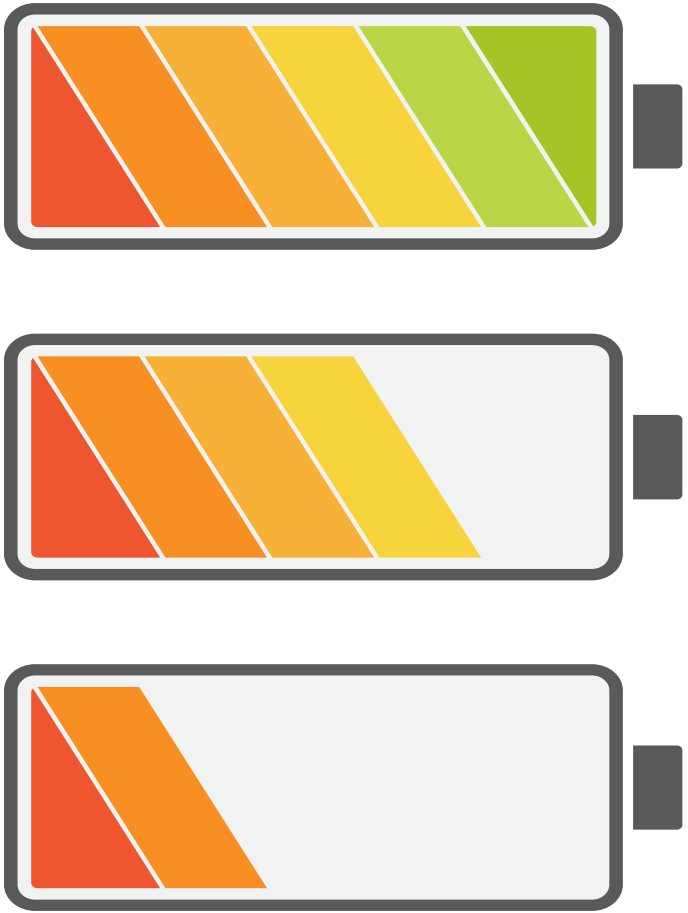
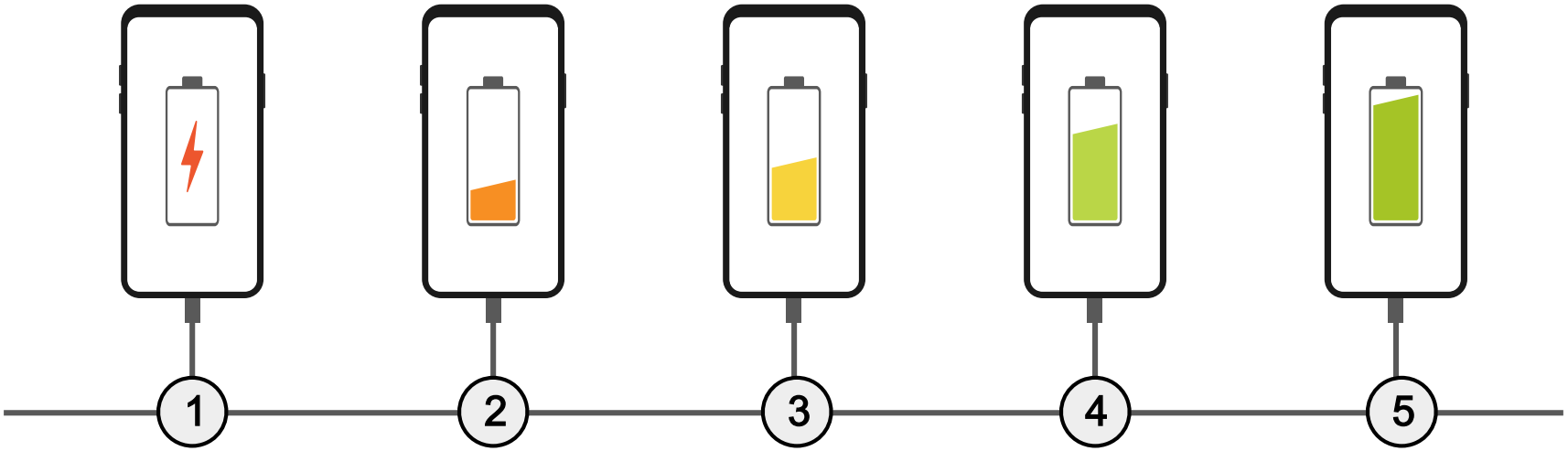


基於深度學習之鋰離子電池健康狀態預測模型

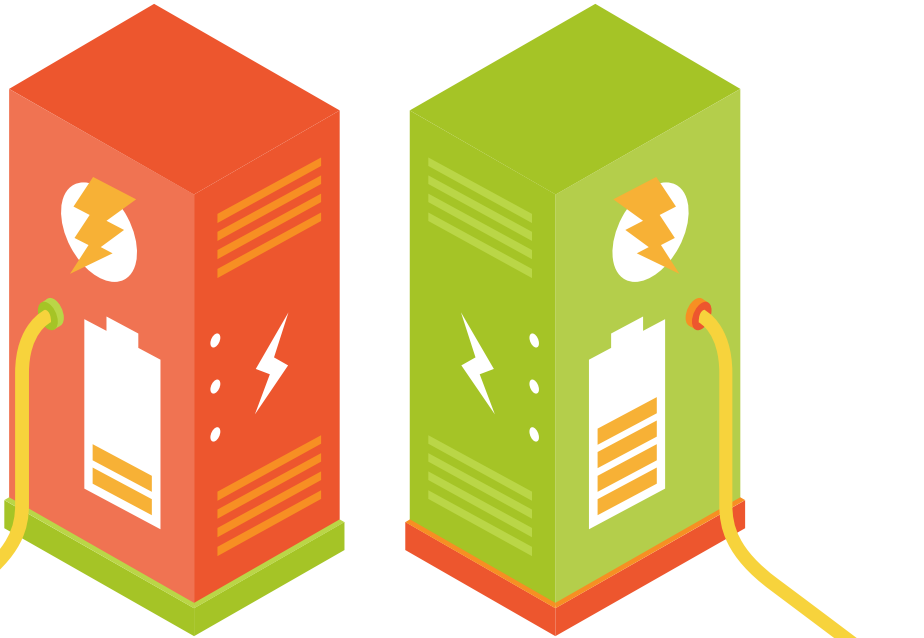
Deep Learning-Based State of Health (SOH) Prediction for Lithium-Ion Battery

伍仰輝 109034403 邱靖中 109034538
張郁杰 109034532 蘇泳心 109034541

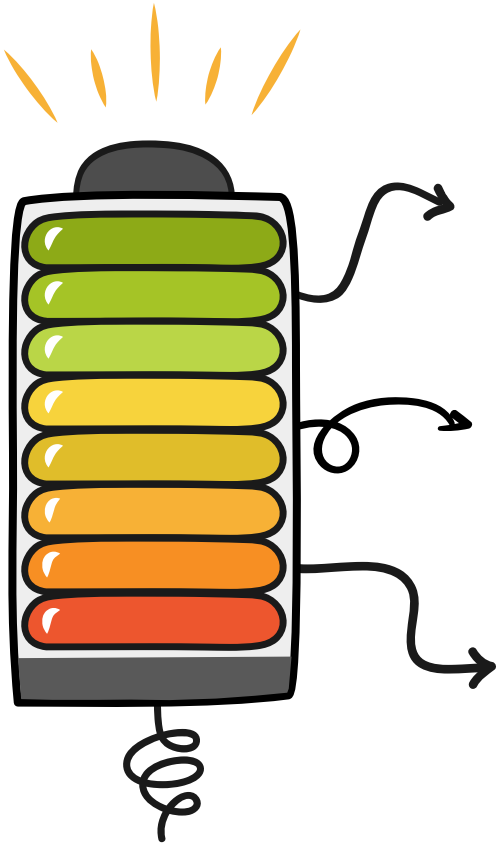
Outline


1
2
3
4
5
Introduction
Data Preprocessing
Deep Learning Model
Hyperparameter Tuning
Conclusion

01
Introduction



背景與動機

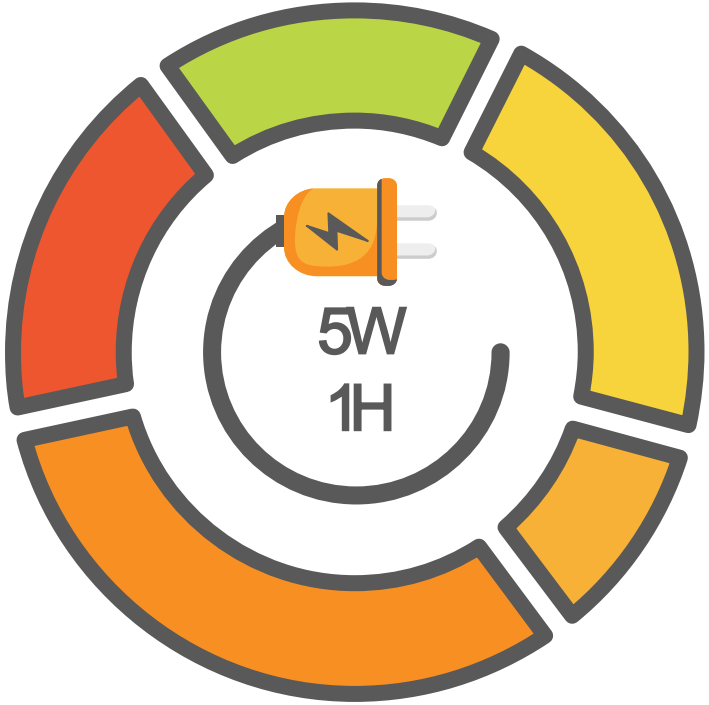


鋰離子電池由於具有能量密度高、使用壽命長、放電率低、無污染等多項優點，成為現在電池發展的趨勢

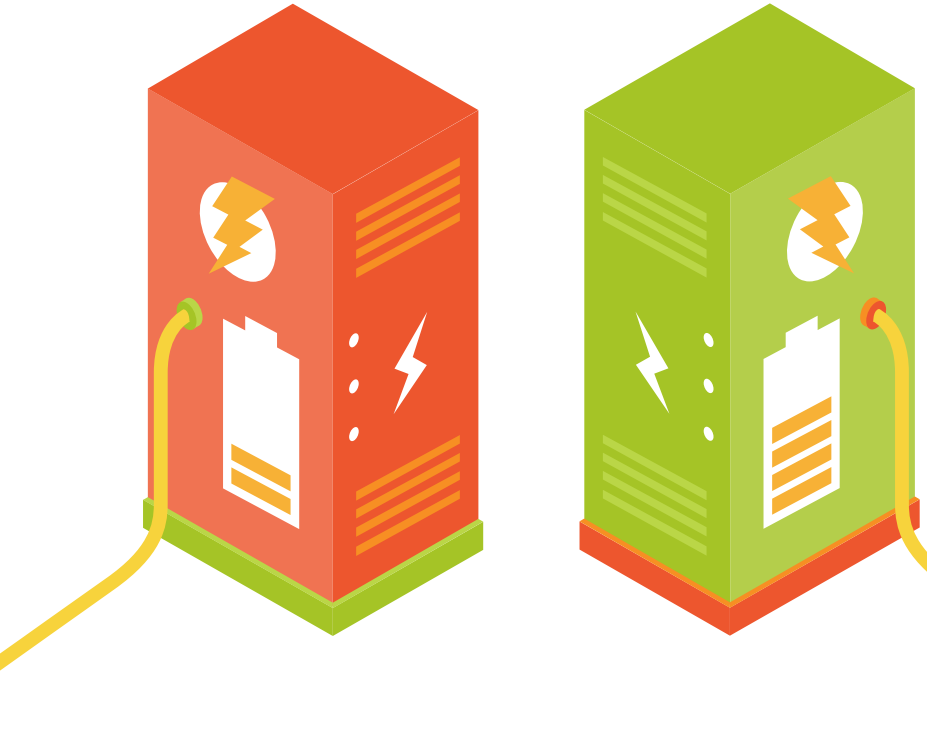
通過評估鋰離子電池的健康狀態（SOH），可以瞭解每個電池的健康程度，以確保電源系統的穩定性

提前預知電池健康狀態，並更換達到故障閾值的電池，將可以確保電池之安全運行

Problem Definition – 5WlH



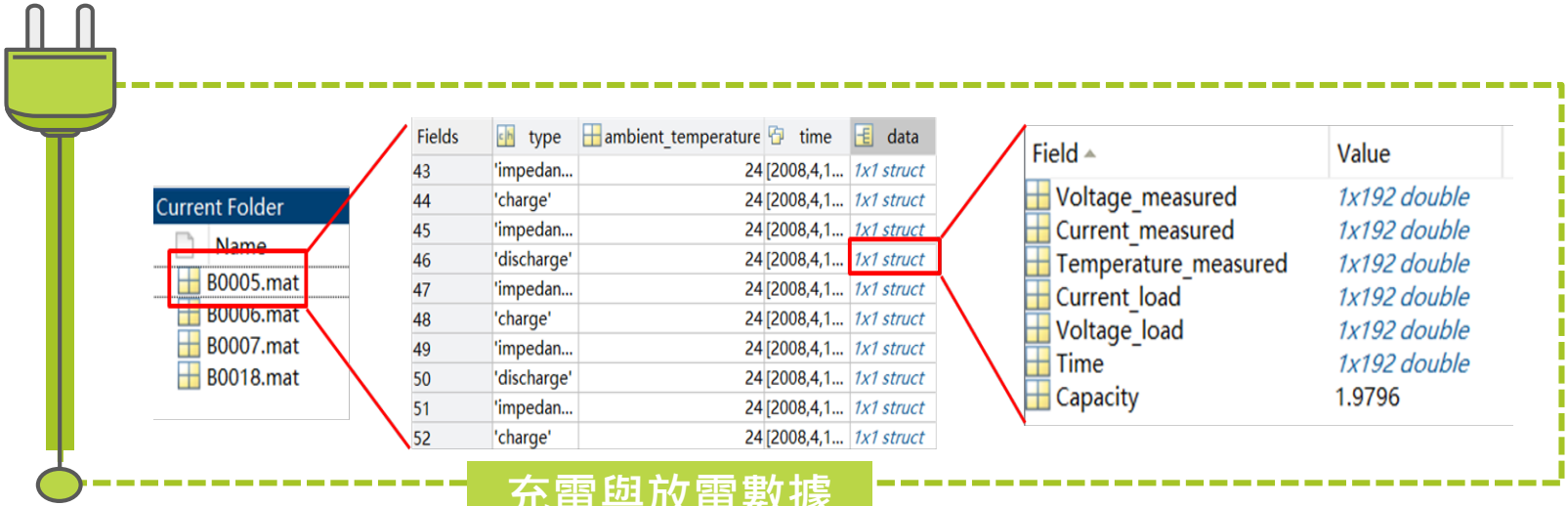
WHAT	鋰離子電池的預測性維護
WHEN	鋰離子電池失效前
WHO	工業設備及其餘需要鋰離子電池之事務，如自動搬運車等
WHERE	使用鋰離子電池之各類設備保養
WHY	若可準確預估鋰離子電池之健康度，即可於電池失效前做出預防措施
HOW	資料分析、機器學習



02 Data-Preprocessing

使用資料集

NASA Prognostics Data Repository “Battery Data Set” (2007) 的公開資料集 (電池編號：B0005)

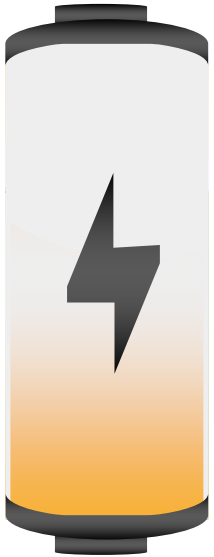


特徵種類

Charging

- Battery voltage
- Battery current
- Battery temperature
- Charger voltage
- Charger current
- Cumulative time

⋮



Discharging

- Battery voltage
- Battery current
- Battery temperature
- Load voltage
- Load current
- Cumulative time

⋮

電池失效定義

由於電池各自的充電與放電數量不一致，因此本研究將總次數採以較小之次數當作循環次數

Cycle SOH

(121, 0.7191275088396454),
(122, 0.7086773587242763),
(123, 0.7034908318388216),
(124, 0.7006018891793813),
(125, 0.6983504116363164),
(126, 0.695642394426363),
(127, 0.6931143838989174),

電池失效

- 在第 124 次充放電電池的SOH已經趨於70%
- 而第 125 次以低於標準，將判定為失效

Data-Preprocessing

1. 將 MATLAB 檔轉換至 Json 檔 — **檔案轉換**
2. 萃取充電與放電狀態的數據 — **特徵萃取**
3. 運用 GRA 篩選對目標值有強烈關係之特徵種類 — **標準化**
4. 利用最大最小標準化將數據進行轉換 — **特徵篩選**
5. 將資料分成訓練資料、驗證資料與測試資料 — **資料分割**

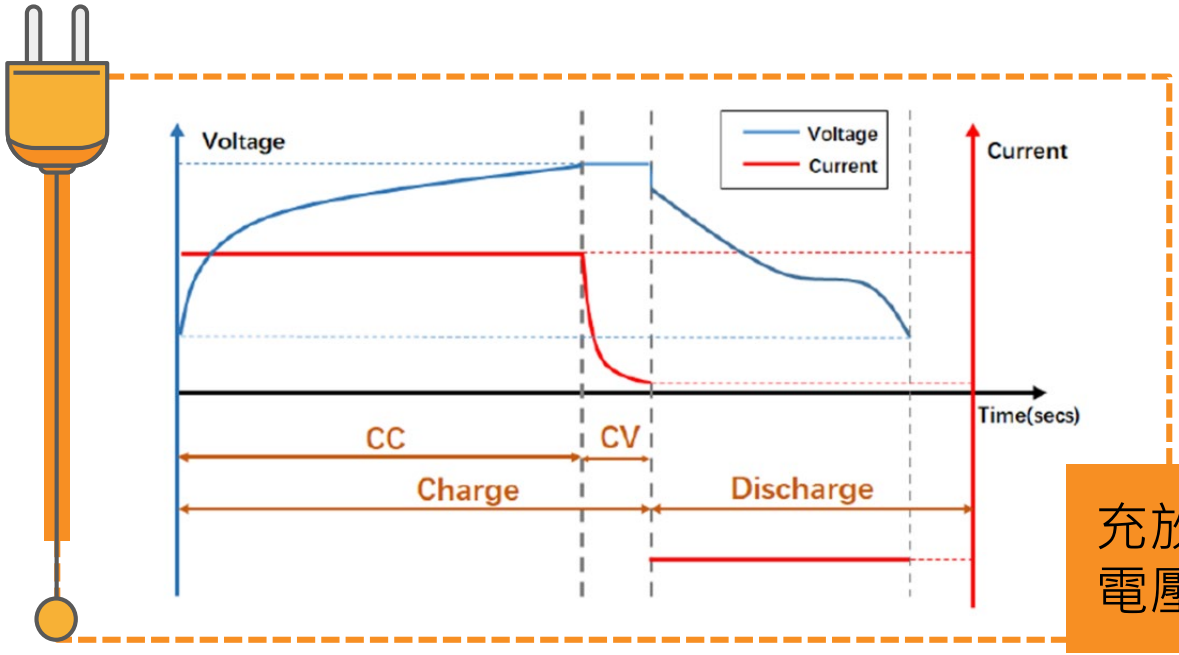
檔案轉換

將原始資料由 MATLAB 檔轉換成 json 的架構方便後續 python 資料萃取的作業

```python
#轉換資料
def build_dictionaries(mess):

    discharge, charge, impedance = {}, {}, {}

    for i, element in enumerate(mess):

        step = element[0][0]

        if step == 'discharge':
            discharge[str(i)] = {}
            discharge[str(i)]["ambient_temperature"] = str(element[1][0][0])
            year = int(element[2][0][0])
            month = int(element[2][0][1])
            day = int(element[2][0][2])
            hour = int(element[2][0][3])
            minute = int(element[2][0][4])
            second = int(element[2][0][5])
            millisecond = int((second % 1)*1000)
            date_time = datetime.datetime(year, month, day, hour, minute, second, millisecond)

            discharge[str(i)]["date_time"] = date_time.strftime("%d %b %Y, %H:%M:%S")

            data = element[3]

            discharge[str(i)]["Voltage_measured"] = data[0][0][0][0].tolist()
            discharge[str(i)]["Current_measured"] = data[0][0][1][0].tolist()
            discharge[str(i)]["Temperature_measured"] = data[0][0][2][0].tolist()
            discharge[str(i)]["Current_load"] = data[0][0][3][0].tolist()
            discharge[str(i)]["Voltage_load"] = data[0][0][4][0].tolist()
            discharge[str(i)]["Time"] = data[0][0][5][0].tolist()
            discharge[str(i)]["Capacity"] = float(data[0][0][6][0][0])

        if step == 'charge':
            charge[str(i)] = {}
            charge[str(i)]["ambient_temperature"] = str(element[1][0][0])
            year = int(element[2][0][0])
            month = int(element[2][0][1])
            day = int(element[2][0][2])
            hour = int(element[2][0][3])
            minute = int(element[2][0][4])
            second = int(element[2][0][5])
            millisecond = int((second % 1)*1000)
            date_time = datetime.datetime(year, month, day, hour, minute, second, millisecond)

            charge[str(i)]["date_time"] = date_time.strftime("%d %b %Y, %H:%M:%S")

            data = element[3]
```

特徵萃取

擷取各循環中每種數據充放電結束的瞬間數值與其紀錄時間

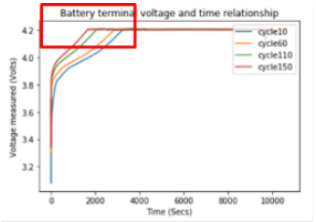


充放電狀態電流與電壓趨勢圖

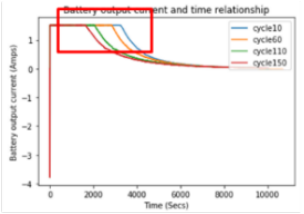
特徵萃取

Charging

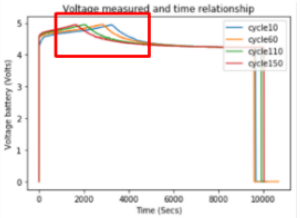
- 隨著充電次數的增加，其趨勢圖會有時間評移的情況
- 電壓、電流、溫度等特徵與時間之關係圖
- 擷取每次循環之最早穩定的值與紀錄時間當作其中兩項特徵值



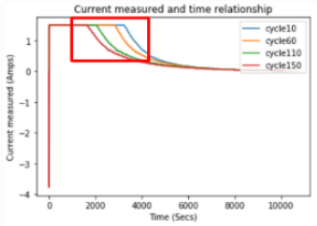
電池電壓與累計時間關係圖



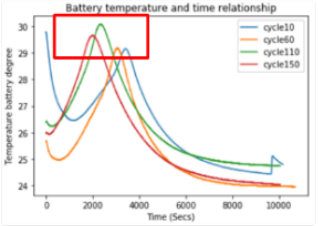
電池電流與累計時間關係圖



充電器電壓與累計時間關係圖



充電器電流與累計時間關係圖

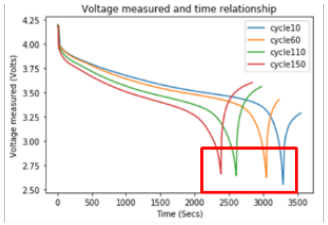


電池溫度與累計時間關係圖

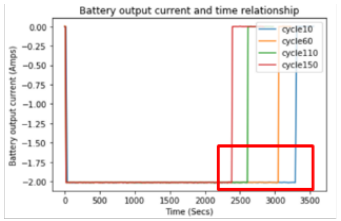
特徵萃取

Discharging

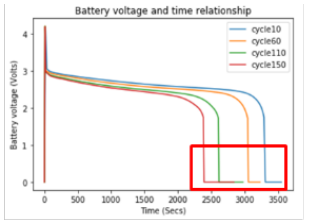
- 隨著放電次數的增加，其趨勢圖會有時間評移的情況
- 電壓、電流、溫度等特徵與時間之關係圖
- 擷取每次循環之最早穩定的值與紀錄時間當作其中兩項特徵值



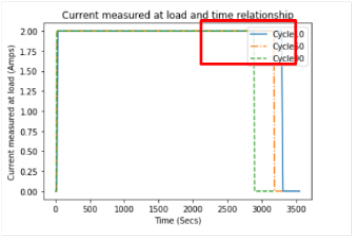
電池電壓與累計時間關係圖



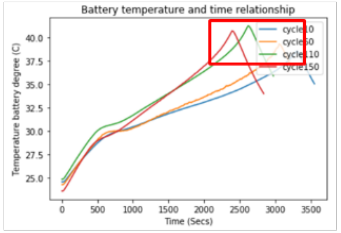
電池電流與累計時間關係圖



負載電壓與累計時間關係圖



負載電流與累計時間關係圖



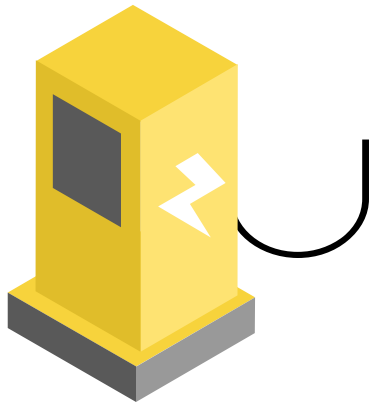
電池溫度與累計時間關係圖

特徵總表

Charging

Discharging

排序	特徵名稱
1	Charge_Voltage_Measured
2	Charge_Voltage_Measured_Time
3	Charge_Current_Measured
4	Charge_Current_Measured_Time
5	Charge_Temperature_Measured
6	Charge_Temperature_Measured_Time
7	Charge_Current_Charger
8	Charge_Current_Charger_Time
9	Charge_Voltage_Charger
10	Charge_Voltage_Charger_Time
11	Discharge_Voltage_Measured
12	Discharge_Voltage_Measured_Time
13	Discharge_Current_Measured
14	Discharge_Current_Measured_Time
15	Discharge_Temperature_Measured
16	Discharge_Temperature_Measured_Time
17	Discharge_Current_Load
18	Discharge_Current_Load_Time
19	Discharge_Voltage_Load
20	Discharge_Voltage_Load_Time

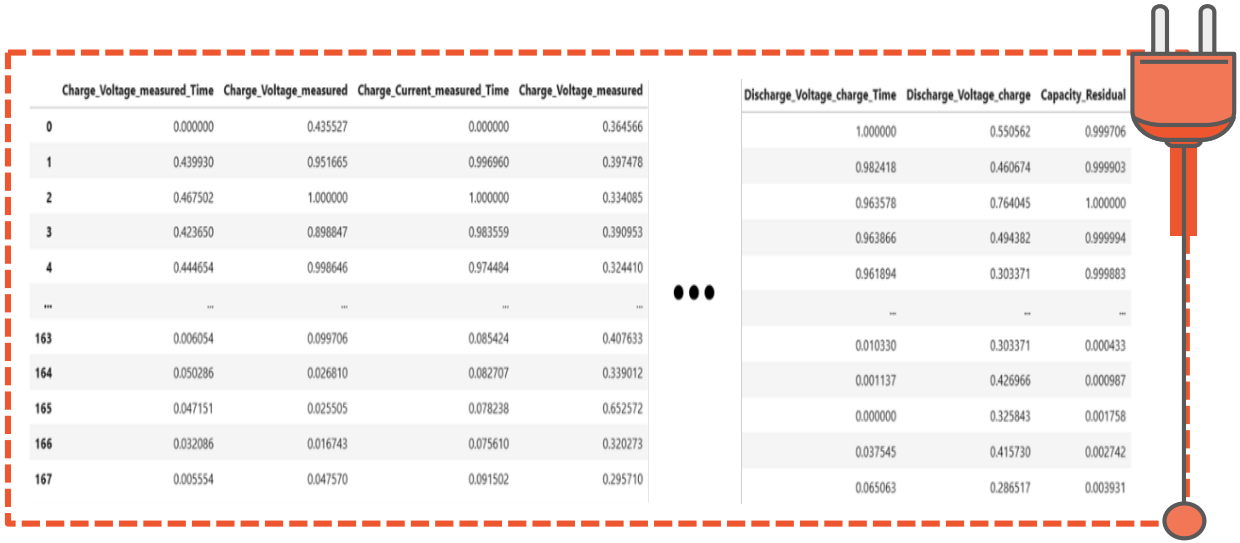


標準化

為了減少數據的跨度，以最大最小標準化調整數據，以利後續模型的預測準確度

$$X_{nom} = \frac{X - X_{min}}{X_{max} - X_{min}} \in [0,1]$$

- 20 Features
- Range from 0 to 1



	Charge_Voltage_measured_Time	Charge_Voltage_measured	Charge_Current_measured_Time	Charge_Voltage_measured
0	0.000000	0.435527	0.000000	0.364566
1	0.439930	0.951665	0.996960	0.397478
2	0.467502	1.000000	1.000000	0.334085
3	0.423650	0.898847	0.983559	0.390953
4	0.444654	0.998646	0.974484	0.324410
...	...	...	...	...
163	0.006054	0.099706	0.085424	0.407633
164	0.050286	0.026810	0.082707	0.339012
165	0.047151	0.025505	0.078238	0.652572
166	0.032086	0.016743	0.075610	0.320273
167	0.005554	0.047570	0.091502	0.295710

Discharge_Voltage_charge_Time	Discharge_Voltage_charge	Capacity_Residual
1.000000	0.550562	0.999706
0.982418	0.460674	0.999903
0.963578	0.764045	1.000000
0.963866	0.494382	0.999994
0.961894	0.303371	0.999883
...	...	...
0.010330	0.303371	0.000433
0.001137	0.426966	0.000987
0.000000	0.325843	0.001758
0.037545	0.415730	0.002742
0.065063	0.286517	0.003931

特徵篩選

灰色關聯分析(Grey Relational Analysis, GRA)進行特徵篩選，找出和SOH關聯最大的特徵種類

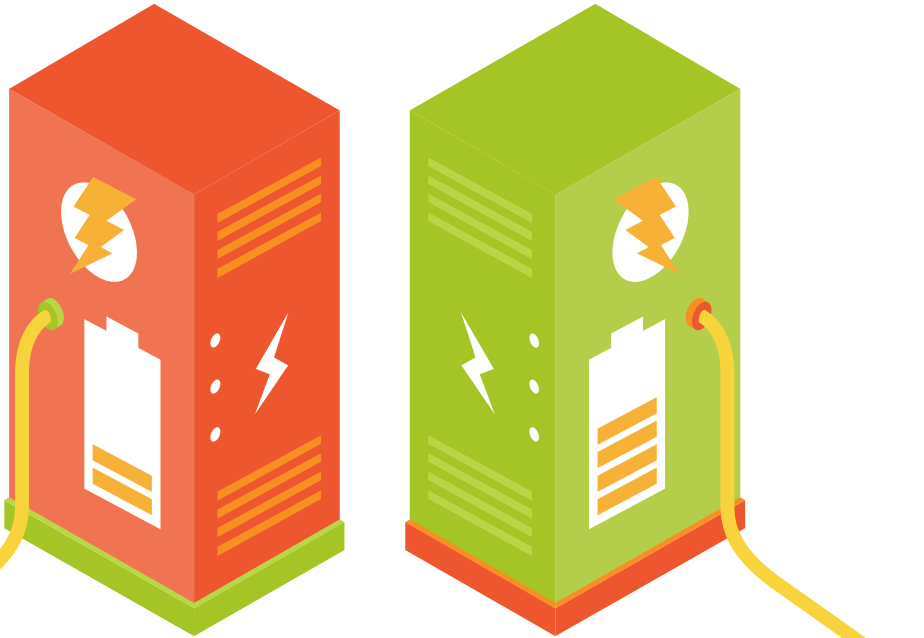
```
def Gray(dataset):

    gray=dataset

    gray=(gray - gray.min()) / (gray.max() - gray.min())
    gray=gray.fillna(method='bfill')
    gray=gray.fillna(method='ffill')


    std=gray.iloc[:,20]#標準
    ce=gray.iloc[:,0:20]#比較

    n=ce.shape[0]
    m=ce.shape[1]

    a=zeros([m,n])
    for i in range(m):
        for j in range(n):
            a[i,j]=abs(ce.iloc[j,i]-std[j])

    c=amax(a)
    d=amin(a)


    result=zeros([m,n])
    for i in range(m):
        for j in range(n):
            result[i,j]=(d+0.5*c)/(a[i,j]+0.5*c)

    #求均值得到灰色關聯值
    result2=zeros(m)
    for i in range(m):
        result2[i]=mean(result[i,:])
    RT=pd.DataFrame(result2)
```

資料分割

訓練資料	80%	99筆
驗證資料	10%	12筆
測試資料	10%	13筆

03

Deep Learning Model



模型建立流程圖

遞歸神經網絡 (Recurrent neural network, RNN)

Fundamenta

- 專門設計用於處理時間序列數據的模型，適用於影片、天氣預報、股票等
- 核心概念是強調數據與數據之間的關係，它會將根據過去的記憶確定當前的輸出
- 神經網絡只記住最近的事件，而較早的事件已被遺忘

Advantage

- 適用於處理時間序列數據

長短期記憶 (Long short-term memory, LSTM)

Fundamenta

- 與RNN相比，LSTM可以更好地處理長期順序數據
- 具有三個控制門，即輸入門，輸出門和忘記門
- LSTM會自己學習，輸出門則控制是否可以讀取存儲單元的值，而遺忘門決定何時應清除存儲單元的值。

Advantage

- 適用於處理時間序列數據
- 考慮輸入的時間順序
- 有選擇地忘記一些記憶以避免干擾現有的模型擬合

門控循環單元 (Gated recurrent unit, GRU)

Fundamenta

- 它用更新門代替 LSTM 中的忘記門和輸入門
- 它合併單元狀態和隱藏狀態以計算新訊息

Advantage

- 適用於時間序列數據
- 考慮輸入的時間順序
- 小樣本量的性能可能比LSTM更好

評估指標

RMSE

$$\sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}$$

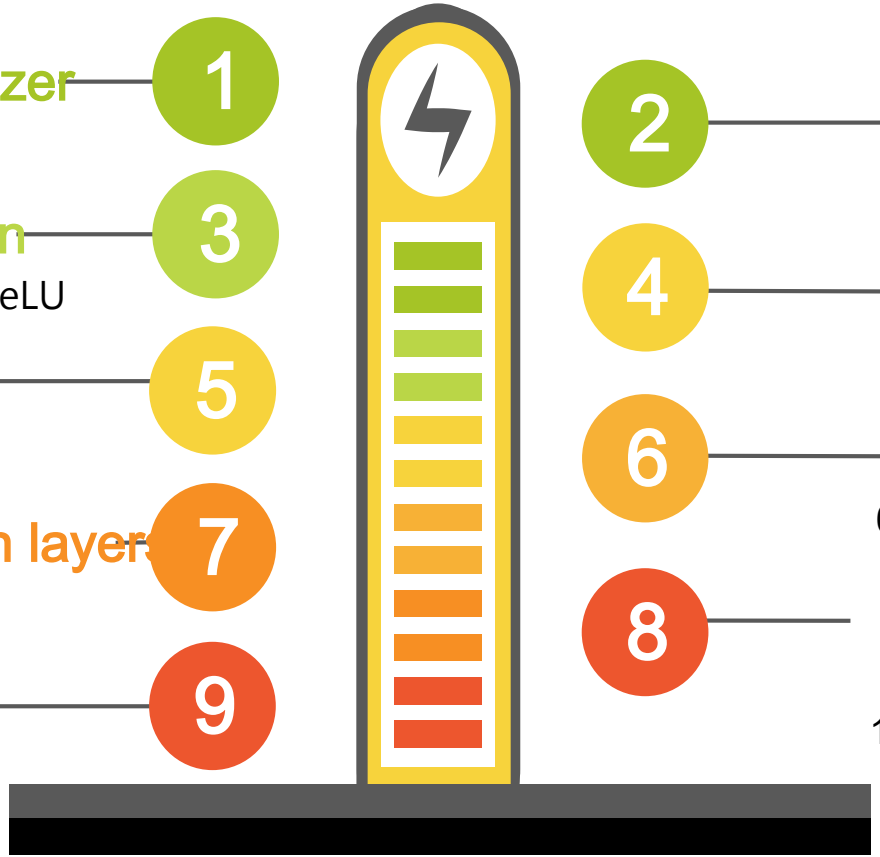
MAE

$$\frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

MAPE

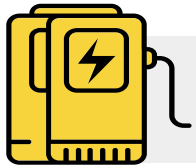
$$\frac{1}{N}\sum_{i=1}^{N}\left|\frac{y_i - \hat{y}_i}{y_i}\right| X100\%$$

超參數種類



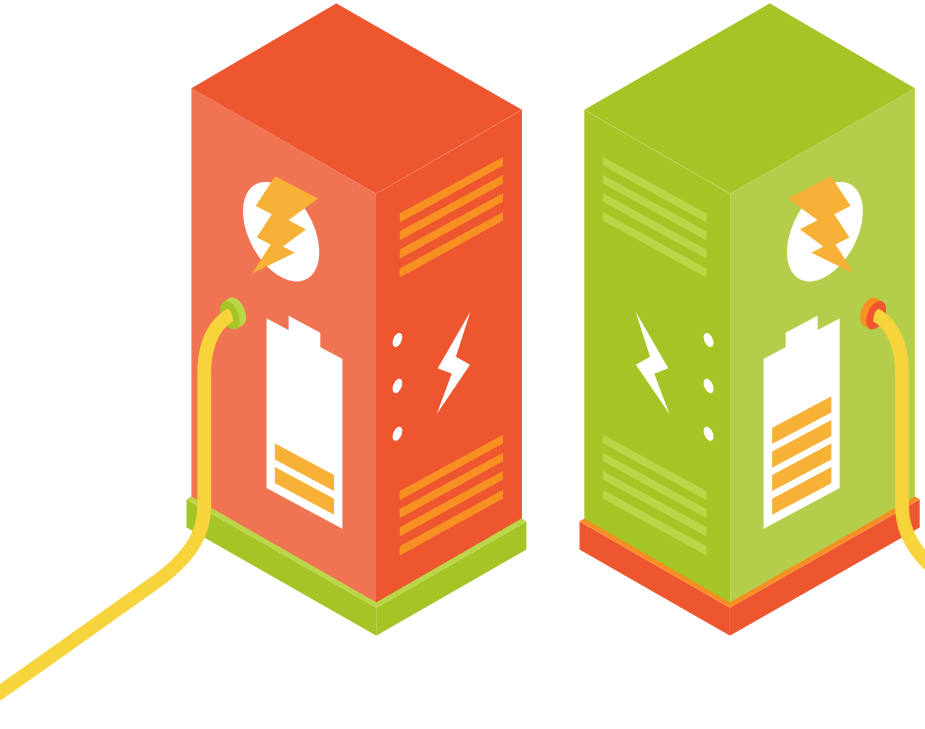
1. **Parameter optimizer**
   Adam
2. **Loss function**
   MSE
3. **Activation function**
   ReLU, Softmax, Leaky ReLU
4. **Batch size**
   10, 16, 32
5. **Epochs**
   500
6. **Dropout rate**
   0.00, 0.01, 0.02, ..., 0.20
7. **Number of hidden layer**
   1, 2, 3
8. **Number of neurons**
   16, 32, 48, 64, 80, 96,
   112, 128, 144, 160, 176,
   192, 208, 224, 240, 256
9. **Learning rate**
   0.01, 0.001, 0.0001

RNN建模



預設參數如右，程式碼如下

```
n_batch = 10
model_RNN_01 = Sequential()
model_RNN_01.add(SimpleRNN(64, batch_input_shape=(n_batch, X_train0.shape[1], X_train0.shape[2])))
model_RNN_01.add(Dense(128, activation='relu'))
model_RNN_01.add(Dropout(0.02))
model_RNN_01.add(Dense(256, activation='relu'))
model_RNN_01.add(Dropout(0.07))
model_RNN_01.add((Dense(1)))
adam = keras.optimizers.Adam(lr=0.001)
model_RNN_01.compile(loss='mean_squared_error', optimizer = adam)
history = model_RNN_01.fit(X_train0, y_train0,epochs=500,validation_data=(X_valid, y_valid), verbose=2, shuffle=False)
```

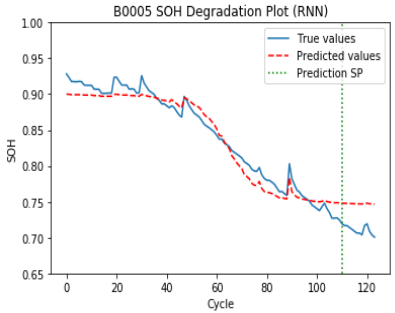


04 Hyperparameter Tuning

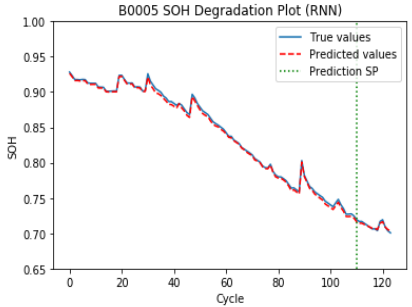
RNN模型手動調參數

使用不同 Activation Function 來評估

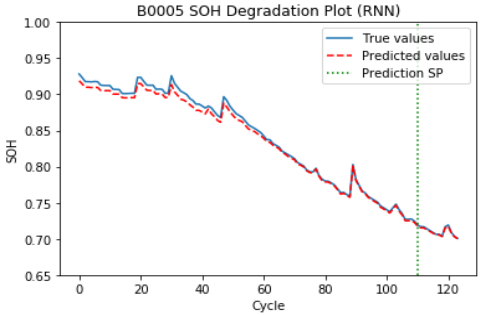
Activation Function	RMSE	MAPE	MAE	Selection
Softmax	0.0175	1.81%	0.0143	
ReLU	0.0043	0.45%	0.0038	V
Leaky ReLU	0.0078	0.78%	0.0067	



Softmax



ReLU

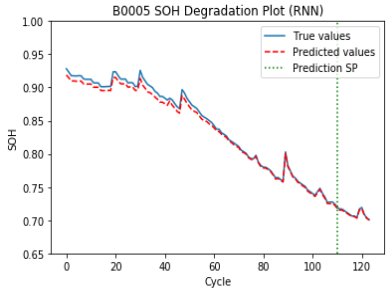


LeakyReLU

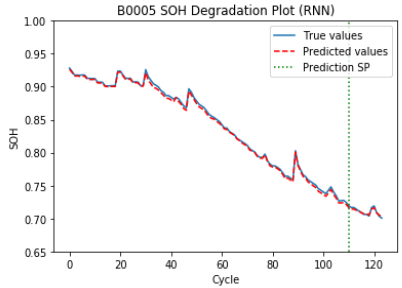
RNN模型手動調參數

使用不同 Batch Size 來評估

Batch Size	RMSE	MAPE	MAE	Selection
10	0.0043	0.45%	0.0038	
16	0.0050	0.51%	0.0040	
32	0.0020	0.28%	0.0020	V



Batch Size = 16



Batch Size =32

貝葉氏超參數最佳化

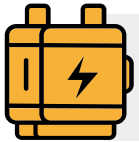
9 個超參數共 25,920 種總組合 1

傳統手動調超參數，無法完全嘗試所有的組合，也有可能無視交互作用 2

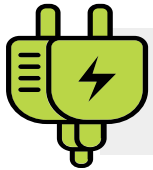
貝葉氏最佳化為一種含有目標函數的機率模型 3

可以將模型擬合到相對適當的觀測值，並使用該模型來預測參數空間 4

RNN模型貝葉氏超參數最佳化



進行貝葉氏超參數最佳化

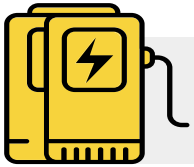


結果會將最為良好的前十個列出來

```
#測試的試驗總數
MAX_TRIALS = 10
#訓練次數
EXECUTIONS_PER_TRIAL = 3


tuner1 = BayesianOptimization(
    model0,
    objective='val_loss',
    max_trials=MAX_TRIALS,
    executions_per_trial=EXECUTIONS_PER_TRIAL,
    directory='test_dir',
    project_name='BayesianOptimizationB62_valid3_0704',

)
tuner1.search_space_summary()
```

```
Results summary
|-Results in test_dir\BayesianOptimizationB52_valid1_1215_4
|-Showing 10 best trials
|-Objective(name='val_loss', direction='min')

Trial summary
|-Trial ID: 18991870b5ed55bf5ded6201e0c3f27e
|-Score: 1.4281386838553722e-08
|-Best step: 0

Hyperparameters:
|-dropout_1: 0.08
|-dropout_2: 0.01
|-learning_rate: 0.001
|-units1: 224
|-units2: 224
|-units3: 16
```

RNN模型貝葉氏超參數最佳化



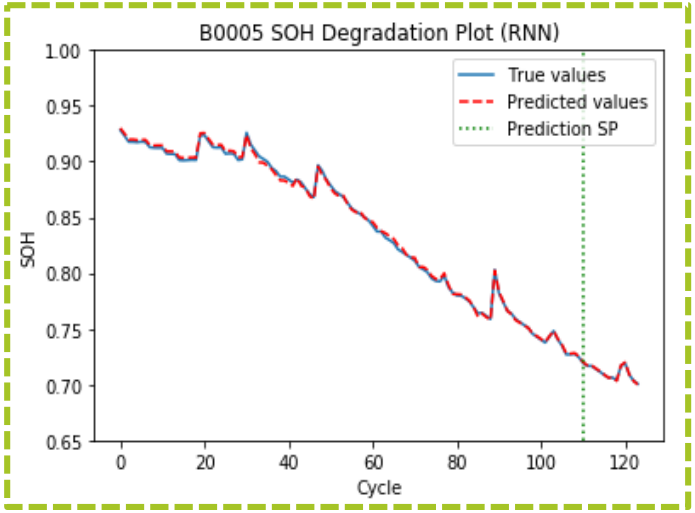
透過貝葉氏超參數最佳化後，
找出最佳的設定如下，以下僅列出 3 種

超參數	參數設定		
Activation Function	ReLU	ReLU	ReLU
Batch Size	32	32	32
第一層神經元數	224	128	80
第二層神經元數	224	240	96
第三層神經元數	16	96	224
第一層 Dropout	0.08	0.07	0.02
第二層 Dropout	0.01	0.09	0.01
學習率	0.001	0.001	

最好的前3種組合，僅神經元數與 Dropout rate 不同

RNN模型評估結果

使用第一種超參數組合，結果如下



評估指標	貝葉氏最佳化結果
RMSE	0.0019
MAPE	0.19%
MAE	0.0016

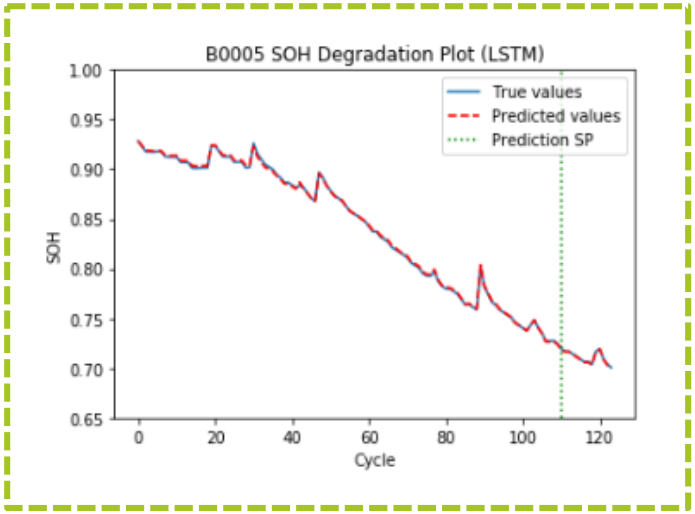
LSTM模型手動調參數

Activation Function	RMSE	MAPE	MAE	Selection
Softmax	0.0124	1.63%	0.0227	
ReLU	0.0019	0.25%	0.0035	V
Leaky ReLU	0.0054	0.71%	0.0099	

Batch Size	RMSE	MAPE	MAE	Selection
10	0.0019	0.25%	0.0035	
16	0.0048	0.65%	0.0047	
32	0.001	0.13%	0.0009	V

LSTM模型貝葉氏超參數最佳化

使用第一種超參數組合，結果如下



評估指標	貝葉氏最佳化結果
RMSE	0.0004
MAPE	0.05%
MAE	0.0004

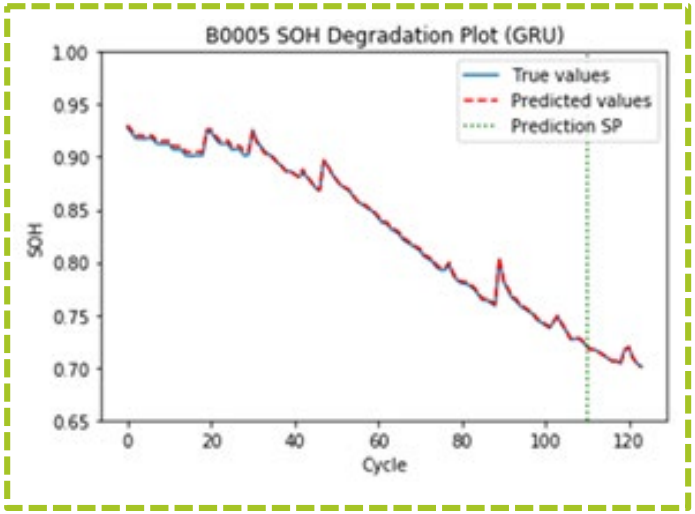
GRU模型手動調參數

Activation Function	RMSE	MAPE	MAE	Selection
Softmax	0.0192	1.92%	0.0192	
ReLU	0.0068	0.68%	0.0068	V
Leaky ReLU	0.0124	1.24%	0.0124	

Batch Size	RMSE	MAPE	MAE	Selection
10	0.0068	0.68%	0.0068	
16	0.0051	0.59%	0.0043	
32	0.0029	0.40%	0.0029	V

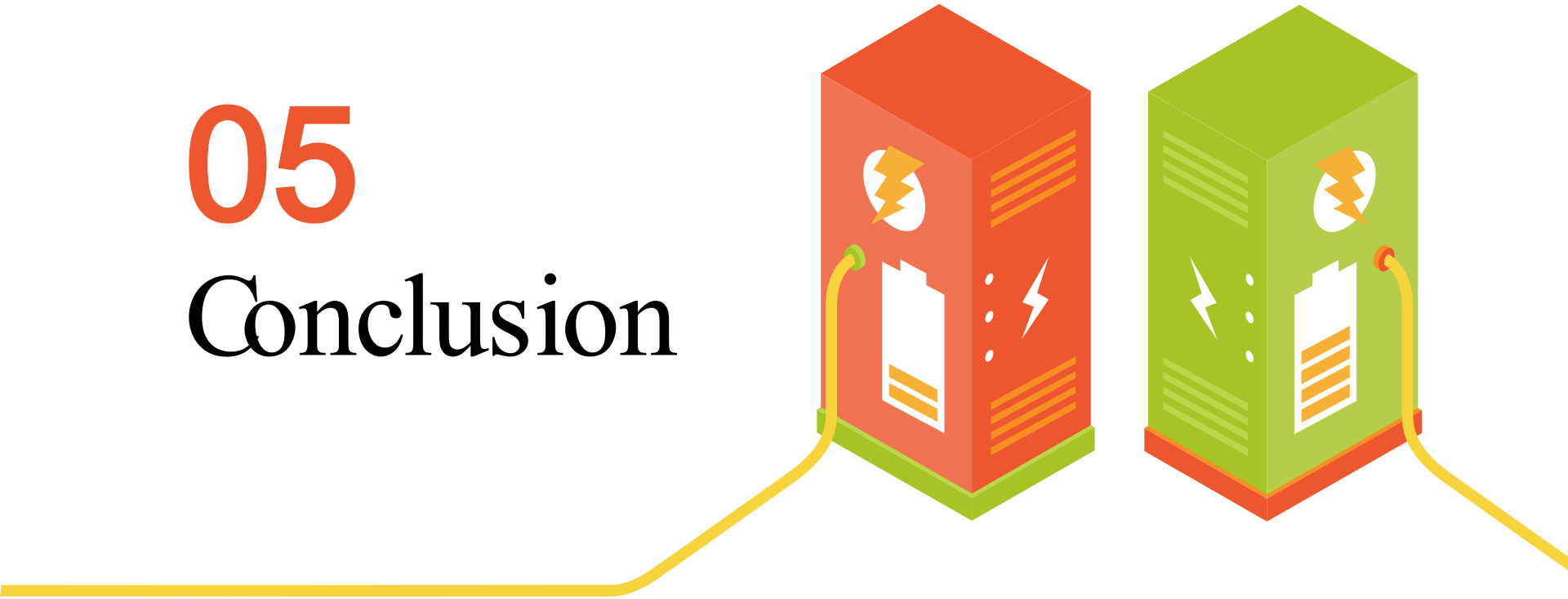
GRU模型貝葉氏超參數最佳化

使用第一種超參數組合，結果如下



評估指標	貝葉氏最佳化結果
RMSE	0.0009
MAPE	0.13%
MAE	0.0009

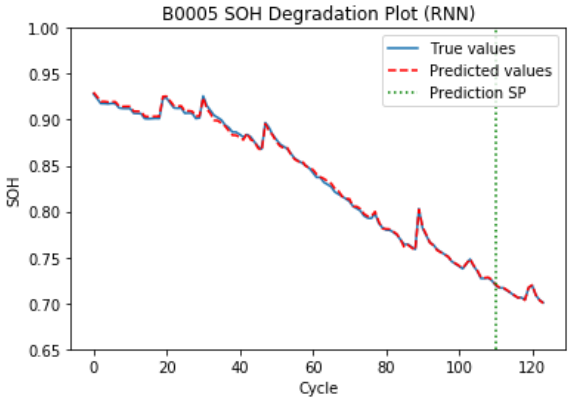
05
Conclusion



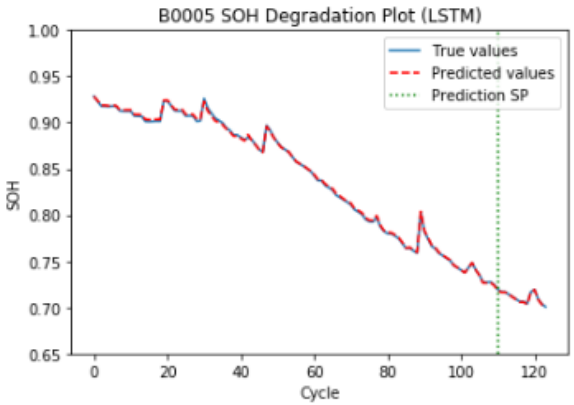
各演算法比較

經貝葉氏最佳化後之結果

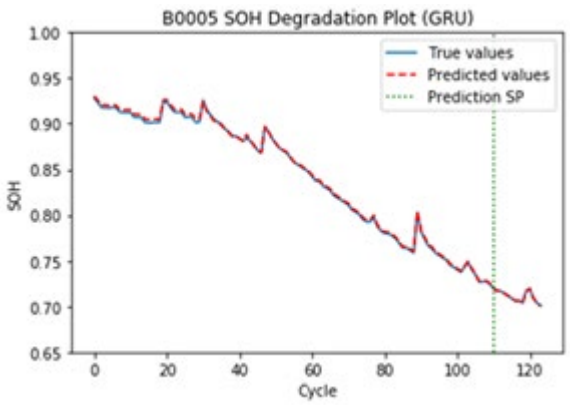
RNN



LSTM



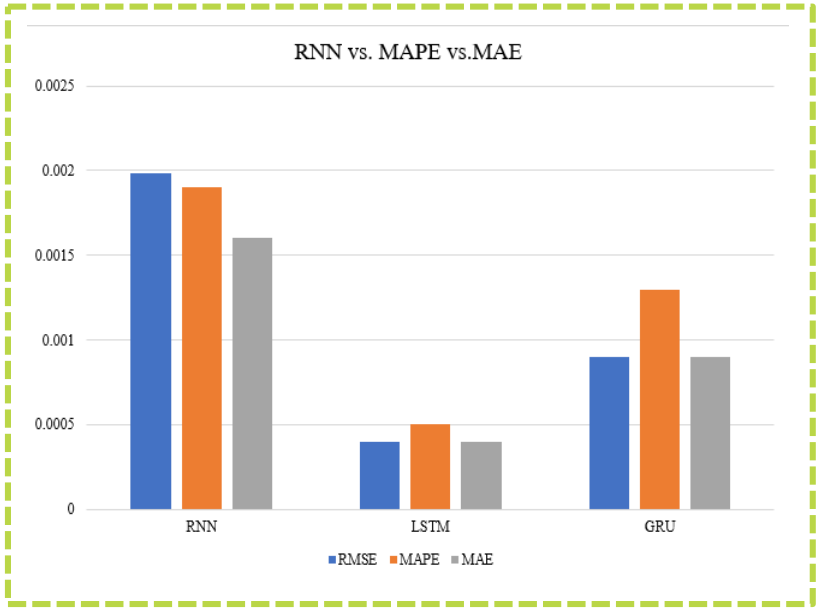
GRU



各演算法比較

經貝葉氏最佳化後之結果

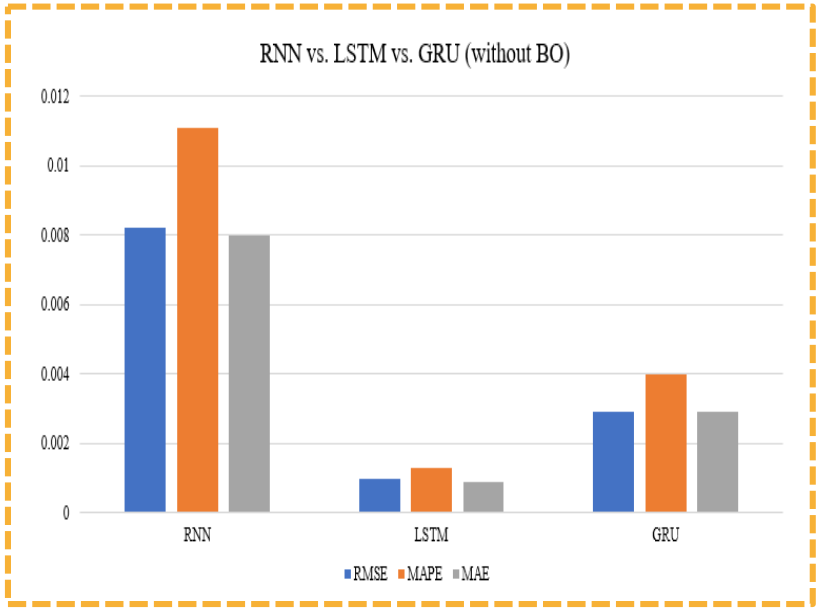
評估指標	RNN	LSTM	GRU
RMSE	0.0019	0.0004	0.0009
MAPE	0.19%	0.05%	0.13%
MAE	0.0016	0.0004	0.0009



各演算法比較

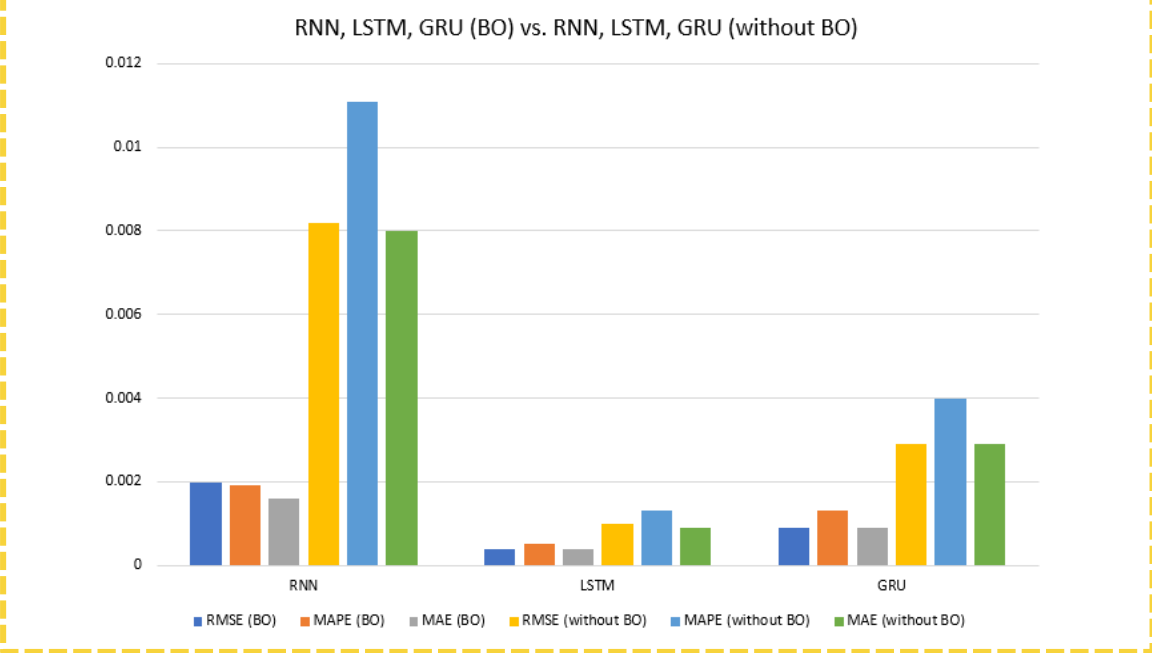
無貝葉氏最佳化後之結果

評估指標	RNN	LSTM	GRU
RMSE	0.0082	0.001	0.0029
MAPE	1.11%	0.13%	0.4%
MAE	0.008	0.0009	0.0029

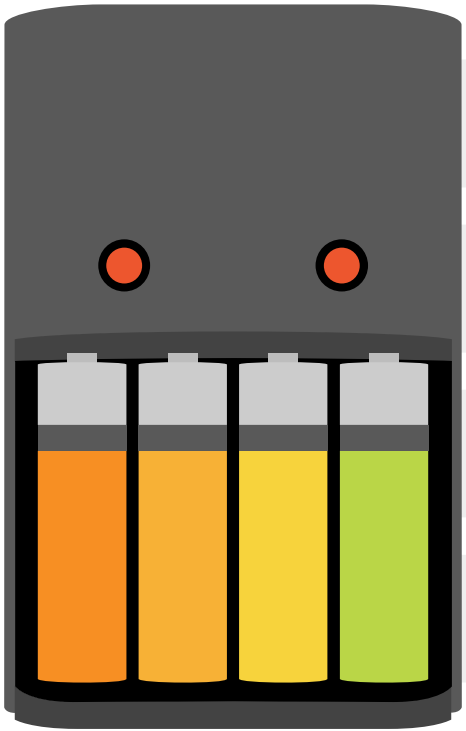


各演算法比較

貝葉氏最佳化與無貝葉氏最佳化後之結果



Discussion



Contributions	預測裡離子電池之健康状況，降低不必要的風險以及浪費
Limitations	由於受到電池循環次數之限制，樣本筆數較少，可能造成評估失準
Applicability	用於預防性保養 (Preventive Maintenance)，可於機台或設備停止之前，提前做出防護措施
Further Improvements	使用預測之 SOH 搭配實時數據來預測電池的剩餘壽命 (RUL)