Machine Learning Lab Report

RNN-flavored Ensembling to Predict Remaining Useful Life of Lithium-ion Batteries

Group 8

Tushar Bauskar	191080010
Utsav Khatu	191080038
Pankaj Khushalani	191080040
Ravi Maurya	191080080

Machine Learning Lab (R4IT3011P)

Semester VI, B.Tech. Information Technology

CE & IT Department,

VJTI, Matunga

Mumbai - 19

Veermata Jijabai Technological Institute,

H.R. Mahajani Road

Matunga, Mumbai 19

(This page is left blank intentionally)

Table of Contents

1. Problem Statement

With its use seen in critical areas of safety and security, it is essential for lithium-ion batteries to be reliable. Prediction of the Remaining Useful Life (RUL) can give insights into the health of the battery. Variations of Recurrent Neural Networks (RNN) are employed to learn the capacity degradation trajectories of lithium-ion batteries. Using several regressor models as the baseline, an ensemble of RNNs is created to overcome the shortcomings of one RNN over the other. The critical point approach and the data-driven approach for regressor models and neural network models respectively help predict the RUL.

2. Dataset

The dataset has been collected from a custom-built battery prognostics tested at the **NASA Ames Prognostics Center of Excellence**. The Lithium-ion batteries were run through 2 different operational profiles (charging and discharging) at different temperatures. The experiments were stopped when the batteries reached the end-of-life (EOL) criteria of 30% fade in rated capacity (from 2 Ah to 1.4 Ah)

2.1. Dataset Structure

Charge

1. **Voltage_measured**: Battery terminal voltage (Volts)
2. **Current_measured**: Battery output current (Amps)
3. **Temperature_measured**: Battery temperature (degree C)
4. **Current_charge**: Current measured at charger (Amps)
5. **Voltage_charge**: Voltage measured at charger (Volts)
6. **Time**: Time vector for the cycle (secs)

Discharge

1. **Voltage_measured**: Battery terminal voltage (Volts)
2. **Current_measured**: Battery output current (Amps)
3. **Temperature_measured**: Battery temperature (degree C)
4. **Current_charge**: Current measured at load (Amps)
5. **Voltage_charge**: Voltage measured at load (Volts)
6. **Time**: Time vector for the cycle (secs)
7. **Capacity**: Battery capacity (Ahr) for discharge till 2.7V

Impedance

1. **Sense_current**: Current in sense branch (Amps)
2. **Battery_current**: Current in battery branch (Amps)
3. **Current_ratio**: Ratio of the above currents
4. **Battery_impedance**: Battery impedance (Ohms) computed from raw data
5. **Rectified_impedance**: Calibrated and smoothed battery impedance (Ohms)
6. **Re**: Estimated electrolyte resistance (Ohms)
7. **Rct**: Estimated charge transfer resistance (Ohms)

3. Motivation

With the increasing importance of Electric Vehicles (EVs) in the automotive industry, research in EV batteries made of Lithium-ion (Li-ion) has gained momentum. As Li-ion batteries are a core component of EVs, their safety and reliability is a critical concerns for the functioning of EVs.

Prognostics and health management (PHM) is a discipline composed of methods and technologies to evaluate system reliability and safety under actual life cycle conditions to predict fault progression. The design of an appropriate Battery Management System (BMS) is crucial to reducing costs and increasing vehicle efficiency and security. One of the major tasks of the BMS is to evaluate the current health conditions of the battery as they degrade over time. Predominant indicators are battery capacity and internal resistance, which inform us about the battery residual energy and power capabilities respectively, indicated by the State of Health (SOH). The SOH and the Remaining Useful Life (RUL) are the most crucial parameters of battery health that must be estimated by the BMS [1].

RUL is defined as the number of cycles remaining from the present cycle to the end-of-life (EOL) which can be chosen as 70–80% of the nominal capacity. An RUL prediction can be described by the probability distribution function (PDF) of the RUL to show the uncertainty of battery failure [2].

With the time-series nature of the charging and discharging cycles of a lithium-ion battery, the intuition is to make use of models that can glance at the data points before and after the current data point, i.e., to make use of Recurrent Neural Networks (RNN).

Due to some shortcomings of RNNS such as Long Short-Term Memory (LSTM) such as being slower and being sensitive to different random weight initializations, several RNN models can be employed and an ensemble can be created to make better predictions.

4. Methodology

The most popular approach for time series analysis is using Recurrent Neural Networks and their various forms. **Long Short-Term Memory (LSTM)** networks are peculiar RNNs that are able to handle long-term sequences and have become the baseline of RNNs. LSTMs therefore can be used for this problem statement.

4.1. Exploratory Data Analysis

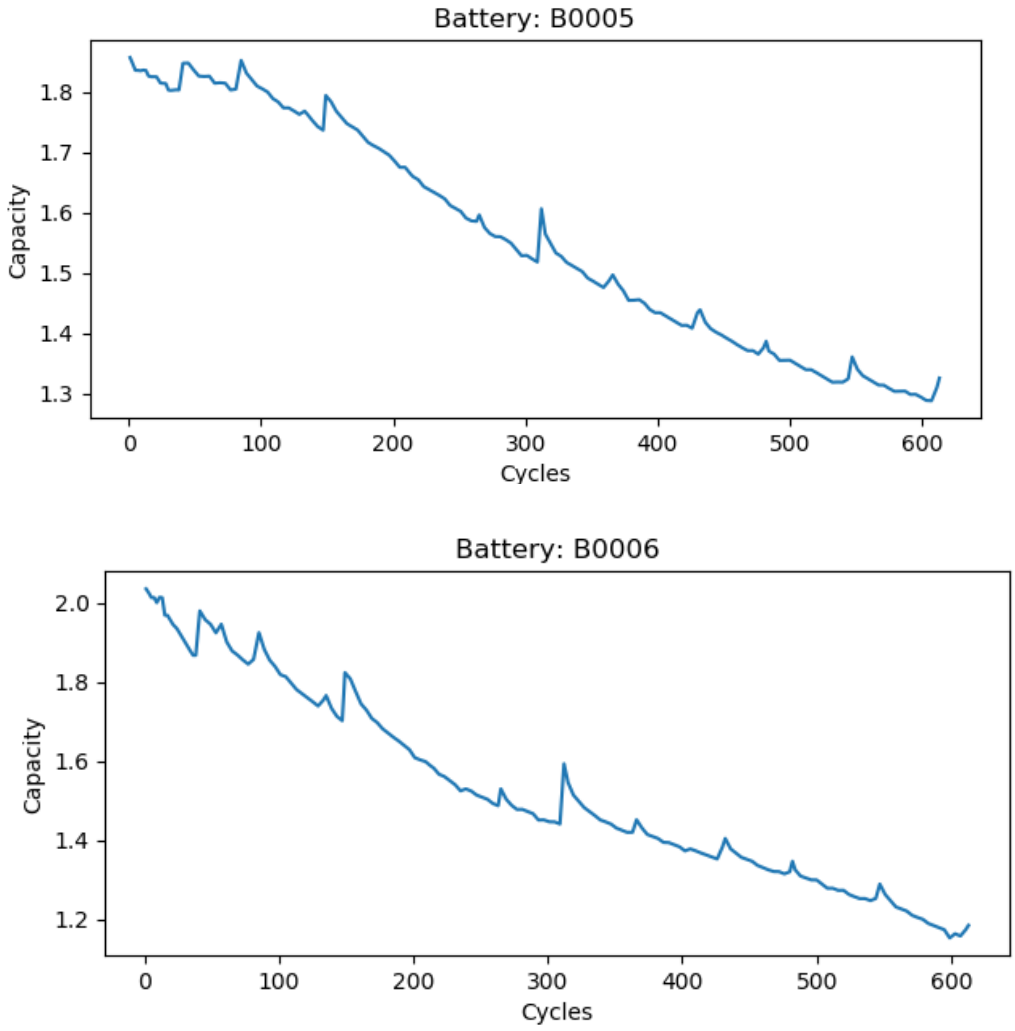
4.1.1. Division of dataset to experiments

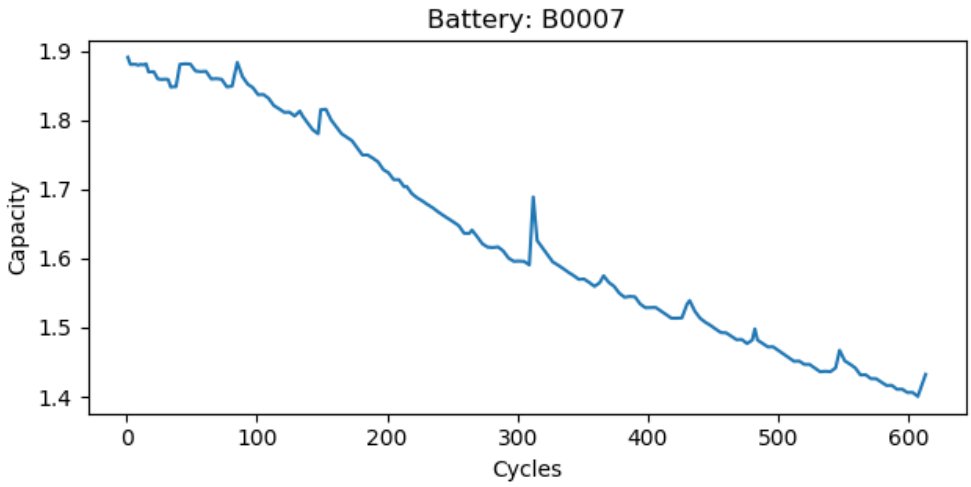
Based on the given metadata of every battery, multiple sets of lithium-ion batteries have been taken under varying experimental conditions. Each of these sets has been identified and the dataset is further divided into 9 individual experiments.

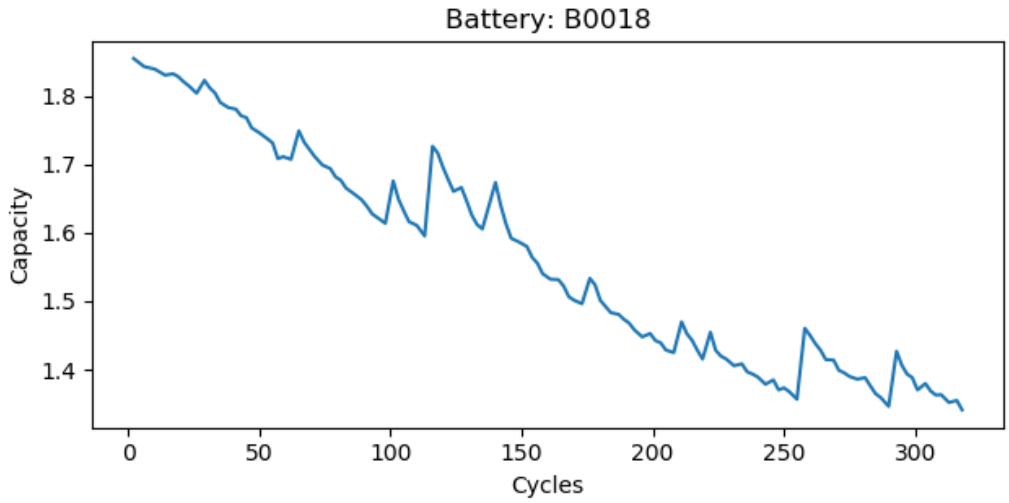
- Experiment 1 - B0005, B0006, B0007, B0018
- Experiment 2 - B0025, B0026, B0027, B0028
- Experiment 3 - B0029, B0030, B0031, B0032
- Experiment 4 - B0033, B0034, "B0036
- Experiment 5 - B0038, B0039, B0040
- Experiment 6 - B0041, B0042, B0043, B0044
- Experiment 7 - B0045, B0046, B0047, B0048
- Experiment 8 - B0049, B0050, B0051, B0052
- Experiment 9 - B0053, B0054, B0055, B0056

Plots of various attributes of the dataset with respect to cycles of different batteries belonging to experiment 1 were created.

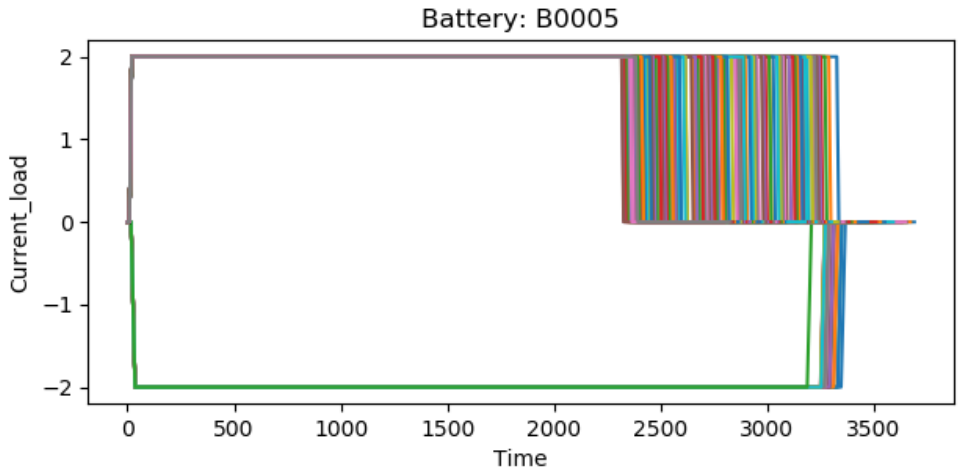
4.1.2. Capacity

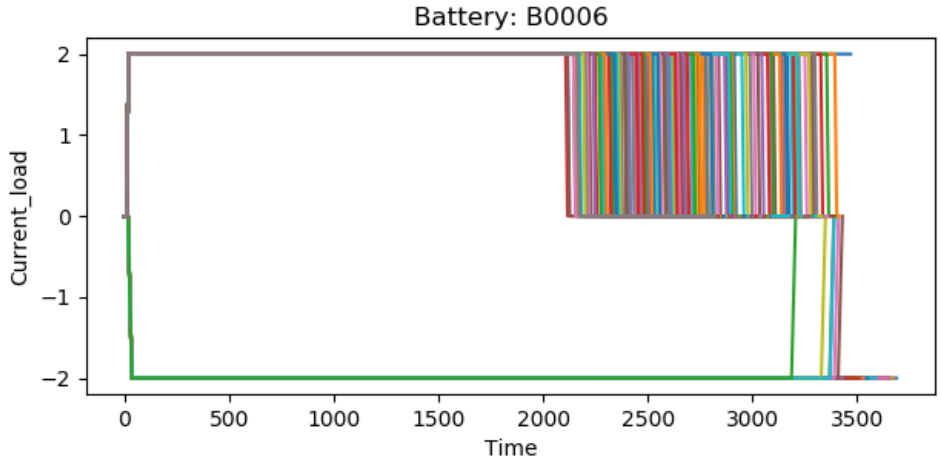


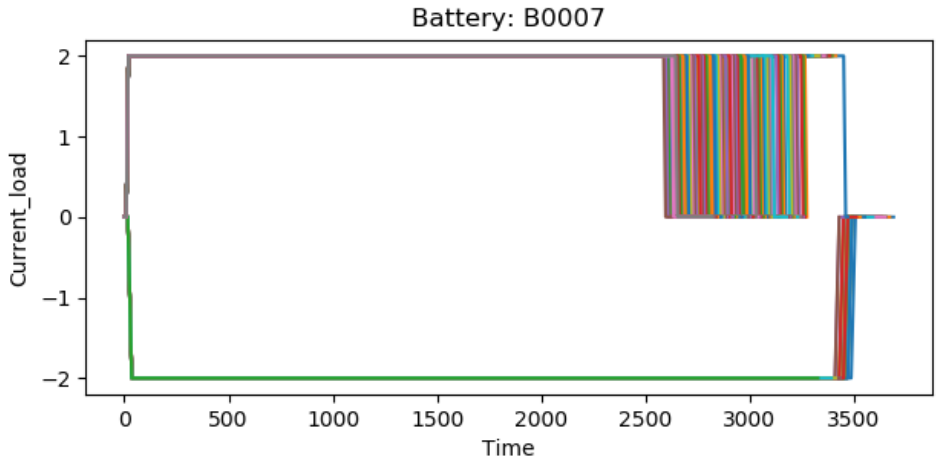

Battery: B0007
Capacity
Cycles
1.9
1.8
1.7
1.6
1.5
1.4
0
100
200
300
400
500
600

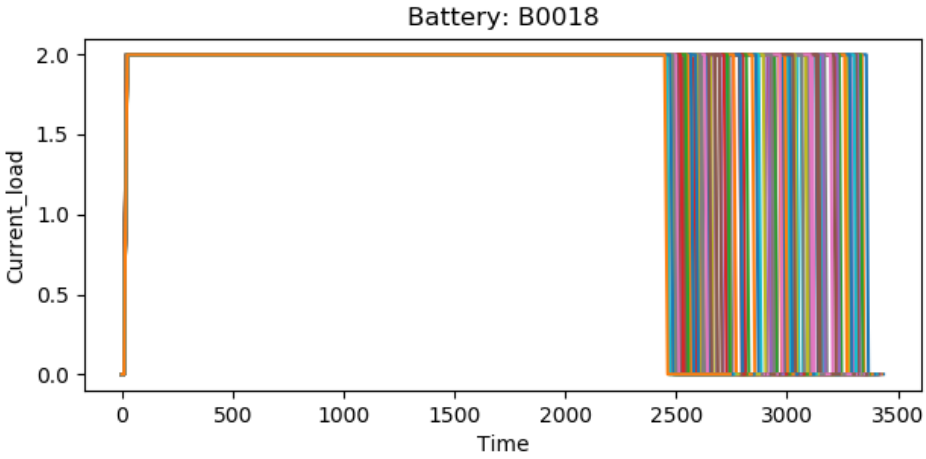

Battery: B0018
Capacity
Cycles
1.8
1.7
1.6
1.5
1.4
0
50
100
150
200
250
300

4.1.3. Current Load

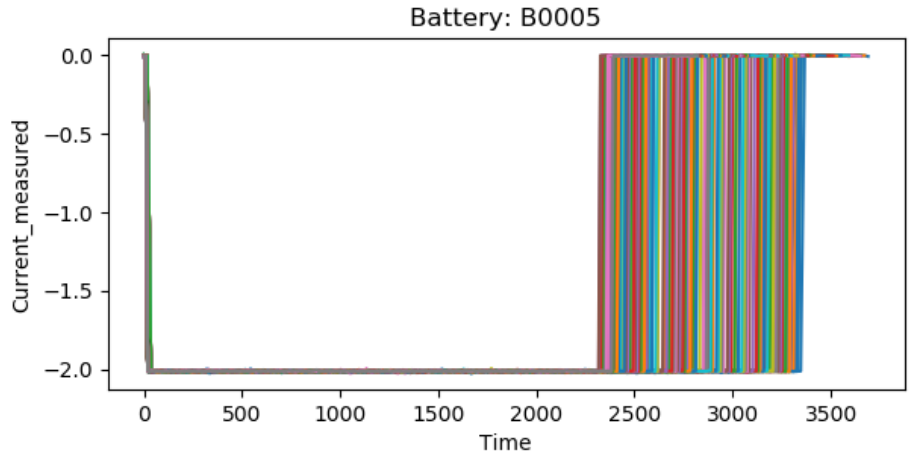


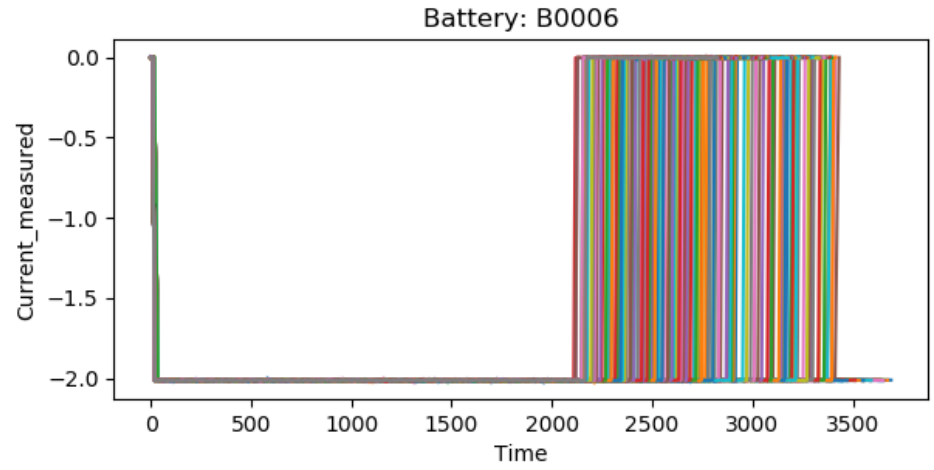


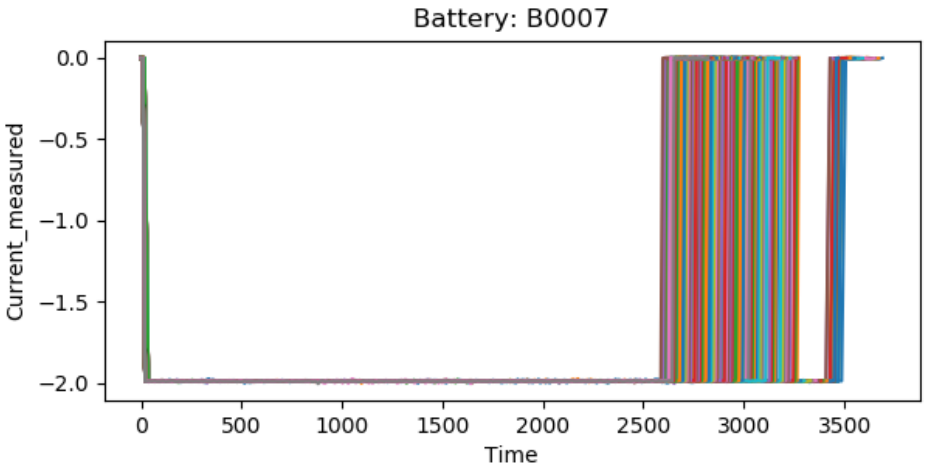

Battery: B0007
Current_load
Time
2
1
0
−1
−2
0
500
1000
1500
2000
2500
3000
3500

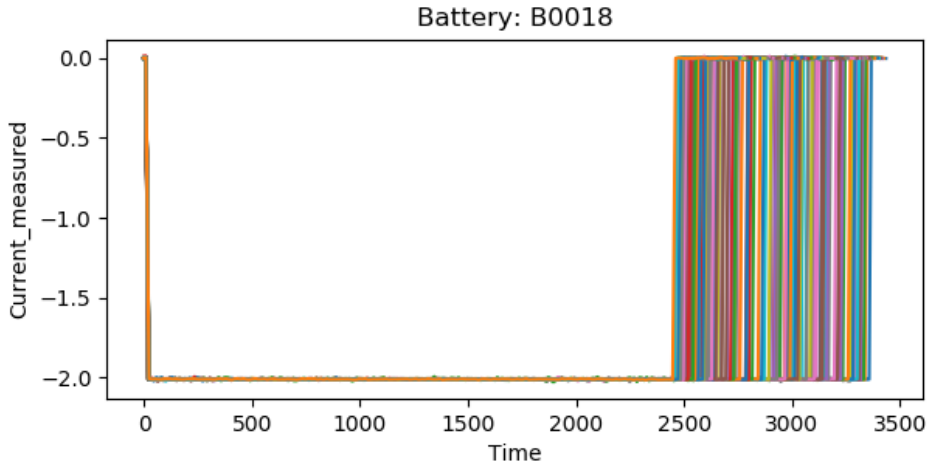

Battery: B0018
Current_load
Time
2.0
1.5
1.0
0.5
0.0
0
500
1000
1500
2000
2500
3000
3500

4.1.4. Current measured

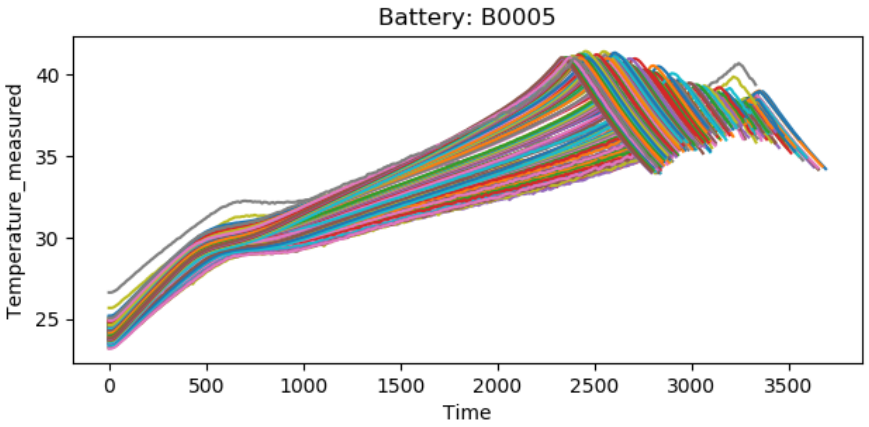


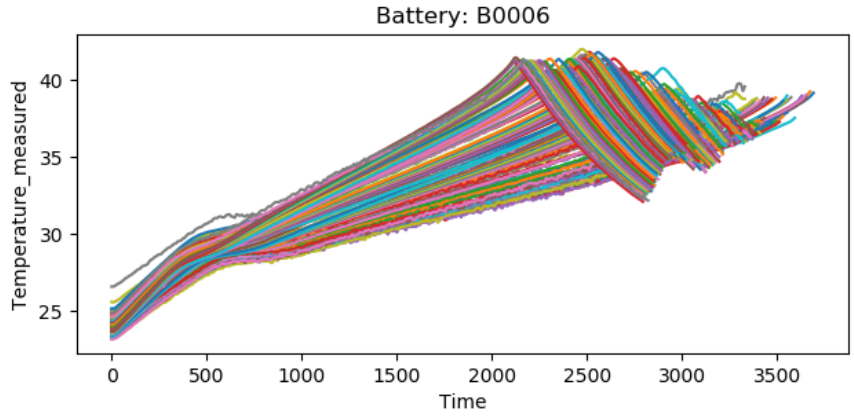


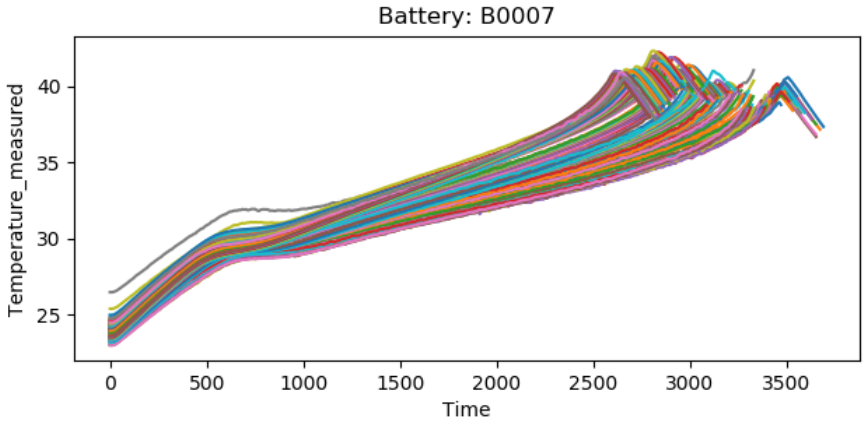

Battery: B0007
Current_measured
0.0
−0.5
−1.0
−1.5
−2.0
0
500
1000
1500
2000
2500
3000
3500
Time

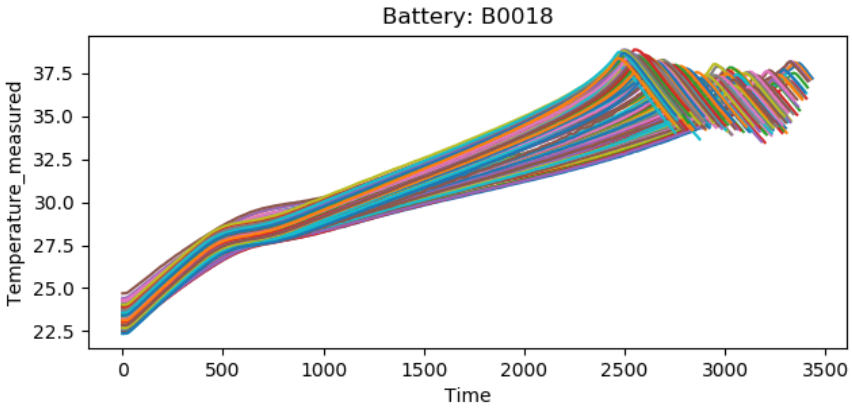

Battery: B0018
Current_measured
0.0
−0.5
−1.0
−1.5
−2.0
0
500
1000
1500
2000
2500
3000
3500
Time

4.1.5. Temperature measured

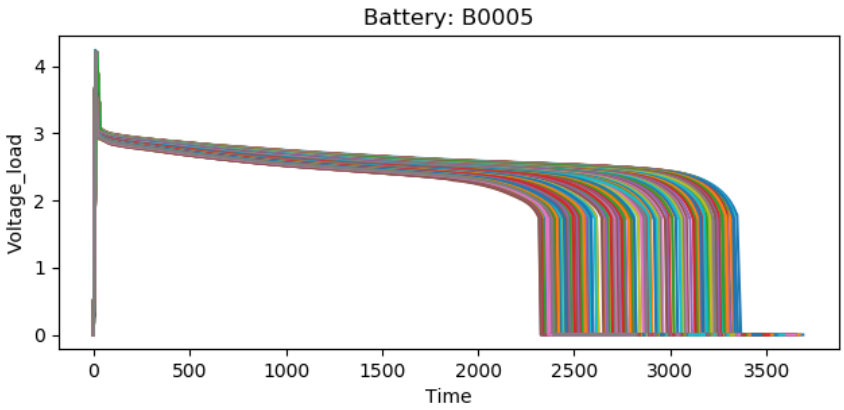


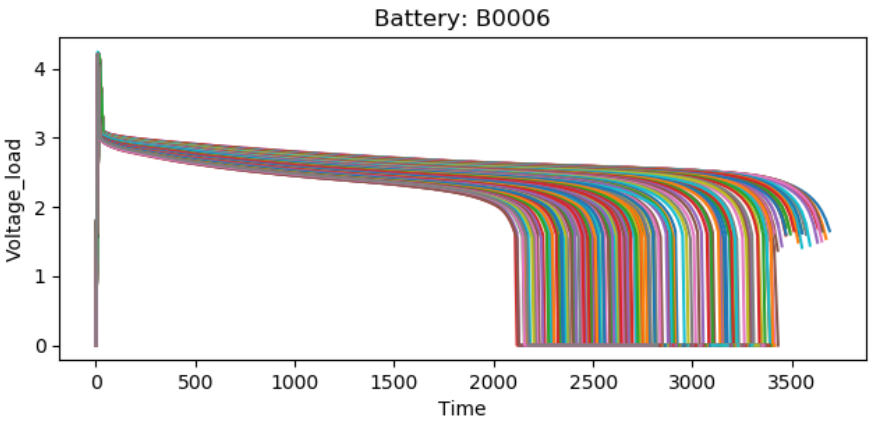


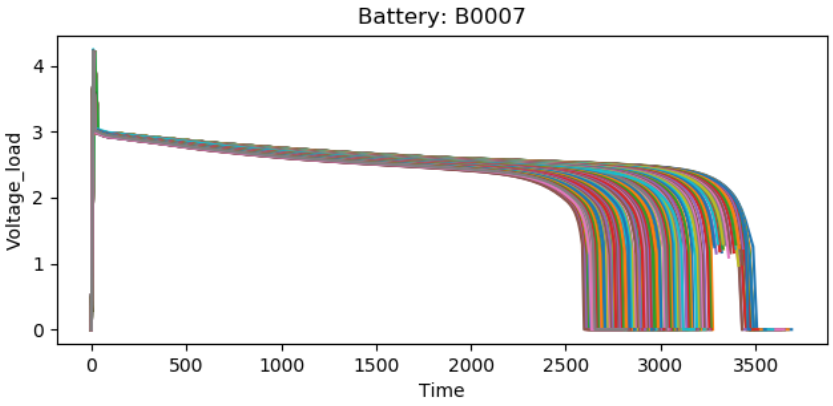


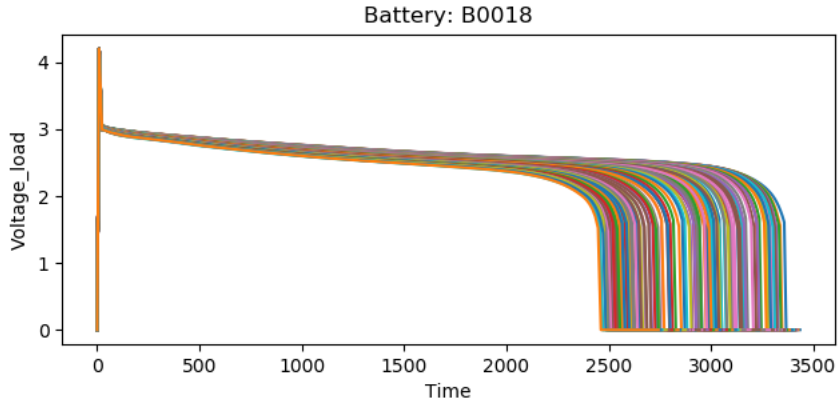


4.1.6. Voltage Load

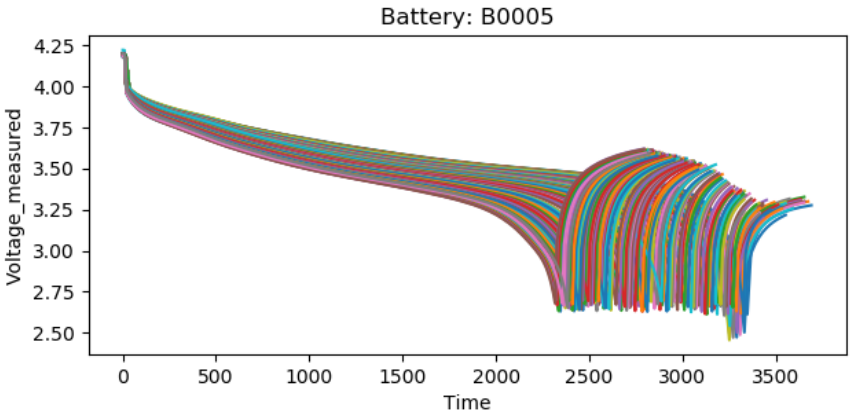


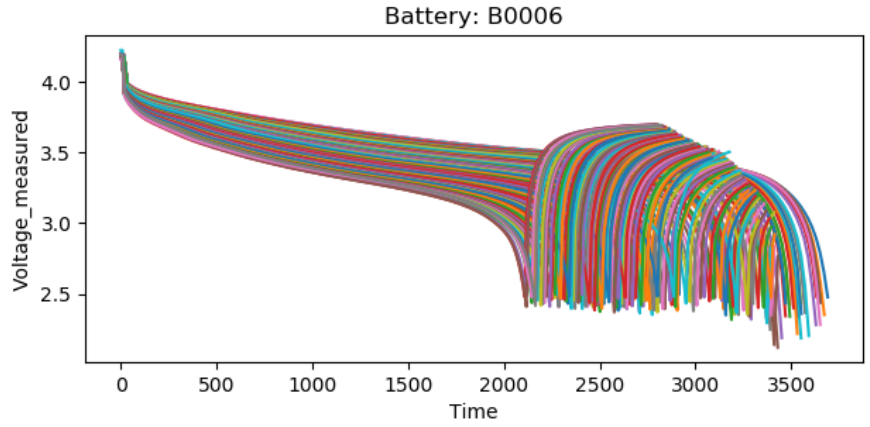


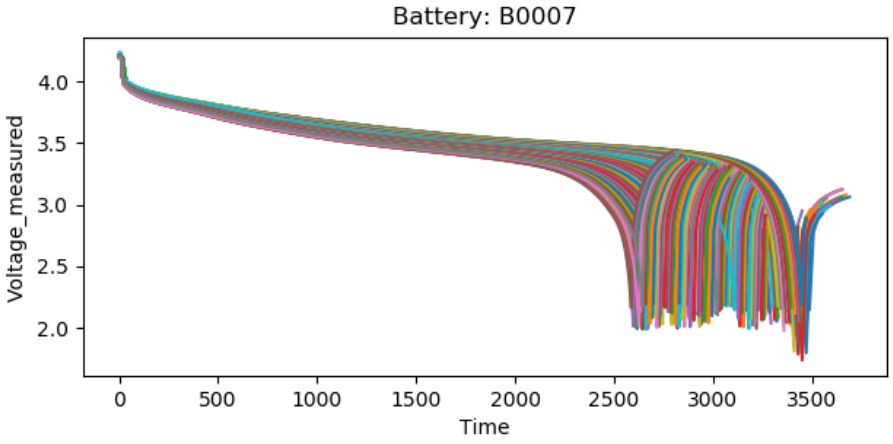


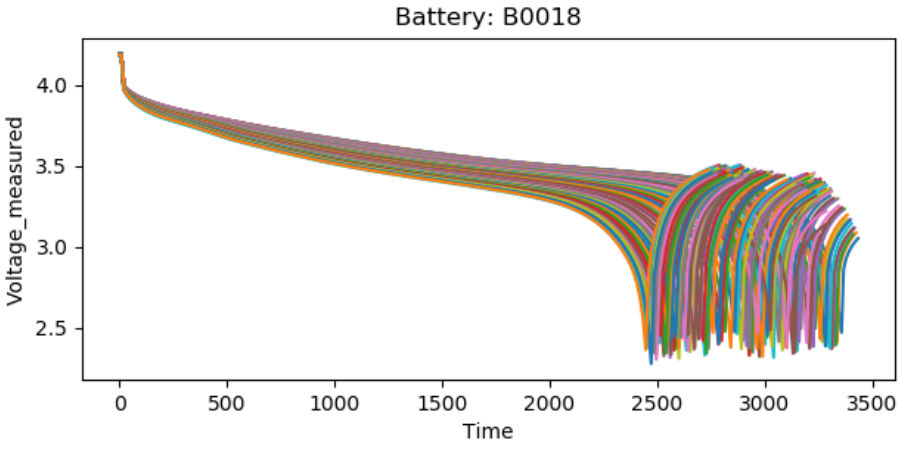


4.1.7. Voltage measured




Battery: B0006
Voltage_measured
Time
4.0
3.5
3.0
2.5
0
500
1000
1500
2000
2500
3000
3500


Battery: B0007
Voltage_measured
Time
4.0
3.5
3.0
2.5
2.0
0
500
1000
1500
2000
2500
3000
3500


Battery: B0018
Voltage_measured
Time
4.0
3.5
3.0
2.5
0
500
1000
1500
2000
2500
3000
3500

4.2. Model Selection

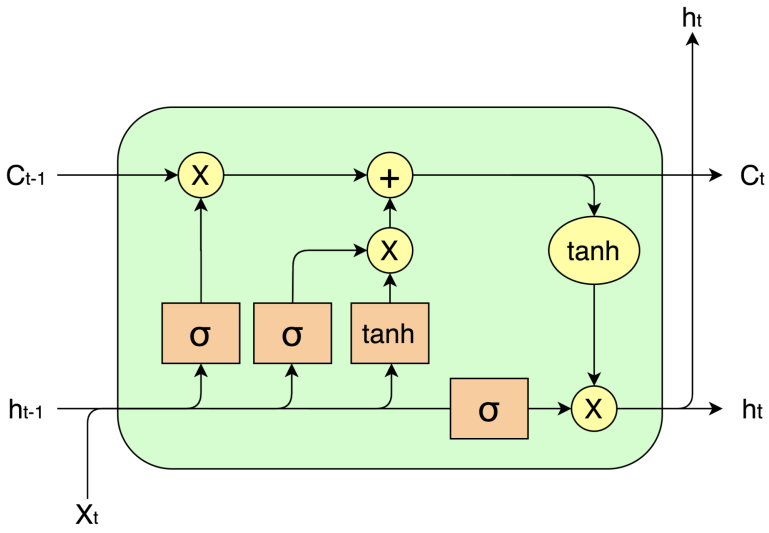
For creating the prediction model, four different variations of RNN were used:

1. LSTM
2. BiLSTM
3. GRU
4. BiGRU

4.2.1. LSTM

It is a special kind of recurrent neural network that is capable of learning long-term dependencies in data. This is achieved because the recurring module of the model has a combination of four layers interacting with each other.

An LSTM module has a cell state and three gates which provides them with the power to selectively learn, unlearn or retain information from each of the units. The cell state in LSTM helps the information to flow through the units without being altered by allowing only a few linear interactions. Each unit has an input, output and a forget gate which can add or remove the information to the cell state. The forget gate decides which information from the previous cell state should be forgotten for which it uses a sigmoid function. The input gate controls the information flow to the current cell state using a pointwise multiplication operation of ‘sigmoid’ and ‘tanh’ respectively. Finally, the output gate decides which information should be passed on to the next hidden state

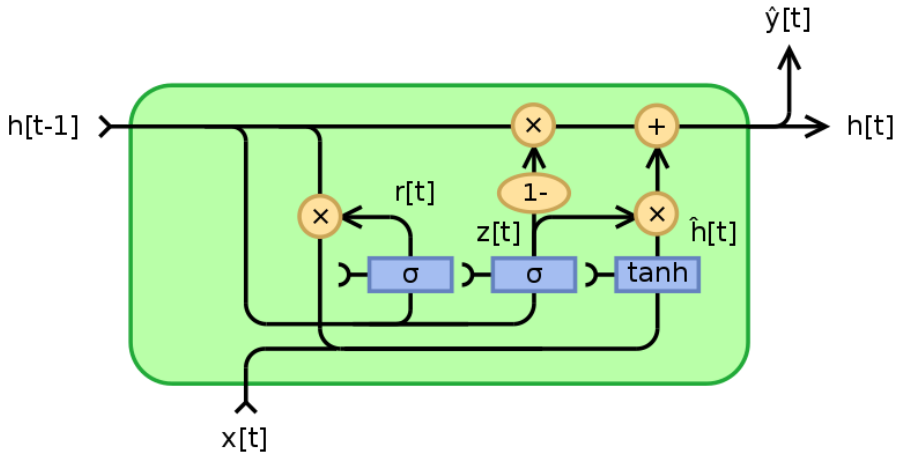


4.2.2. BiLSTM

A Bidirectional LSTM, or biLSTM, is a sequence processing model that consists of two LSTMs: one taking the input in a forward direction, and the other in a backward direction. BiLSTMs effectively increase the amount of information available to the network, improving the content available to the algorithm

4.2.3. GRU

GRUs are very similar to Long Short Term Memory(LSTM). Just like LSTM, GRU uses gates to control the flow of information. They are relatively new as compared to LSTM. This is the reason they offer some improvement over LSTM and have simpler architecture. Unlike LSTMs, it does not have a separate cell state (Ct). It only has a hidden state(Ht). Due to the simpler architecture, GRUs are faster to train.



4.2.4. BiGRU

A Bidirectional GRU, or BiGRU, is a sequence processing model that consists of two GRUs. one taking the input in a forward direction, and the other in a backwards direction. It is a bidirectional recurrent neural network with only the input and forget gates.

4.3. Model Training

Using the above mentioned RNN variation 4 models are trained for each experiment. The input consists of following attributes

1. **Voltage_measured**
2. **Current_measured**
3. **Temperature_measured**
4. **Current_load**
5. **Voltage_load**
6. **Time**

And output is the **Capacity** column

The models for experiment 1 are as follows:

4.3.1. LSTM

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 masking (Masking)           (None, 6, 371)            0
 lstm (LSTM)                 (None, 6, 256)            643072
 lstm_1 (LSTM)               (None, 6, 256)            525312
 lstm_2 (LSTM)               (None, 6, 128)            197120
 lstm_3 (LSTM)               (None, 6, 128)            131584
 lstm_4 (LSTM)               (None, 6, 64)             49408
 lstm_5 (LSTM)               (None, 64)                33024
 dense (Dense)               (None, 64)                4160
 dense_1 (Dense)             (None, 64)                4160
 dense_2 (Dense)             (None, 32)                2080
 dense_3 (Dense)             (None, 32)                1056
 dense_4 (Dense)             (None, 1)                 33

=================================================================
Total params: 1,591,009
Trainable params: 1,591,009
Non-trainable params: 0
```

4.3.2. BiLSTM

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 masking_5 (Masking)         (None, 6, 371)            0
 bidirectional (Bidirectiona  (None, 6, 512)           1286144
 l)
 bidirectional_1 (Bidirectio  (None, 6, 512)           1574912
 nal)
 bidirectional_2 (Bidirectio  (None, 6, 256)           656384
 nal)
 bidirectional_3 (Bidirectio  (None, 6, 256)           394240
 nal)
 bidirectional_4 (Bidirectio  (None, 6, 128)           164352
 nal)
 bidirectional_5 (Bidirectio  (None, 128)              98816
 nal)
 dense_23 (Dense)            (None, 64)                8256
 dense_24 (Dense)            (None, 64)                4160
 dense_25 (Dense)            (None, 32)                2080
 dense_26 (Dense)            (None, 32)                1056
 dense_27 (Dense)            (None, 1)                 33
=================================================================
```

```
Total params: 4,190,433

Trainable params: 4,190,433

Non-trainable params: 0

_________________________________________________________________
```

4.3.3. GRU

```
Model: "sequential_2"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 masking_2 (Masking)         (None, 6, 371)            0

 gru_6 (GRU)                 (None, 6, 256)            483072

 gru_7 (GRU)                 (None, 6, 256)            394752

 gru_8 (GRU)                 (None, 6, 128)            148224

 gru_9 (GRU)                 (None, 6, 128)            99072

 gru_10 (GRU)                (None, 6, 64)             37248

 gru_11 (GRU)                (None, 64)                24960

 dense_8 (Dense)             (None, 64)                4160

 dense_9 (Dense)             (None, 64)                4160

 dense_10 (Dense)            (None, 32)                2080

 dense_11 (Dense)            (None, 32)                1056

 dense_12 (Dense)            (None, 1)                 33

=================================================================
```

```
Total params: 1,198,817
Trainable params: 1,198,817
Non-trainable params: 0
_________________________________________________________________
```

4.3.4. BiGRU

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 masking_6 (Masking)         (None, 6, 371)            0
 bidirectional_35 (Bidirecti  (None, 6, 512)           966144
 onal)
 bidirectional_36 (Bidirecti  (None, 6, 512)           1182720
 onal)
 bidirectional_37 (Bidirecti  (None, 6, 256)           493056
 onal)
 bidirectional_38 (Bidirecti  (None, 6, 256)           296448
 onal)
 bidirectional_39 (Bidirecti  (None, 6, 128)           123648
 onal)
 bidirectional_40 (Bidirecti  (None, 128)              74496
 onal)
```

```
 dense_30 (Dense)            (None, 64)                8256

 dense_31 (Dense)            (None, 64)                4160

 dense_32 (Dense)            (None, 32)                2080

 dense_33 (Dense)            (None, 32)                1056

 dense_34 (Dense)            (None, 1)                 33

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

Total params: 3,152,097

Trainable params: 3,152,097

Non-trainable params: 0

_________________________________________________________________
```

4.4. RNN Models' Results

The aforementioned four variations of RNN were trained on all 9 experiments and the following results were obtained:

Experiment	Model	Training RMSE	Testing RMSE	Validation RMSE
Experiment 1	LSTM	0.0312	0.0304	0.0311
	BiLSTM	0.2870	0.2792	0.3259
	GRU	0.0278	0.0342	0.0356
	BiGRU	0.0901	0.0945	0.1059
Experiment 2	LSTM	0.0190	0.0173	0.0122
	BiLSTM	0.5521	0.1376	0.4871
	GRU	0.0962	0.0868	0.1957
	BiGRU	1.5680	1.3482	1.7741
Experiment 3	LSTM	0.0183	0.0336	0.0542
	BiLSTM	0.1070	0.1108	0.1237
	GRU	0.0290	0.0425	0.0516
	BiGRU	0.0340	0.0454	0.0530
Experiment 4	LSTM	0.0248	0.0236	0.0232
	BiLSTM	0.2583	0.2232	0.1943
	GRU	0.0152	0.0242	0.0452
	BiGRU	0.2186	0.2282	0.1775
Experiment 5	LSTM	0.0145	0.0981	0.1329
	BiLSTM	1.2602	1.1037	0.9943
	GRU	0.0253	0.0811	0.1761
	BiGRU	0.3437	0.4544	0.4535

Experiment 6	LSTM	0.0123	0.0189	0.0277
	BiLSTM	0.7338	0.6262	0.6111
	GRU	0.0967	0.1094	0.2436
	BiGRU	0.1380	0.2562	0.2627
Experiment 7	LSTM	0.0132	0.0253	0.0245
	BiLSTM	0.2486	0.3564	0.3278
	GRU	0.0578	0.0645	0.0689
	BiGRU	0.1896	0.2486	0.2156
Experiment 8	LSTM	0.0226	0.0356	0.0312
	BiLSTM	0.2270	0.2792	0.4123
	GRU	0.0156	0.0236	0.0384
	BiGRU	0.0689	0.1562	0.1047
Experiment 9	LSTM	0.0196	0.0265	0.0241
	BiLSTM	0.3568	0.3956	0.4256
	GRU	0.0452	0.0546	0.0514
	BiGRU	0.0918	0.1256	0.1298

5. Pseudocode with Output

5.1. Preprocessing

```
bs = list of battery

params = [ 'Voltage_measured', 'Current_measured', 'Temperature_measured',
'Current_load', 'Voltage_load', 'Time', 'Capacity' ]
```

5.1.1. Loading and reading mat file

1. Read .mat file using loadmat() function
2. Access the necessary data and divide the data into 3 section
    a. Types : (Charge, Discharge, Impedance)
    b. Times :
    c. Datas : (Contains all the time series data related to the charge-discharge cycle)
3. For row in dataset:

```
        types.append(row[type])

        times.append(row[time])

        datas.append(row[data])
```

5.1.2. Creating the dataset

1. Batteries = []
2. Cycles = {}
3. For i in range(len(bs)):

```
        For each parameter:

            Cycles[bs[i]][parameter] = []
```

```
                For value in datas[i]:

                    Cycles[bs[i]][parameter].append(value)
```

4. Save cycles as dataframe df

5.1.3. Creating dataset

1. df_x = df.drop([‘Capacity’])
2. df_y = df[‘Capacity’]
3. Split the dataset into training (train_x, train_y), validation (val_x, val_y) and testing set (test_x, test_y) using **train_test_split()** function twice

5.2. Model training

5.2.1. Create and train RNN models

1. Initialize optimizer, loss and metrics, NUM_EPOCHS and validation split
2. Create model
3. model.compile(optimizer, loss='huber', metrics=['mse', 'mae', 'mape', tf.keras.metrics.RootMeanSquaredError(name='rmse')])
4. Add LSTM/GRU/Bidirectional layers as required
5. model.fit(train_x, train_y,epochs=NUM_EPOCHS, batch_size=BATCH_SIZE,verbose=1, validation_split=0.1,callbacks=[early_stopping,lr_scheduler])
6. Save model

5.2.2. Creating the baseline model

1. Time at which the highest temperature is measured

```
def get_temp_critical(temp_measured, timestamps):
      highest_temp = 0
      critical_point = 0
```

```
        for timestamp in temp_measured:
                if temp_measured[timestamp] > highest_temp:
                        highest_temp = temp_measured[timestamp]
                        critical_point = timestamps[timestamp]
        return critical_point
```

2. Time at which the lowest voltage has been measured

```
def get_voltage_critical(voltage_measured, timestamps):
        lowest_voltage = 0
        critical_point = 0
        for timestamp in voltage_measured:
                if voltage_measured[timestamp] < lowest_voltage:
                        lowest_voltage = voltage_measured[timestamp]
                        critical_point = timestamps[timestamp]
        return critical_point
```

3. The first instance at which voltage drops below 1 volt after 1500 seconds

```
def get_voltage_load_critical(voltage_load_measured, timestamps):
        for timestamp in voltage_load_measured:
                if timestamps[timestamp] > 1500 and
voltage_load_measured[timestamp] < 1:
                        return timestamps[timestamp]
        return -1
```

5.2.3. Creating the ensemble model

1. In total 36 Models will be trained, 4 Models (LSTM,GRU,BILSTM,BIGRU) each for 9 experiments.
2. Ensembling 4 Models for each 9 experiments using various ML approaches. Notable are mentioned below
   a. LinearRegression
   b. HuberRegressor

c. KNeighborsRegressor
d. LinearSVR
e. NuSVR
f. SVR
g. DecisionTreeRegressor
h. ExtraTreeRegressor
i. RandomForestRegressor
j. ExtraTreesRegressor
k. XGBRegressor
l. LGBMRegressor
m. CatBoostRegressor

```
def get_scores(y_true, y_pred):
      return np.sqrt(mean_squared_error(y_true,y_pred))

for exp in exps:
      df_x, df_y = ensemble_df[exp]
      train_x, test_x, train_y, test_y = train_test_split(df_x, df_y,
test_size=0.2, random_state=0)
      test_x, val_x,test_y, val_y = train_test_split(test_x, test_y,
test_size=0.5, random_state=0)

      algos = (LinearRegression, HuberRegressor, KNeighborsRegressor
,LinearSVR, NuSVR, SVR, DecisionTreeRegressor, ExtraTreeRegressor,
RandomForestRegressor, ExtraTreesRegressor, XGBRegressor,
LGBMRegressor, CatBoostRegressor)
      for algo in algos:
            model = algo()
            model_results_train = get_scores(train_y, get_preds(model,
train_x))
            model_results_val = get_scores(val_y, get_preds(model, val_x))
            model_results_test = get_scores(test_y, get_preds(model, test_x))
```

```
                data = {
                       "Train": model_results_train,
                       "Val": model_results_val,
                       "Test": model_results_test
                }

                model_results = model_results.append(data,
index=[f'{exp}_{type(model).__name__}'])
```

6. Discussion

In reference to the paper [3], our contribution extends the given data-driven approach for recurrent neural networks to multiple flavours of RNNs such as LSTM, BiLSTM, GRU, and BiGRU. Instead of comparing the RNN with different neural networks such as feed-forward neural networks (FFNN), our methodology creates an ensemble of different RNNs and uses several regression models for a baseline for the ensemble.

6.1. Critical Point Results

For the below mentioned regression models, the data was analyzed for critical points, and for these critical points, the models were trained. [4] The following are the criteria for a critical point in the several charging and discharging cycles of a battery:

- Time at which the highest temperature is measured
- Time at which the lowest voltage has been measured
- The first instance at which voltage drops below 1 volt after 1500 seconds

The results obtained by using critical points and regression models are as follows

Model	**Train RMSE**	**Validation RMSE**	**Test RMSE**
LinearRegression	0.01383664415	0.01392169419	0.01385578774
HuberRegressor	0.01418872062	0.01437233097	0.01427887303
KNeighborsRegressor	0.01150298217	0.01522225374	0.01286339815
LinearSVR	0.01541403963	0.01587278478	0.01559617838
NuSVR	0.01391938576	0.01504345164	0.01364735242
XGBRegressor	0.00333922421	0.01612931921	0.01382234489
LGBMRegressor	0.01270568957	0.0151634902	0.01586447461
CatBoostRegressor	0.009080155243	0.0143311698	0.01264471488

6.2. Ensembling

It is an approach to combining multiple machine learning models in the prediction process. These models are referred to as base estimators. It is a solution to overcome the following technical challenges of building a single estimator:

- High variance: The model is very sensitive to the provided inputs to the learned features.
- Low accuracy: One model or one algorithm to fit the entire training data might not be good enough to meet expectations.
- Features noise and bias: The model relies heavily on one or a few features while making a prediction.

In order to surpass the results obtained by the reference paper [3], ensembling has been adopted for different RNN models.

6.2. RNN Ensemble Results

Various RNN models were trained on the 9 experiments which were then ensembled to reduce each other's shortcomings.

The results were obtained using various ensembling models are as follows:

Experiment	Model	Train RMSE	Validation RMSE	Test RMSE
Experiment 1	Experiment1_CatBoostRegressor	0.007525	0.027265	0.020191
	Experiment1_DecisionTreeRegressor	0	0.050215	0.04708
	Experiment1_ExtraTreeRegressor	0	0.030343	0.025839
	Experiment1_ExtraTreesRegressor	2.05E-15	0.032114	0.019708
	Experiment1_HuberRegressor	0.022796	0.038286	0.034124

	Experiment1_KNeighborsRegressor	0.025611	0.043548	0.038138
	Experiment1_LGBMRegressor	0.012195	0.034856	0.030906
	Experiment 1_LinearRegression	0.022753	0.036982	0.033052
	Experiment 1_LinearSVR	0.023171	0.03899	0.034141
	Experiment 1_NuSVR	0.018096	0.031045	0.028898
	Experiment 1_RandomForestRegressor	0.006901	0.034251	0.023834
	Experiment 1_SVR	0.043566	0.053136	0.053495
	XGBRegressor	0.001642	0.031135	0.02428
Experiment 2	Experiment 2_CatBoostRegressor	0.002012	0.022546	0.122705
	Experiment 2_DecisionTreeRegressor	0	0.017048	0.124136
	Experiment 2_ExtraTreeRegressor	0	0.022856	0.136466
	Experiment 2_ExtraTreesRegressor	2.32E-15	0.025255	0.127762
	Experiment 2_HuberRegressor	0.018581	0.017476	0.124463
	Experiment 2_KNeighborsRegressor	0.02195	0.017122	0.123213
	Experiment 2_LGBMRegressor	0.015742	0.01869	0.121217

	Experiment 2_LinearRegression	0.018389	0.018143	0.124049
	Experiment 2_LinearSVR	0.018887	0.018774	0.125641
	Experiment 2_NuSVR	0.018596	0.019502	0.129266
	Experiment 2_RandomForestRegressor	0.008446	0.016777	0.118291
	Experiment 2_SVR	0.041897	0.032013	0.122625
	Experiment 2_XGBRegressor	0.000803	0.019872	0.126571
Experiment 3	Experiment 3_CatBoostRegressor	0.00378	0.032155	0.05338
	Experiment 3_DecisionTreeRegressor	0	0.029699	0.060948
	Experiment 3_ExtraTreeRegressor	0	0.03498	0.060703
	Experiment 3_ExtraTreesRegressor	1.97E-15	0.027225	0.050309
	Experiment 3_HuberRegressor	0.017599	0.03181	0.054312
	Experiment 3_KNeighborsRegressor	0.01975	0.031337	0.058484
	Experiment 3_LGBMRegressor	0.017093	0.031057	0.055703
	Experiment 3_LinearRegression	0.017511	0.031071	0.05419

	Experiment 3_LinearSVR	0.02083	0.03787	0.054505
	Experiment 3_NuSVR	0.0168	0.032595	0.056635
	Experiment 3_RandomForestRegressor	0.007274	0.027616	0.056833
	Experiment 3_SVR	0.04701	0.037065	0.06863
	Experiment 3_XGBRegressor	0.001102	0.032008	0.052786
Experiment 4	Experiment 4_CatBoostRegressor	0.008048	0.055911	0.09889
	Experiment 4_DecisionTreeRegressor	0	0.041849	0.101487
	Experiment 4_ExtraTreeRegressor	0	0.038955	0.096077
	Experiment 4_ExtraTreesRegressor	1.91E-15	0.034634	0.097582
	Experiment 4_HuberRegressor	0.015199	0.034741	0.097402
	Experiment 4_KNeighborsRegressor	0.03315	0.034436	0.106181
	Experiment 4_LGBMRegressor	0.064483	0.076645	0.094264
	Experiment 4_LinearRegression	0.015045	0.035629	0.097711
	Experiment 4_LinearSVR	0.015473	0.034592	0.097312
	Experiment 4_NuSVR	0.016541	0.039091	0.114237

	Experiment 4_RandomForest Regressor	0.014185	0.0366	0.097902
	Experiment 4_SVR	0.036244	0.051548	0.127786
	Experiment 4_XGBRegressor	0.001557	0.037118	0.100524
Experiment 5	Experiment 5_CatBoostRegressor	0.008265	0.045774	0.074877
	Experiment 5_DecisionTreeRegressor	0	0.036995	0.027503
	Experiment 5_ExtraTreeRegressor	0	0.039634	0.092641
	Experiment 5_ExtraTreesRegressor	2.05E-15	0.030819	0.037207
	Experiment 5_HuberRegressor	0.045422	0.03647	0.047261
	Experiment 5_KNeighborsRegressor	0.062932	0.058599	0.074977
	Experiment 5_LGBMRegressor	0.081215	0.061909	0.090574
	Experiment 5_LinearRegression	0.044266	0.036351	0.04396
	Experiment 5_LinearSVR	0.048037	0.038868	0.045309
	Experiment 5_NuSVR	0.05028	0.04351	0.051569
	Experiment 5_RandomForest Regressor	0.023868	0.033271	0.033192

	Experiment 5_SVR	0.070282	0.063454	0.08037
	Experiment 5_XGBRegressor	0.001058	0.034712	0.053746
Experiment 6	Experiment 6_CatBoostRegressor	0.00643	0.033666	0.031511
	Experiment 6_DecisionTreeRegressor	0	0.027417	0.021631
	Experiment 6_ExtraTreeRegressor	0	0.022854	0.025788
	Experiment 6_ExtraTreesRegressor	1.27E-15	0.018433	0.024366
	Experiment 6_HuberRegressor	0.012126	0.018583	0.020066
	Experiment 6_KNeighborsRegressor	0.048935	0.033489	0.090067
	Experiment 6_LGBMRegressor	0.02413	0.027329	0.038727
	Experiment 6_LinearRegression	0.01206	0.018489	0.019934
	Experiment 6_LinearSVR	0.013803	0.019245	0.020488
	Experiment 6_NuSVR	0.012369	0.020624	0.021946
	Experiment 6_RandomForestRegressor	0.007108	0.027944	0.028014
	Experiment 6_SVR	0.054645	0.068334	0.066069
	Experiment 6_XGBRegressor	0.001206	0.036924	0.027523

	Experiment 7_CatBoostRegressor	0.004757	0.036878	0.068955
	Experiment 7_DecisionTreeRegressor	0	0.033404	0.020211
	Experiment 7_ExtraTreeRegressor	0	0.021588	0.019902
	Experiment 7_ExtraTreesRegressor	1.58E-15	0.027808	0.018275
	Experiment 7_HuberRegressor	0.015001	0.026221	0.013487
	Experiment 7_KNeighborsRegressor	0.088213	0.073078	0.130387
	Experiment 7_LGBMRegressor	0.051494	0.066722	0.115932
	Experiment 7_LinearRegression	0.014968	0.026394	0.013531
	Experiment 7_LinearSVR	0.016352	0.029895	0.01475
	Experiment 7_NuSVR	0.044845	0.059594	0.086954
	Experiment 7_RandomForestRegressor	0.008481	0.025483	0.018102
	Experiment 7_SVR	0.072164	0.076319	0.114204
Experiment 7	Experiment 7_XGBRegressor	0.00124	0.028077	0.026071
Experiment 8	Experiment 8_CatBoostRegressor	0.004868	0.132407	0.42213

	Experiment 8_DecisionTreeRegressor	0	0.075652	0.339136
	Experiment 8_ExtraTreeRegressor	0	0.110393	0.394653
	Experiment 8_ExtraTreesRegressor	1.16E-15	0.082802	0.371635
	Experiment 8_HuberRegressor	0.029361	0.064163	0.392227
	Experiment 8_KNeighborsRegressor	0.266649	0.491128	0.519707
	Experiment 8_LGBMRegressor	0.13042	0.330439	0.449304
	Experiment 8_LinearRegression	0.029229	0.063454	0.391643
	Experiment 8_LinearSVR	0.042245	0.108166	0.342323
	Experiment 8_NuSVR	0.107418	0.251599	0.391663
	Experiment 8_RandomForestRegressor	0.03998	0.123165	0.379643
	Experiment 8_SVR	0.129907	0.291281	0.421267
	Experiment 8_XGBRegressor	0.000734	0.084321	0.407328
Experiment 9	Experiment 9_CatBoostRegressor	0.006819	0.043317	0.031779
	Experiment 9_DecisionTreeRegressor	0	0.040082	0.034598

	Experiment 9_ExtraTreeRegressor	0	0.045053	0.035488
	Experiment 9_ExtraTreesRegressor	1.59E-15	0.039875	0.031513
	Experiment 9_HuberRegressor	0.016913	0.03794	0.033117
	Experiment 9_KNeighborsRegressor	0.055853	0.132274	0.052844
	Experiment 9_LGBMRegressor	0.04493	0.124974	0.031858
	Experiment 9_LinearRegression	0.016866	0.037705	0.033333
	Experiment 9_LinearSVR	0.017088	0.039126	0.032529
	Experiment 9_NuSVR	0.017986	0.043229	0.032546
	Experiment 9_RandomForestRegressor	0.019014	0.063436	0.034201
	Experiment 9_SVR	0.051839	0.077244	0.05949
	Experiment 9_XGBRegressor	0.001369	0.039465	0.034176

6.4. Future Scope

Since in our contribution, various RNNs were ensembled to overcome some shortcomings of each other such as LSTMs being sensitive to different random weight initializations and GRU having a lower convergence rate, as a part of our future scope, more complex RNNs such as transformers can be included in the ensemble to get better results.

7. References

1. M. Hannan, M. Lipu, A. Hussain, and A. Mohamed, “A review of lithium-ion battery state of charge estimation and management system in electric vehicle applications: Challenges and recommendations,” Renewable and Sustainable Energy Reviews, vol. 78, pp. 834–854, 2017.
2. Y. Chen, Y. He, Z. Li, L. Chen and C. Zhang, "Remaining Useful Life Prediction and State of Health Diagnosis of Lithium-Ion Battery Based on Second-Order Central Difference Particle Filter," in IEEE Access, vol. 8, pp. 37305-37313, 2020, doi: 10.1109/ACCESS.2020.2974401.
3. Ansari, S.; Ayob, A.; Hossain Lipu, M.S.; Hussain, A.; Saad, M.H.M. Data-Driven Remaining Useful Life Prediction for Lithium-Ion Batteries Using Multi-Charging Profile Framework: A Recurrent Neural Network Approach. Sustainability 2021, 13, 13333. https://doi.org/10.3390/su132313333
4. Y. S. Jain, A. D. Veer, G. S. Sawant, Y. R. Jain and S. S. Udmale, "Novel Statistical Analysis Approach for Remaining Useful Life Prediction of Lithium-Ion Battery," 2021 12th International Conference on Computing Communication and Networking Technologies (ICCCNT), 2021, pp. 1-6, doi: 10.1109/ICCCNT51525.2021.9579982.