날씨 데이터를 활용한 태양광 발전량 예측

8강. 모델의 학습과 결과 분석

1. 모델 학습을 위한 데이터 준비하기

2. Keras를 활용한 시계열 딥러닝

학습용 Feature와 Label 분리하기

merge_df에서 DSR, DSR_DQF, 강수량, 풍속, 풍향, 기온, 습도는 X로 정의하고,

총 발전량은 y로 정의합니다.

```
# merge_df데이터프레임에서 총발전량만을 제외한 컬럼으로 'X'라는 이름으로 데이터프레임 생성
X = merge_df.drop(columns = ['제주태양광 총발전량'])
y = merge_df['제주태양광 총발전량']
```

전체 데이터의 갯수는 7186개 입니다.

```
X.shape, y.shape
```

```
((7186,), (7186,))
```

훈련 데이터와 테스트 데이터를 나누는 기준을 6500개로 설정합니다.

```
TRAIN_SPLIT = 6500
```

데이터 정규화 하기

X와 y데이터를 정규화 합니다.

이때 scikit-learn의 MinMaxScaler를 활용하면 쉽게 정규화 할 수 있습니다.

```
from sklearn.preprocessing import MinMaxScaler, StandardScaler

scaler = MinMaxScaler(feature_range = (0,1))
#scaler = StandardScaler()

def normalize(X,y):
    X_norm = X.copy()

    for name in X:
        temp = X[name].to_numpy().reshape(-1,1)
        #print(temp)
        X_norm[name] = scaler.fit_transform(temp)


    temp = y.to_numpy().reshape(-1,1)
    y_norm = scaler.fit_transform(temp)

    return X_norm, y_norm

X_norm, ydata = normalize(X,y)
Xdata = X_norm.to_numpy()
```

0~1 사이의 값으로 scaling

기상청 Korea Meteorological Administration

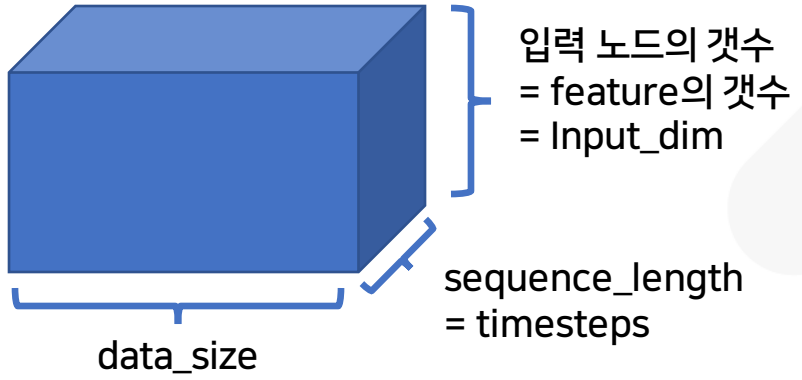
데이터셋 나누고 변환하기

multivariate_data함수는 시계열 데이터 타입으로 변환해 주는 함수입니다.

```
sequence_length = 24
future_target = 1
# 다음 한시간을 예측
STEP = 1

X_train, y_train = multivariate_data(Xdata, ydata, 0, TRAIN_SPLIT, sequence_length,
                                                    future_target, STEP, single_step=True)
X_test, y_test = multivariate_data(Xdata, ydata, TRAIN_SPLIT, None, sequence_length,
                                                  future_target, STEP, single_step=True)
X_train.shape, X_test.shape
```

```
((6476, 24, 270), (661, 24, 270))
```

(Data_size, timesteps, input_dim)



데이터 입력 파이프라인

X_train과 y_train데이터를 메모리(cache)로 가져와서 batch 사이즈로 잘라서 train_data로 전달

X_test과 y_test데이터를 batch 사이즈로 잘라서 test_data로 전달

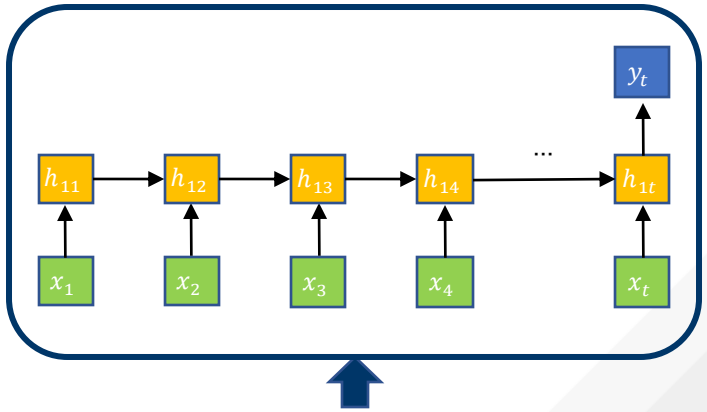
```
import tensorflow as tf

BATCH_SIZE = 28
BUFFER_SIZE = 1000

train_data = tf.data.Dataset.from_tensor_slices((X_train, y_train)).cache().batch(BATCH_SIZE)
test_data = tf.data.Dataset.from_tensor_slices((X_test, y_test)).batch(BATCH_SIZE)
```

LSTM 모델 설정 (은닉층 1개)

Keras에서 함수형 모델로 LSTM 모델을 정의합니다.

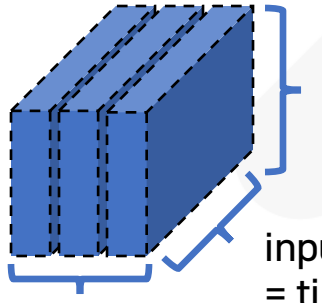
```
from tensorflow.keras import Input, Model
from tensorflow.keras.layers import Dense, SimpleRNN, Bidirectional, Dropout

inputs = Input(shape=(X_train.shape[-2:]))
x = LSTM(h_units, recurrent_dropout=0.5, kernel_initializer='he_normal')(inputs)
#x = LSTM(h_units, return_sequences=False, kernel_initializer='he_normal')(x)
x = Dropout(0.5)(x)

outputs = Dense(1)(x)

model = Model(inputs, outputs)

model.compile(loss='mse', metrics=['mae'], optimizer=Adam(learning_rate = 0.0006))
```

```
WARNING:tensorflow:Layer gru will not use cuDNN kernels since it doesn't meet the criteria. It
will use a generic GPU kernel as fallback when running on GPU.
```



Inputs

(batch_size, timestep, input_dim)



입력 노드의 갯수
= Input_dim

input_length
= timesteps

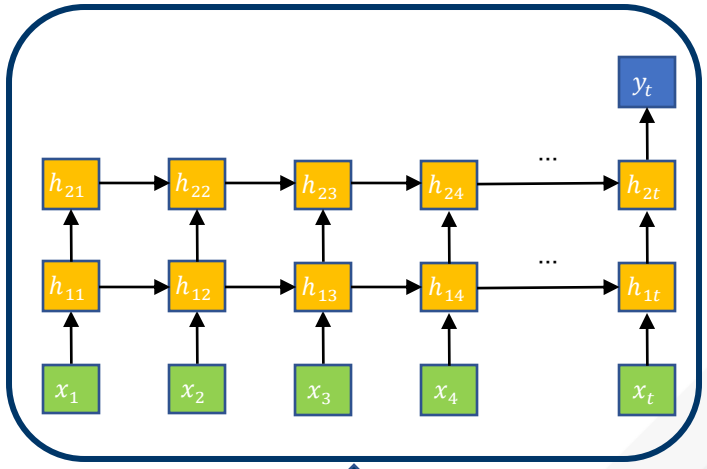
샘플의 갯수
= batch_size

LSTM 모델 설정 (은닉층 2개)

Keras에서 함수형 모델로 LSTM 모델을 정의합니다.

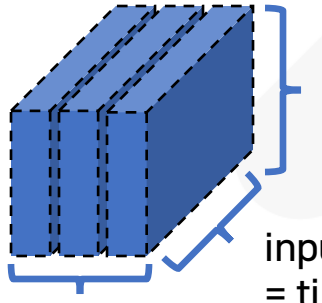
```
from tensorflow.keras import Input, Model
from tensorflow.keras.layers import Dense, SimpleRNN, Bidirectional, Dropout

inputs = Input(shape=(X_train.shape[-2:]))
x = LSTM(h_units, recurrent_dropout=0.5, kernel_initializer='he_normal')(inputs)
x = LSTM(h_units, return_sequences=False, kernel_initializer='he_normal')(x)
x = Dropout(0.5)(x)

outputs = Dense(1)(x)

model = Model(inputs, outputs)

model.compile(loss='mse', metrics=['mae'], optimizer=Adam(learning_rate = 0.0006))
```

```
WARNING:tensorflow:Layer gru will not use cuDNN kernels since it doesn't meet the criteria. It
will use a generic GPU kernel as fallback when running on GPU.
```



Inputs

(batch_size, timestep, input_dim)



입력 노드의 갯수
= Input_dim

input_length
= timesteps

샘플의 갯수
= batch_size

Keras Callback함수 설정

callback함수로 학습 진행중 learning rate 변경 조건, 학습 중단 조건, 모델의 저장 조건을 설정합니다.

```
from tensorflow.keras.callbacks import EarlyStopping,ReduceLROnPlateau, ModelCheckpoint


# 'val_mae'를 기준으로 7epoch동안 val_mae 값이 줄어들지 않을 경우 기존의 learning_rate를 반으로 줄인다.
reduce_lr = ReduceLROnPlateau(monitor='val_mae', factor=0.5,patience=7)

#'val_mae'가 20EPOCH동안 내려가지 않으면 학습을 중단한다.
es=EarlyStopping(monitor='val_mae', patience=20)

# 학습중에 'val_mae'가 가장 낮을 때 'best.h5'에 weight값을 저장한다.
mc = ModelCheckpoint('best.h5', monitor='val_mae', mode='auto', verbose = 1, save_best_only=True)
callback = [reduce_lr, es, mc]

history = model.fit(train_data, epochs=150, batch_size=BATCH_SIZE, validation_data=test_data, callbacks=[callback])
```

하이퍼 파라메터

- h_size = 100
- drop_ratio = 0.5
- lr = 0.0005
- recurrent_dropout=0.5
- 시계열 신경망 layer의 갯수
- sequence_length = 30
- future_target = 1

h_size를 변경해 가면서 mae와 mse가 낮은 조건 테스트

Dropout 비율을 변경해서 테스트

학습율을 조정해서 테스트

출력값을 다시 은닉층으로 입력할때 dropout의 비율 테스트

레이어의 갯수를 1개~3개까지 테스트

시계열 데이터의 길이를 변경해가면서 테스트

예측하는 시간의 길이

테스트 결과 – 시계열 모델간 비교

RNN과 LSTM, GRU 3개의 모델을 비교

Model	hidden_unit	Val_mae	Val_mse
SimpleRNN	100	0.06639	0.01312
LSTM	100	0.042459	0.00626
GRU	100	0.053627	0.00817

테스트 결과 – 은닉층 노드의 갯수 비교

LSTM모델로 은닉층 노드의 갯수를 변경했을 때

Model	hidden_unit	Val_mae	Val_mse
LSTM	95	0.04485	0.006634
LSTM	100	0.04245	0.006267
LSTM	105	0.04632	0.007820

테스트 결과 – 입력 Feature의 차이 비교

LSTM모델로 은닉층 노드의 갯수를 변경했을 때

Model	h_unit	Val_mae	Val_mse	Features
LSTM	100	0.0502	0.008843	DSR
LSTM	100	0.04471	0.00694	DSR,DSR_DQF
LSTM	100	0.04783	0.00673	DSR, DSR_DQF, 습도
LSTM	100	0.04658	0.00633	DSR,DSR_DQF,습도,풍속
LSTM	100	0.04645	0.00669	DSR,DSR_DQF,풍속
LSTM	100	0.04245	0.00626	DSR,DSR_DQF,습도,풍속,기온,강수량

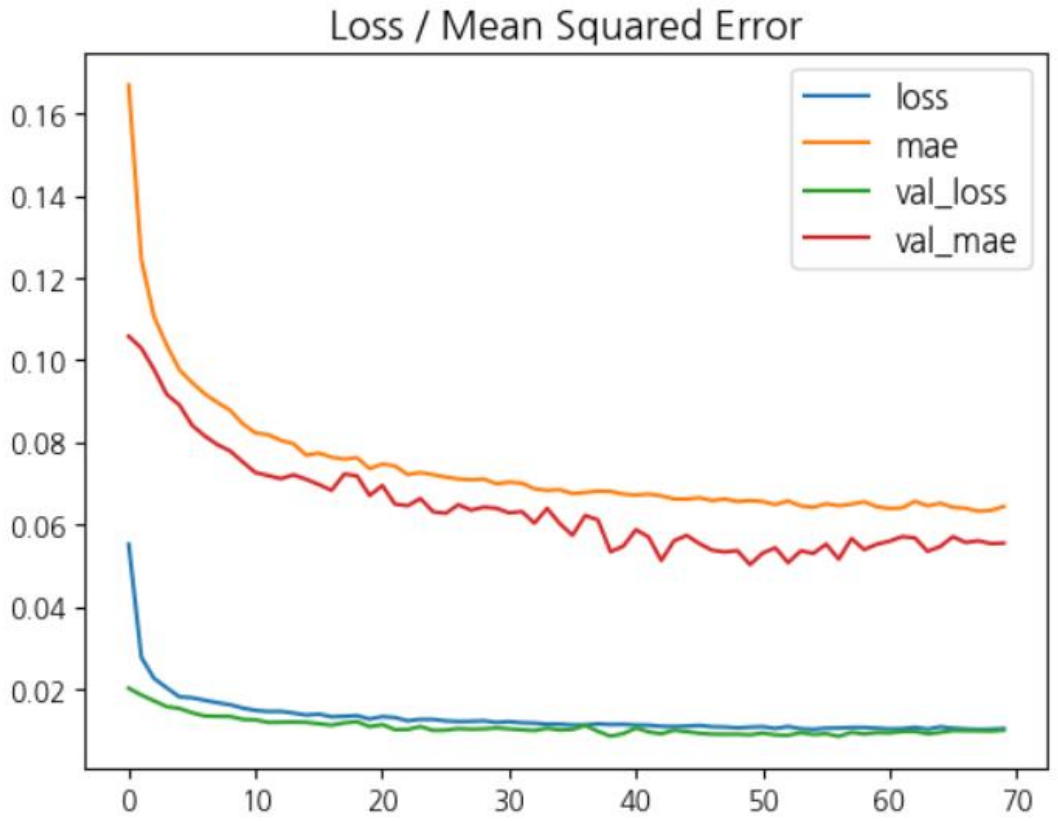
테스트 결과 – 입력 Feature의 차이 비교

LSTM모델로 은닉층 노드의 갯수를 변경했을 때

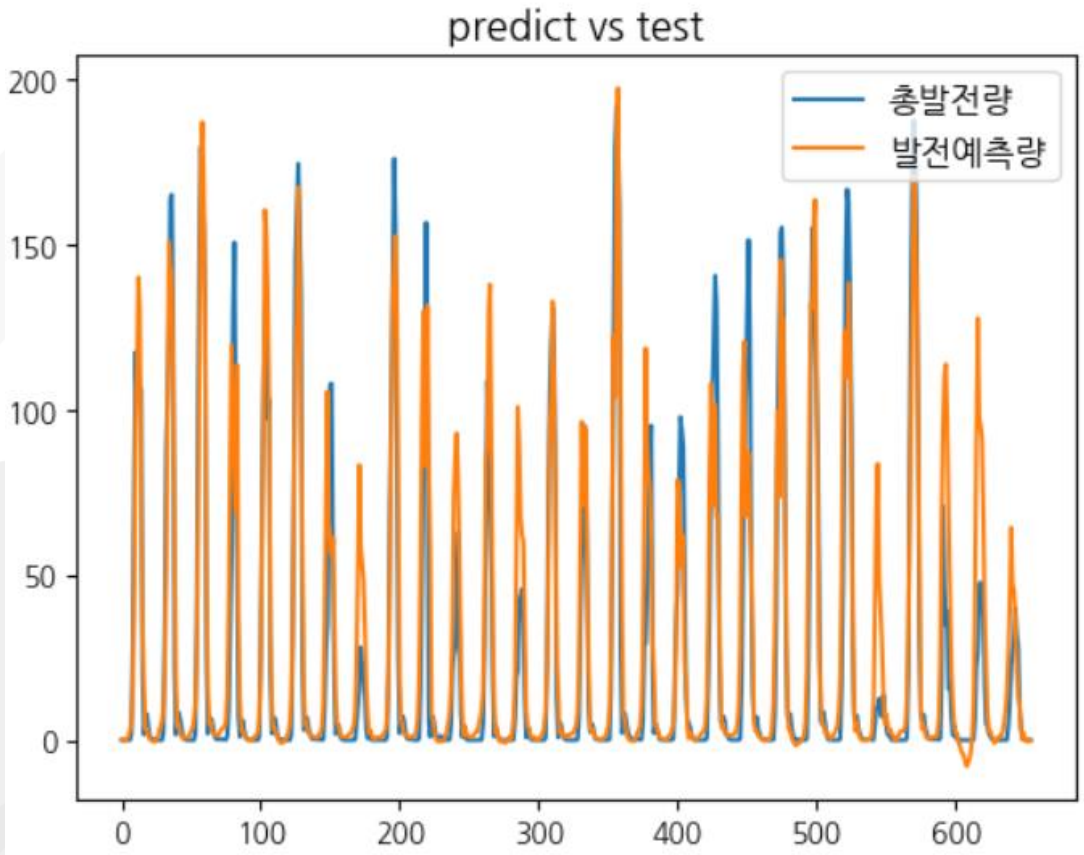
Model	h_unit	Val_mae	Val_mse	Features
LSTM	100	0.0502	0.008843	DSR
LSTM	100	0.04471	0.00694	DSR,DSR_DQF
LSTM	100	0.04783	0.00673	DSR, DSR_DQF, 습도
LSTM	100	0.04658	0.00633	DSR,DSR_DQF,습도,풍속
LSTM	100	0.04645	0.00669	DSR,DSR_DQF,풍속
LSTM	100	0.04245	0.00626	DSR,DSR_DQF,습도,풍속,기온,강수량

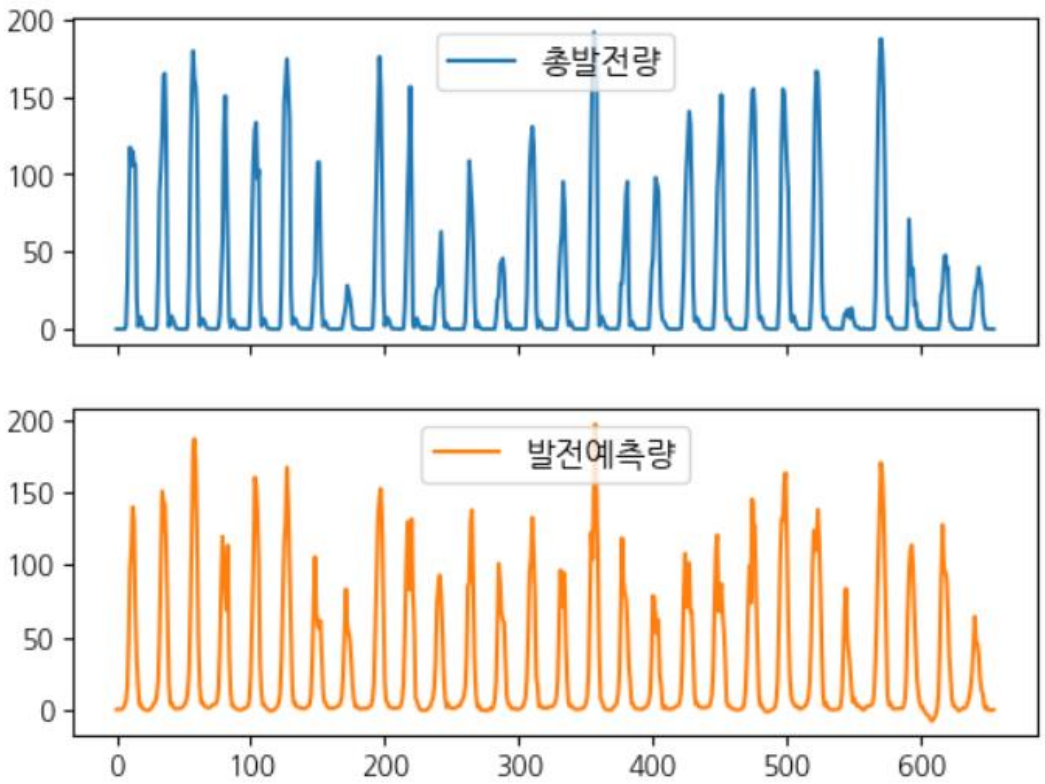
테스트 결과

```
import matplotlib.pyplot as pyplot


model.load_weights('best.h5')

pyplot.title('Loss / Mean Squared Error')
pyplot.plot(history.history['loss'], label='loss')
pyplot.plot(history.history['mae'], label='mae')
pyplot.plot(history.history['val_loss'], label='val_loss')
pyplot.plot(history.history['val_mae'], label='val_mae')
pyplot.legend()
pyplot.show()

test_loss = model.evaluate(X_test, y_test)

y_pred = model.predict(X_test)
```



테스트 결과


predict vs test
총발전량
발전예측량
200
150
100
50
0
0
100
200
300
400
500
600


총발전량
200
150
100
50
0
발전예측량
200
150
100
50
0
0
100
200
300
400
500
600

추가적으로 검토해 볼 내용

- 태양광 발전소의 발전 설비 용량 데이터를 추가해서 학습해보기 (추천)
  매년 증설되는 설비의 양이 커서 이에 대한 보완이 필요
- 일조량, 일조시간 데이터 추가해서 학습해 보기
  (앞선 논문에서 일조시간 데이터가 상관계수가 높아서 도움이 되었다고함)
- 발전량이 전력소비량을 상회할 경우 출력제한조치에 대한 데이터 추가 필요
  (발전 설비 용량이 커서 발전 설비를 멈춰야 하는 상황 발생, 발전 설비를 멈추면 발전량이 인위적으로 조절됨)
- 추가로 태양광 발전량과 상관관계를 갖는 데이터 찾아보기

수고하셨습니다