딥러닝 기반 학습을 통한 이차전지 SoH 예측

2018111902 정상훈

# 개념

NASA Battery Dataset[[1]](#footnote-1)의 B0005 Li-on Battery의 충방전 주기에 따른 배터리의 용량 변화에 따른 배터리의 SoH를 추정합니다. 딥러닝 기반 학습으로서 LSTM 모델을 채택합니다.

## SoH이란?

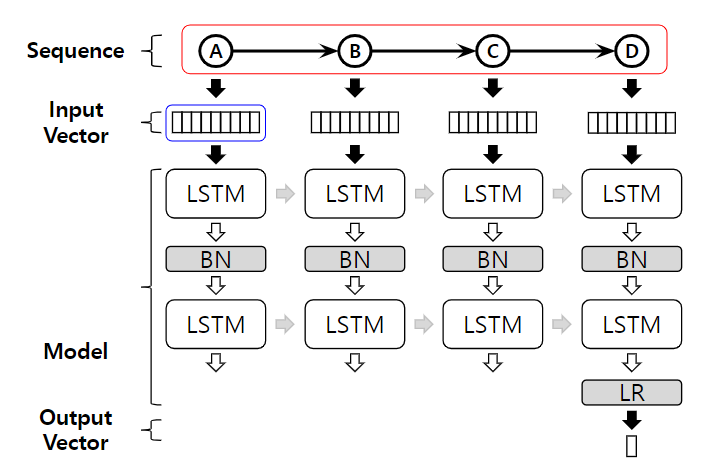
SoH는 "State of Health"로서, 배터리의 건강 상태를 나타냅니다. 이 용어는 주로 전지 또는 배터리의 성능, 용량, 충전 효율성 등을 평가하는 데 사용됩니다. 배터리 SoH는 시간이 지남에 따라 변화하며, 배터리의 노화, 손상, 사용 조건 등에 영향을 받습니다.

배터리 SoH가 높을수록 배터리의 성능이 좋다는 것을 의미하며, 낮을수록 성능이 감소하고 있거나 손상되었음을 나타냅니다. 배터리 SoH를 모니터링하면 사용자에게 배터리 교체 시기를 알려주거나, 배터리 관리를 최적화하여 최대한의 수명을 유지할 수 있습니다.

## LSTM이란?

LSTM은 "Long Short-Term Memory"의 약자로, 순환 신경망(RNN, Recurrent Neural Network)의 한 종류입니다. LSTM은 주로 시퀀스 데이터를 처리하고 장기 의존성(Long-Term Dependencies)을 효과적으로 학습하는 데 사용됩니다.

LSTM 셀은 크게 입력 게이트(input gate), 삭제 게이트(forget gate), 셀 상태 갱신, 출력 게이트(output gate)로 구성되어 있습니다. 이 구조를 통해 LSTM은 정보를 장기적으로 기억하거나 필요한 정보를 삭제하며, 입력과 출력을 효과적으로 제어하여 장기 의존성 문제를 극복할 수 있습니다.



*BN : Batch Normalization LR : Learning Rate*

# 코드

모델 설정을 위한 적절한 데이터세트으로 수정합니다. 불필요한 변수값을 제거하고 충방전 주기에 따른 배터리 용량을 SoH로 변환한 열을 삽입합니다.

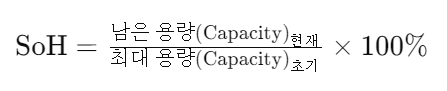


그림 SoH Equation



## ReadME

########## Battery Estimate by NASA Battery Dataset###########

                                #battery\_id : B0005

                                #total\_cycle : 168

                        #initial capacity : 1.85648742081815

                  #SoH equation : current capacity / initial capacity

## 라이브러리

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler

from sklearn.model\_selection import train\_test\_split

from keras.models import Sequential

from keras.optimizers import Adam

from sklearn.metrics import mean\_squared\_error

from keras.layers import Dense, LSTM, Dropout

## 데이터 불러오기

from google.colab import drive

drive.mount('/content/drive')

data = pd.read\_csv("/content/drive/MyDrive/battery\_SoH.CSV")

data.head()

구글 드라이브를 이용하여 CSV 형식 데이터를 로드합니다.

|  | **Cycle** | **Capacity** | **SoH** |
| --- | --- | --- | --- |
| **0** | 1 | 1.856487 | 1.000000 |
| **1** | 2 | 1.846327 | 0.994527 |
| **2** | 3 | 1.835349 | 0.988614 |
| **3** | 4 | 1.835263 | 0.988567 |
| **4** | 5 | 1.834646 | 0.988235 |

data.shape

(168, 3)

## X, y 데이터 할당

X = data.drop(['Capacity','SoH'],axis = 1)

y = data['SoH']

X는 Cycle, y는 SoH를 할당하여 Cycle에 따른SoH를 추정할 수 있도록 한다. (순차적)

## X, y 그래프 분석

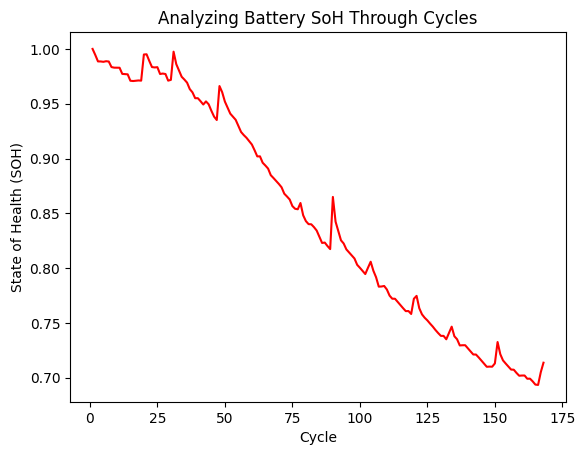
# Plot graph

plt.plot(X, y, linestyle='-', color='r')

plt.title("Analyzing Battery SoH Through Cycles")

plt.xlabel("Cycle")

plt.ylabel("State of Health (SOH)")

plt.show()

## 데이터 Train/Test분할

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.3,  random\_state = 7)

print('Train shape :',X\_train.shape)

print('Test shape :',X\_test.shape)

Train shape : (117, 1)

Test shape : (51, 1)

Train\_test\_split을 이용하여 X\_train, X\_test, y\_train, y\_test 분할, test\_size = 0.3, train\_size = 0.7 비율로 구성한다.

## 스케일링

scaler = MinMaxScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

## 입력 데이터 차원 변경

X\_train = X\_train.reshape(X\_train.shape[0],X\_train.shape[1],1)

X\_test = X\_test.reshape(X\_test.shape[0],X\_test.shape[1],1)

Sequence 정보를 1차원 array로 가지고 있기 때문에 각각의 *Cycle* 값을 Input Vector로 만들어 주기 위해 차원을 확장합니다.

## LSTM 모델 생성

model=Sequential()

model.add(LSTM(units=50, return\_sequences = True, input\_shape=(X\_train.shape[1], X\_train.shape[2])))

model.add(Dropout(0.3))

model.add(LSTM(units=50, return\_sequences = True))

model.add(Dense(10))

model.add(Dense(1))

model.summary()

Model: "sequential\_25"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

lstm\_42 (LSTM) (None, 1, 50) 10400

dropout\_20 (Dropout) (None, 1, 50) 0

lstm\_43 (LSTM) (None, 1, 50) 20200

dense\_38 (Dense) (None, 1, 10) 510

dense\_39 (Dense) (None, 1, 1) 11

=================================================================

Total params: 31121 (121.57 KB)

Trainable params: 31121 (121.57 KB)

Non-trainable params: 0 (0.00 Byte)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

<LSTM>

return\_sequences=True는 LSTM을 여러층 쌓을 때 모든 Sequence의 hidden state를 출력하게 하는 설정이다. 모든 Squence(140)에 대해서 hidden state가 출력되었다.

이때 hidden state의 길이는 100이다.

<Dropout>

30%의 가중치를 랜덤하게 제거한다. 이때 차원정보는 변하지 않는다.

<Dense>

입력받은 길이 50의 Vector로부터 길이 10의 Vector를 출력한다.

재차, 입력받은 길이 10의 Vector로부터 길이 1의 Vector를 출력한다.

## 컴파일

model.compile(loss='mse', optimizer='adam')

## 모델 학습

training\_record = model.fit(X\_train, y\_train, epochs=60, batch\_size=32,

                            validation\_data=(X\_train, y\_train), shuffle=True)

데이터 수가 작기 때문에 validation data에 train set을 삽입하였다.

## Learning curve 시각화

fig, ax = plt.subplots()

plt.plot(training\_record.history["loss"], label="Training Loss")

plt.plot(training\_record.history["val\_loss"], label="Validation Loss")

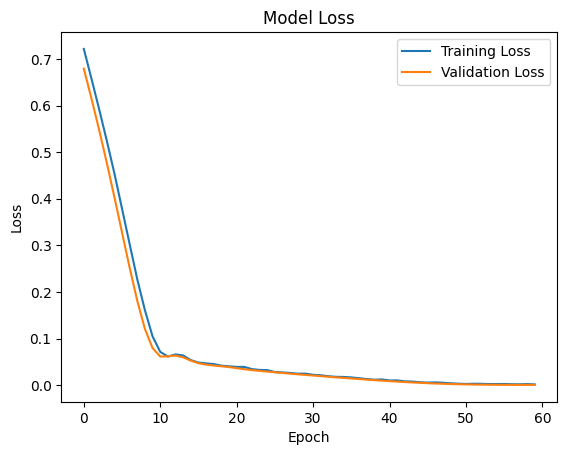
plt.legend()

plt.title("Model Loss")

plt.xlabel("Epoch")

plt.ylabel("Loss")

plt.show()



## 예측

y\_pred = model.predict(X\_test)

## MSE 계산하기

y\_pred = y\_pred.reshape(-1)

mse\_test = mean\_squared\_error(y\_test, y\_pred)

print(f'Mean Squared Error: {round(mse\_test\*100,3)}%')

Mean Squared Error: 0.03%

1. <https://www.kaggle.com/datasets/patrickfleith/nasa-battery-dataset/data> [↑](#footnote-ref-1)