


## 1단계(3차년도) 주요 결과물

(과제명) 대규모 분산 에너지 저장장치 인프라의 안전한 자율운영  
및 성능 평가를 위한 지능형 SW 프레임워크 개발  
(과제번호) 2021-0-00077

- 결과물명 : 전이학습을 완료한 성능 진단 알고리즘(SW)
- 작성일자 : 2023년 11월 14일

과학기술정보통신부 SW컴퓨팅산업원천기술개발사업  
“1단계(3차년도) 주요 결과물”로 제출합니다.

수행기관	성명/직위	확인
서울대학교	강명주/연구책임자	

정보통신기획평가원장 귀하

## <목차>

1. 개요 .....	1
가. 목적 .....	1
나. 범위 .....	1
2. 기법 .....	2
3. 소프트웨어 .....	2
가. 주요 기능 .....	2
나. 사용 환경 .....	4
다. 사용 방법 .....	4
라. 코드 .....	5

## 1. 개요

### 가. 목적

- 본 문서는 “대규모 분산 에너지 저장장치 인프라의 안전한 자율운영 및 성능 평가를 위한 지능형 SW 프레임워크 개발” 사업의 주요 결과물에 대한 보고서이다.
- 안전 AI 분석 엔진 “전이학습을 완료한 성능 진단 알고리즘(SW)” 중 “딥러닝 기반의 성능 진단 모델 개발 및 평가” 소프트웨어에 대한 설명을 한다.

[ 안전 AI 분석 엔진 SW ]

구분	1차년도	2차년도	3차년도	4차년도	5차년도
안전 AI 분석 엔진	물성 진단 지표 후보(SW)	실험 데이터로 학습한 이상 탐지 알고리즘(SW)	전이학습을 완료한 이상 탐지 알고리즘(SW)	최적화된 이상 탐지, 성능 진단 알고리즘(SW)	이상 탐지 알고리즘 및 사고 대응 알고리즘(SW)
	데이터 전처리 알고리즘(SW)	실험 데이터로 학습한 성능 진단 알고리즘(SW)	전이 학습을 완료한 성능 진단 알고리즘(SW)	개선된 이상 탐지, 성능 진단 알고리즘(SW)	상태 진단 알고리즘 및 효율적 운영 분석 결과(SW)

### 나. 범위

- 딥러닝 모델 학습 및 평가를 위한 데이터 전처리
- 공개 배터리 데이터(NASA) 사용
- 주요 데이터 칼럼 (전압, 전류, SoH) 활용

## 2. 기법

- 정상 배터리 데이터의 주요 칼럼인 SoH를 측정하기 위해 전처리 과정을 거친다.
- 시계열 데이터의 특성을 잘 잡아내는 1D-CNN 및 RNN 계열 딥러닝 모델을 활용하여 배터리 데이터의 숨겨진 패턴 및 정보를 활용한다.

## 3. 소프트웨어

### 가. 주요 기능

- 데이터 plot 기능

```
def cycle_plot(df, start_cyle, end_cycle):
    df['I'] = -df['I']

    # cycle numbering
    df['N'] = np.where(df['charge_type'] != df['charge_type'].shift(1), 1, 0)
    df.loc[0, 'N'] = 0
    df['N'] = df['N'].cumsum()

    ref = df.loc[(df['N'] >= start_cyle) & (df['N'] <= end_cycle)]
    first_t = ref['t'].iloc[0]
    _ref_t = ref['t'].values - first_t
    _ref_v = ref['V'].values
    _ref_i = ref['I'].values

    fig = plt.figure(figsize=(10, 5))
    plt.title(f'Original Data (V, I curves)', fontdict={'family': 'Sans-serif', 'color': 'black', 'weight': 'bold', 'size': 14})
    plt.axis('off')

    ax = fig.add_subplot(211)
    ax.scatter(_ref_t, _ref_v, s=5)
    ax.set_ylabel('Voltage (V)', labelpad=12, fontdict={'family': 'Sans-serif', 'color': 'purple', 'weight': 'bold', 'size': 14})
    ax.grid()

    ax = fig.add_subplot(212)
    ax.scatter(_ref_t, _ref_i, s=5)
    ax.set_xlabel('Time (sec)', fontdict={'family': 'Sans-serif', 'color': 'darkred', 'weight': 'bold', 'size': 14})
    ax.set_ylabel('Current (A)', fontdict={'family': 'Sans-serif', 'color': 'darkgreen', 'weight': 'bold', 'size': 14})
    ax.grid()

    plt.tight_layout()
    plt.show()

df = pd.read_parquet(os.path.join(DATA_ROOT, 'original', 'RW06.parquet'))

start_cycle = 30
end_cycle = 90
# RW06 reference cycle (ref A) : 0~2
# RW06 reference cycle (ref B) : 3~10, 862~869, 1651~1658, ...
# RW06 reference cycle (ref C) : 11~35, 870~894, 1659~1683, ...
# RW06 random walk cycle : 36~861, 896~1650, ...
```

## - 데이터 전처리

```

1  import os
2  import numpy as np
3  import pandas as pd
4  from scipy.interpolate import interp1d
5
6  import utils.config as config
7
8
9  def preprocessing(df):
10     # Modify current "I" (+/- change)
11     df['I'] = -df['I']
12
13     # Define time difference "dt"
14     df['dt'] = df['t'] - df['t'].shift(1)
15     df.loc[0, 'dt'] = 0
16     df.loc[df['dt'] > 50, 'dt'] = 0 # for some outliers
17
18     # Define Q difference "dQ"
19     df['dQ'] = df['I'] * df['dt']
20
21     # Define "relative Q"
22     df['relative Q'] = df['dQ'].cumsum()
23
24     # Define "Q(mAh)"
25     df['Q(mAh)'] = df['relative Q'] / 86400 * 1000
26
27     # Define the real Q "Q"
28     df['Q'] = 2100 + df['Q(mAh)'] # by the manual (README_RW_*.html)
29
30     # Define cycle number "N"
31     df['N'] = np.nan
32     temp = df.loc[df['CDR'] != df['CDR'].shift(1)].copy()
33     cycle_num = len(temp)
34     temp.loc[:, 'N'] = [i+1 for i in range(cycle_num)]
35     df.loc[temp.index, 'N'] = temp.loc[:, 'N']
36     df = df.fillna(method='ffill')
37     df['N'] = np.array(df['N'], dtype='int')
38
39     # Define reference cycle number "ref_N"
40     df['ref_N'] = np.nan
41     temp = df.loc[(df['charge_type'] == 'reference charge') | (df['charge_type'] == 'reference discharge')]
42     temp1 = temp.copy()
43     temp = temp[temp['charge_type'] != temp['charge_type'].shift(1)]
44
45     temp = temp[temp['charge_type'] != temp['charge_type'].shift(1)]
46     ref_num = len(temp)
47     temp.loc[:, 'ref_N'] = [i+1 for i in range(ref_num)]
48     temp1.loc[temp.index, 'ref_N'] = temp.loc[:, 'ref_N']
49     temp1 = temp1.fillna(method='ffill')
50     df.loc[:, 'ref_N'] = 0
51     df.loc[temp1.index, 'ref_N'] = temp1.loc[:, 'ref_N']
52     df['ref_N'] = np.array(df['ref_N'], dtype='int')
53
54     # define SOC
55     df['SOC'] = df['Q'] / df['Q'].max()
56
57     # define RW (random walk), cycle(from charge to next charge)
58     df['RW'] = False
59     temp = df.loc[df['charge_type'].str.contains('random')].copy()
60     df.loc[temp.index, 'RW'] = True
61
62     df['start_of_cycle'] = np.nan
63     sample_rw = df[(df['RW'] == True) & (df['charge_type'] != df['charge_type'].shift(1))].copy()
64     sample_ref = df[(df['RW'] == False) & (df['charge_type'] != df['charge_type'].shift(1))].copy()
65     temp1 = sample_ref[(sample_ref['CDR'] == 'C')].copy()
66     temp3 = sample_ref[(sample_ref['charge_type'].str.contains('pulse')) & ((sample_ref['charge_type'].shift(1).str.contains('reference discharge')))]
67     temp2 = sample_rw[(sample_rw['CDR'] == 'C')].copy()
68     df.loc[temp1.index, 'start_of_cycle'] = 1
69     df.loc[temp2.index, 'start_of_cycle'] = 1
70     df.loc[temp3.index, 'start_of_cycle'] = 1
71
72     temp = df[df['start_of_cycle'] == 1].copy()
73     df['cycle'] = np.nan
74     df.loc[temp.index, 'cycle'] = [i+1 for i in range(len(temp))]
75     df['cycle'].fillna(method='ffill', inplace=True)
76     df['cycle'].fillna(0, inplace=True)
77     df['cycle'] = np.array(df['cycle'], dtype='int')
78
79     # define ref_cycle and ref_type
80     df['ref_cycle'] = np.nan
81     ref_index = df[(df['RW'] == False) & (df['cycle'] != df['cycle'].shift(1))].copy()
82
83     df.loc[ref_index.index, 'ref_cycle'] = np.array(range(1, len(ref_index) + 1)) # reference_cycle's ref_cycle > 0
84     df['ref_cycle'].fillna(method='ffill', inplace=True)
85     df.loc[df['RW'] == True, 'ref_cycle'] = 0 # if cycle is not reference_cycle

```

```

81 df.loc[ref_index.index, 'ref_cycle'] = np.array(range(1, len(ref_index) + 1)) # reference_cycle's ref_cycle > 0
82 df['ref_cycle'].fillna(method = 'ffill', inplace = True)
83 df.loc[df['RW'] == True, 'ref_cycle'] = 0 # if cycle is not reference_cycle
84
85 df['ref_type'] = np.nan
86
87 temp = df.loc[df['ref_cycle'] > 0]
88 grouped = temp.groupby('cycle')
89 for i, ind in enumerate(list(set(temp['cycle']))) :
90     sample = grouped.get_group(ind)
91     for charge_type in list(set(sample['charge_type'].values)):
92         if 'pulse' in charge_type:
93             df.loc[sample.index, 'ref_type'] = 2 # if pulsed load is in this cycle
94             break
95         elif 'low' in charge_type:
96             df.loc[sample.index, 'ref_type'] = 0 # low current discharge is in this cycle
97             break
98         else:
99             df.loc[sample.index, 'ref_type'] = 1 # else, used to measure SOH, usually there are 2 cycles in one reference
100
101 # define SOH by Q with scipy's interp1d
102 df['SOH'] = np.nan
103 temp = df[(df['start_of_cycle'] == 1) & (df['ref_type'] == 1)]
104
105 SOH_list = []
106 done_list = []
107 for i in temp['cycle']:
108     if i-1 in done_list:
109         continue
110     Q_list = df[(df['cycle'] == i) & (df['CDR'] == 'D')]['Q'] # discharge.max - discharge.min
111     SOH_list.append(Q_list.max() - Q_list.min())
112     done_list.append(i)
113 f1 = interp1d(np.array(done_list), np.array(SOH_list), kind='linear', fill_value='extrapolate')
114 start_point = df[df['start_of_cycle'] == 1]
115
116 df.loc[start_point.index, 'SOH'] = f1(np.array(range(1, len(start_point) + 1)))
117
118 # Capacity added.
119 df['capacity'] = df['SOH']
120 df['capacity'].fillna(method = 'ffill', inplace = True)

```

```

112     done_list.append(i)
113 f1 = interp1d(np.array(done_list), np.array(SOH_list), kind='linear', fill_value='extrapolate')
114 start_point = df[df['start_of_cycle'] == 1]
115
116 df.loc[start_point.index, 'SOH'] = f1(np.array(range(1, len(start_point) + 1)))
117
118 # Capacity added.
119 df['capacity'] = df['SOH']
120 df['capacity'].fillna(method = 'ffill', inplace = True)
121
122 df['SOH'] = df['SOH']/df['SOH'].max()
123 df['SOH'].fillna(method = 'ffill', inplace = True)
124
125 temp = df[(df['ref_cycle'] > 0) & (df['RW'].shift(1) != False)].copy()
126 df['group'] = np.nan
127 df.loc[temp.index, 'group'] = [i+1 for i in range(len(temp))]
128 df['group'].fillna(method = 'ffill', inplace = True)
129 df['group'] = np.array(df['group'], dtype = 'int')
130
131
132 return df
133
134
135
136 if __name__ == "__main__":
137     raw_data_dir = config.RAW_DATA_DIR
138     processed_data_dir = config.PROCESSED_DATA_DIR
139
140     if not os.path.exists(processed_data_dir):
141         os.mkdir(processed_data_dir)
142
143     for battery_id in config.BATTERY_NAME:
144         try:
145             battery_data = pd.read_parquet(os.path.join(raw_data_dir, battery_id))
146             processed_data = preprocessing(battery_data)
147             processed_data.to_parquet(os.path.join(processed_data_dir, battery_id))
148         except Exception as e:
149             print("Error occurs at", battery_id, "dataset.")

```

## 나. 사용 환경

- 본 프로그램은 python 코드로 구현되었음
- Python이 설치된 어떠한 OS 환경에서도 사용이 가능함
- 코드의 구동에는 python의 OS, numpy, pandas 등의 라이브러리가 필요함

## 다. 사용 방법

- 입력과 출력 데이터는 parquet 형식을 사용함
- raw 데이터를 넣으면 전처리된 preprocessed 데이터가 저장됨.
- 전체 코드는 노트북 파일 형식(.ipynb)으로 데이터 파일을 지정한 후 일련의 과정을 따라 이상치를 합성할 수 있음
- 월 단위 데이터를 입력하면 모델 학습을 위한 학습 데이터(training data)와 평가 데이터(test data)를 나누고, 평가 데이터에 이상치를 합성하여 저장함
- 데이터 정보로 완전 충전이 된 날(통상적인 맑은 날)의 구분과 데이터 충방전 구간이 기록됨

## 라. 코드

OpenESS/SNU\_AI/state\_estimation\_nasa