/149 [==============================] - 1s 8ms/step - loss: 0.0053 - val_loss: 0.0097
Epoch 182/200
149/149 [==============================] - 1s 8ms/step - loss: 0.0056 - val_loss: 0.0091
Epoch 183/200
149/149 [==============================] - 1s 8ms/step - loss: 0.0051 - val_loss: 0.0090
Epoch 184/200
149/149 [==============================] - 1s 7ms/step - loss: 0.0052 - val_loss: 0.0093
Epoch 185/200
149/149 [==============================] - 1s 8ms/step - loss: 0.0056 - val_loss: 0.0089
Epoch 186/200
149/149 [==============================] - 1s 8ms/step - loss: 0.0053 - val_loss: 0.0101
Epoch 187/200
149/149 [==============================] - 1s 8ms/step - loss: 0.0049 - val_loss: 0.0096
Epoch 188/200
149/149 [==============================] - 1s 8ms/step - loss: 0.0052 - val_loss: 0.0091
Epoch 189/200
149/149 [==============================] - 1s 8ms/step - loss: 0.0051 - val_loss: 0.0092
Epoch 190/200

149/149 [==============================] - 1s 8ms/step - loss: 0.0049 - val_loss: 0.0096
Epoch 191/200
149/149 [==============================] - 1s 8ms/step - loss: 0.0051 - val_loss: 0.0098
Epoch 192/200
149/149 [==============================] - 1s 8ms/step - loss: 0.0048 - val_loss: 0.0099
Epoch 193/200
149/149 [==============================] - 1s 8ms/step - loss: 0.0050 - val_loss: 0.0094
Epoch 194/200
149/149 [==============================] - 1s 8ms/step - loss: 0.0047 - val_loss: 0.0098
Epoch 195/200
149/149 [==============================] - 1s 8ms/step - loss: 0.0051 - val_loss: 0.0114
Epoch 196/200
149/149 [==============================] - 1s 8ms/step - loss: 0.0051 - val_loss: 0.0099
Epoch 197/200
149/149 [==============================] - 1s 8ms/step - loss: 0.0047 - val_loss: 0.0098
Epoch 198/200
149/149 [==============================] - 1s 8ms/step - loss: 0.0047 - val_loss: 0.0106
Epoch 199/200
149/149 [==============================] - 1s 8ms/step - loss: 0.0051 - val_loss: 0.0102
Epoch 200/200
149/149 [==============================] - 1s 8ms/step - loss: 0.0046 - val_loss: 0.0096
CPU times: user 4min 14s, sys: 15.3 s, total: 4min 30s
Wall time: 4min 22s
```

4) 학습결과 시각화

```
plt.figure(figsize = (12, 5))
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])

plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['Train Loss', 'Valid Loss'], loc = 'upper right')
plt.show()
```

5) Model Evaluate

```
trainScore = model.evaluate(train_X, train_y, verbose = 0)
print('Train Score: ', trainScore)

testScore = model.evaluate(test_X, test_y, verbose = 0)
print('Test Score: ', testScore)
```

```
Train Score:  0.004603151232004166
Test Score:  0.009577494114637375
```

V. Model Predict

```
look_ahead = 550

xhat = test_X[0]

predictions = np.zeros((look_ahead, 1))

for i in range(look_ahead):
    prediction = model.predict(np.array([xhat]), batch_size = 1)
    predictions[i] = prediction
    xhat = np.vstack([xhat[1:], prediction])

plt.figure(figsize = (12, 5))
plt.plot(np.arange(look_ahead), predictions, 'r', label = 'Prediction')
plt.plot(np.arange(look_ahead), test_y[:look_ahead], label = 'Test_Data')
plt.legend()
plt.show()
```

#

#

#

The End

#

#

#

✓ 0초 오전 8:33에 완료됨