BUILDING A MACHINE LEARNING MODEL FOR BATTERY CAPACITY PREDICTION

Vincenzo Gallicchio, Luigi Piccolo, Mauro Vitale, Nicola Volpe
University of Salerno

July 13, 2019

Abstract

Capacity degradation monitoring of lithium batteries is necessary to ensure the reliability and safety of electric objects in the "Internet Of Things". However, capacity of cell is related to its complex internal physicochemical reactions and thermal effects and cannot be measured directly. A data-driven remaining capacity estimation approach for lithium-ion batteries based on Machine Learning is proposed in this work.

Keywords: *Data Science, Machine Learning, Deep Learning, Predictive model, Random forest*

I. Introduction

Energy storage devices, such as lithium-ion batteries, experience degradation over time and during usage. These phenomena, respectively known as calendar-aging and cycle-aging, lead to a reduction of the energy storage capacity (Q) and power capabilities. These indicators are usually grouped under the term State-of-Health (SoH). Being able to predict the actual capacity of a battery, and its degradation rate, is important for performance reasons. In Internet of Things (IoT) applications, e.g. in wireless sensor networks, the possibility of assessing the current status of the battery can be used to enable new energy consumption strategies. For example, data transmission, a well known energy greedy process, can be adapted basing on the SoH. In mobility applications, the vehicle driving distance is directly related to the capacity (namely the maximum amount of energy that is possible to store). Moreover, the market of stationary applications, can take advantage of the so-called Second-Life of batteries. Since the performance requirements for these applications are less strict than others, a battery dismissed from the main application (e.g. an automotive battery) can still be used to store energy harvested from renewable sources (such as photo-voltaic) in domestic scenarios. This allows to reduce the cost of the plant and helps the spreading of green solutions. However, the Second Life transition requires first a characterization of the cell SoH. Unfortunately, battery degradation is a long-term process, and, moreover, it is caused by a manifold of electrochemical reactions. This makes the physical modeling of the degradation phenomenon a very challenging task and motivates the research for new techniques to assess the battery capacity. The simplest way to determine the battery capacity is by following the capacity definition: discharge the battery from a fully charged state with nominal current at nominal temperature from a fully charged state until the battery's cut-off voltage is reached. This allows to obtain the discharged ampere hours. This procedure can be easily performed in laboratory tests but it is not feasible in most of the application. Indeed it requires to inhibit the application from the normal operations, to perform the discharge and to keep constant the operating conditions (e.g. the temperature). Another simple possibility is to measure the charged ampere hours, but this has similar disadvantages. Moreover it would require to start the experiment from a fully discharged battery, a condition that is avoided in most of the applications. For these reasons, various techniques for the estimation of Q have been suggested

[Farman2019] [2]. These are mainly classified in model-based and data-driven methods. The first are based on mathematical models of the battery and adaptive filtering techniques such as the Kalman filter or the particle filter. These techniques allow to perform an estimate of the capacity but are conditioned to the dynamics of the battery current or to the estimate of the state of charge [Plett] [3,4,5]. This means that, under certain conditions, the estimate could be unreliable. In contrast, data-driven methods are promising for improving generality. Most of the works in this category considers current(I), voltage(V) and temperature (T) data from charging-discharging cycles. In these conditions aging is accelerated and the load pattern is kept constant [Pan] [6]. While these techniques are proved to work, also in this case the operating conditions are not generalized. The use of artificial load patterns shorten the time for the experimental and the data collecting activities, but may introduce misleading results where a simple mapping based on the correlation between the variable of the system is established. Thus, this approach is no longer valid when new dynamic conditions, not captured by the correlation between variables, may arise. For this reason the open source Randomized Battery Usage Repository has been used [1] and some experiments on deep neural network have been performed. Deep learning techniques are known to require a massive amount of data, that is still not available for an extensive training. However, car manufacturer are starting to log data from vehicle Battery Management Systems (BMS).

II. Dataset

The Dataset used is called the "Randomized Battery Usage Data Set", downloaded from the NASA data repository [1]. This data set is composed of 7 experiments each carried out on 4 different batteries.

i. File

Each experiment contains:

- A **.mat** file: Matlab data structure for each battery;
- A **.Rda** file: An R data frame containing the same data as the Matlab data structure;
- A **.m** file: An example Matlab code that recreates all of the plots shown in the README document.

ii. Data structure

Each of the **.mat** file contains a Matlab data structure called "data". The top level of this structure contains 3 fields:

1. **procedure**: a string naming the experimental procedure;
2. **description**: a more detailed text description of the experimental procedure;
3. **step**: an array of structs containing cycling data.

Within the **step** array you will find a struct with the following fields:

1. **comment**: string description of step;
2. **type**: one character identifier of step: "C" = Charging, "D" = Discharging, "R" = Resting (current = 0);
3. **relativeTime**: vector of sample time in seconds, referenced to the beginning of the current step;
4. **time**: vector of sample time in seconds, referenced to the beginning of the experiment;
5. **voltage**: vector of sample voltage in units of Volts;
6. **current**: vector of sample current in units of Amps;
7. **temperature**: vector of sample temperature in units of degrees C;
8. **date**: date and time at which the current step was started in dd-Mon-yyyy HH:MM:SS format.

Each of the .Rda files contains the same information as the Matlab data files. The step data structure in each .Rda file is formatted as a dataframe wherein each row represents a unique charging, discharging, or resting step in the battery cycling experiment.

iii. Types of reference profiles

Batteries are cycled using three types of reference profiles:

1. A low current discharge at 0.04A is used to observe the batteries open circuit voltage as a function of SOC.
    - This profile is identified in the **step** data structure with the **comment** field equal to "**low current discharge at 0.04A**".
    - Resting periods that occur immediately before and after the low current discharge are identified with the **comment** field = "**rest prior low current discharge**" and **comment** field = "**rest post low current discharge**" respectively.
2. A reference charge and discharge cycle is used to observe the battery capacity after every 1500 RW steps cycles.
    - Batteries are first charged at 2A (constant current), until they reach 4.2V, at which time the charging switches to a constant voltage mode and continues charging the batteries until the charging current falls below 0.01A: this step is identified with the **comment** field = **"reference charge"**.
    - Batteries are then rested for a period of time with no current draw: this step is identified with the **comment** field = **"rest post reference charge"**.
    - Batteries are then discharged at 2A until the battery voltage crosses 3.2V: this step is identified with the **comment** field = **"reference discharge"**.
    - Batteries are then rested for a period of time: this step is identified with the **comment** field = **"rest post reference discharge"**.
3. A pulsed current discharge of fully charged batteries is performed after every 3000 RW steps in order to benchmark changes to battery transient dynamics. The pulsed current discharge consist of a 1A load applied for 10 minutes, followed by 20 minutes of no load.
    - This discharging profile is identified by alternating steps of **comment** = **"pulsed load (rest)"** and **comment** = **"pulsed load (discharge)"**;
    - A resting period after either a pulsed discharge or a pulsed charge is denoted by **comment** = **"rest post pulsed load or charge"**.
4. A pulsed current charge of recently discharged batteries is performed after every 3000 RW cycles in order to benchmark changes to battery transient dynamics. The pulsed current charge consists of a 1A charging current applied for 10 minutes, followed by 20 minutes of rest.
    - This discharging profile is identified by alternating steps of **comment** = **"pulsed charge (rest)"** and **comment** = **"pulsed charge (charge)"**;
    - A resting period after either a pulsed discharge or a pulsed charge is denoted by **comment** = **"rest post pulsed load or charge"**.

iv. RW mode

The random walk (RW) mode of battery cycling consist of:

1. Selecting a charging or discharging current at random from the set -4.5A, -3.75A, -3A, -2.25A, -1.5A, -0.75A, 0.75A, 1.5A, 2.25A, 3A, 3.75A, 4.5A. Negative currents are associated with charging and positive currents indicate discharging.
2. The selected current setpoint is applied until either the battery voltage goes out-

side the range (3.2V - 4.2V) or 5 minutes has passed.

- These steps are identified with the **comment** field = **"discharge (random walk)"** and **comment** field = **"charge (random walk)"**.

3. After each charging or discharging period, there will be a <1s period of rest while a new charging or discharging current setpoint is selected.

- This steps is identified with the **comment** field = **"rest (random walk)"**;
- Steps 2 and 3 are repeated 1500 times, then characterization cycles are preformed to benchmark battery state of health.

III. Pre-processing and feature extraction

This phase is the most important, in fact when we talk about deep learning or machine learning, it is necessary to process, arrange and clean data in order to maximize feature discrimination and model performance. To achieve this we used Python Programming language and libraries in combination with the GOOGLE COLAB platform, which provides online hardware resources in the cloud, such as GPU, TPU and pre-configured environments. Specifically, we used Pandas, Scipy, Numpy for data processing and Sklearn and Keras for Machine Learning operations.
The first step that has been accomplished was to download the entire dataset. We then uploaded it on Google drive and through some Google util libraries, we proceeded to download it to the colab OS directory. All the dataset content was then processed and only the data related to the steps were taken under consideration. As we previously said each "Random Walk" is composed by a series of steps, only some of them are actually relevant for learning, so we performed an initial filtering operation discarding the "Rest" steps lasting less than one second which would have not offer any useful information since the battery capacity doesn't relevantly change in those circumstances. Each step contains the wave forms of V,T and I sampled at 1 second. These were then analyzed,processed and appended to a vector to be used for training. For the creation of the final data frame, however, two more sub-procedures were carried out. Since the capacity is not directly measurable, it is necessary to perform a specific step called *reference discharge*, to obtain it. This means that there are specific steps where the capacity was obtained while the normal RW steps were unlabelled. This left us with a semi labelled dataset and a choice to make: training a model with few labelled instances or arrange some other solution. We decide to perform a deductive data imputation technique. Since battery capacity decreases over time, we labelled the unlabelled data with the first successive available capacity going backward building a temporal step structure in which the unknown capacities are on the same level of the first available one. In reality, this is not true, since battery capacity does not decrease as a piece wise constant function but follow a smoother trend. Moreover battery capacity may augment of a small value after a rest period. In fact, battery capacity does increase in those situations but, once the battery is used again it immediately drops down to the original state in which it was left before the "Rest" phase and this justify data our imputation strategy. Once the label problem was fixed, we had feature vectors composed of 6 values plus the label. Their formatting have been normalized and changed in order to help the model to understand and learn, in fact:

- The time has been converted into months.
- The relative time has been converted into hours.
- The voltage, current and temperature vectors were averaged into one value.

We have then decided to create an artificial feature value, the column related to the previous capacity to highlight the temporal sequence in training data. At this point, the dataframe has been created containing as rows, all the

steps previously formatted and as columns the following: type, time, relative time, voltage, temperature, previous capacity and capacity. We have adopted two strategies: Deep Learning and Machine Learning using a regressive method. Each of the techniques takes the same dataset as its input but in two different variations, which will be explained below.

IV. Deep Learning

In order to build a model that would properly generalize on battery capacity degradation data we initially chose to rely on the complexity and power of *deep neural networks* (DNN). The rationale behind our choice was that training data had many artificial labels temporally arranged in a way that would form a step structure of the dependent attribute which does not adequately reflects the real label distribution and would probably confuse the simplest Machine Learning models.

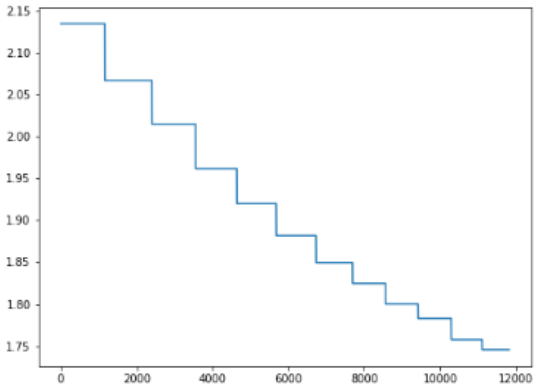

Figure 1: *Capacity values step structure - Capacity(Ah) on the y-axis and Sample on x-axis*

In fact, as we discussed in the previous chapter, a simplistic data imputation operation on training instances with missing labels was performed causing label graphic representation to be plainer and not much discriminant and meaningful in many temporal segments. So, there was a need for a powerful and reliable model capable of generalizing the decreasing trend over time as much realistically as possible. Moreover, at the beginning of our experiments, we had a discrete number of features: "Type", "Absolute Time", "Relative Time", "Voltage", "Current","Temperature" and "Previous-Capacity". A total of 7 features which made us wonder about the complexity of the hypothetical multi-dimensional predictive mathematical model that would have came out of them and the adequacy of the simplest and most famous Machine Learning techniques, such as Linear-Regression. At the end of our experiment only 6 of them were left ("Type" was dropped) since the other one, after an empirical process, happened not to have relevant weight in decision processes. Now that we have explained the reasons behind the initial selection of DNN let's dive into the description of its structure, activation and optimization functions.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 1024)              7168
_________________________________________________________________
dense_2 (Dense)              (None, 1024)              1049600
_________________________________________________________________
dense_3 (Dense)              (None, 1024)              1049600
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 1025
=================================================================
Total params: 2,107,393
Trainable params: 2,107,393
Non-trainable params: 0
_________________________________________________________________
```

Figure 2: *Model structure*

We have found an acceptable balance in a structure composed of an input layer, 3 full-dense hidden layers with 1024 neurons each and an output layer composed of one regression aggregator neuron. The hidden layer neurons activation function is the famous ReLU that let flow in the network non-negative numbers only. The last layer neuron simply outputs the numerical input it has been given. Fig. 2 As for the optimization function, we decided to choose "Adam" (the most recommended in literature) and stick with its default parameters. The only parameter we tuned and through which we influenced the learning rate was batch-size parameter with a value of 32, in fact, it is only after a training batch has been processed that the NN weights get updated by the optimizer. Let's now proceed through the training phase. As we said in the previous chapter we had a total of 28 'Random walk',

each one composed of thousand of steps, forming a dataset of approximately 600.000 training examples. We decided to cut the 28th Random walk out in order to use it for the test phase, shuffle and split the remaining examples in training and validation set using a 80-20 criteria. The optimal value for the number of epochs was found to be 10 over which the models start to get stable for many more epochs. (For time and computational constraints not more than 30 epochs were tested) Regularization and overfitting prevention was obtained through structure tuning operations, we did in fact, at the beginning of our experiment, let the model overfit using layer with over 2000 neurons, we then regularized it diminishing the number of neurons per layer until we started to obtain optimal e relevant results with a loss e mean absolute error value of 0.03. Fig. 3

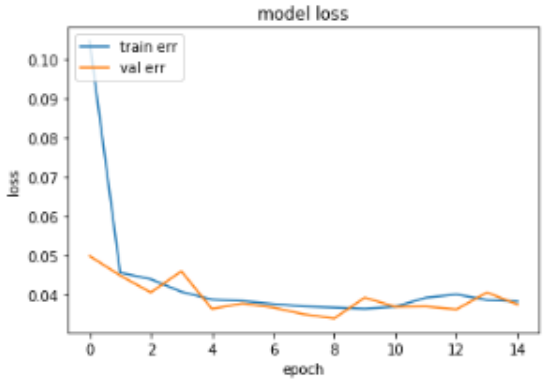

Figure 3: *Training and Validation Loss*

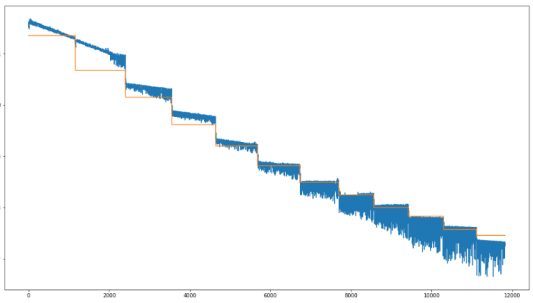

Figure 4: *DNN Predictions on 28th RW*

As we can see, the DNN models the data perfectly in the points where real capacity were given as a the real label and approximates the decreasing trend of a real battery in a real world scenario where data imputation was performed. Unfortunately, Deep learning models are hardly analyzable, we were aware of their effectiveness but we were unable to offer sufficient reasons why, for example the importance of features, although we were able to extract some, like weights per neuron. The only way we had to deduct feature importance were to perform different training sessions with one or more missing feature value and then perform comparations. (These test can be found on our Google drive rep) We found out the importance of "PreviousCapacity" and "AbsoluteTime" without which the original model wouldn't perform so good.

V. Random Forest

After an evaluation of the obtained behaviour and results using a Deep Learning model, we concluded the problem not to be as much complex as we expected, so we decided to try with one of the best regression Machine Learning models, the Random Forest. Initially, like in the DNN tests, we used the entire stack of features but, once again, we were able to see some features' irrelevance in the decision processes. The number of features was reduced to 5:"Time" expressed in month, "Relative Time" expressed in hours, "Voltage", "Current" and "Temperature". Less than the final DNN model.This is immediately explained: Random Forests have a simpler regression criteria, they, in a certain way, model data strictly basing their decisions on simpler (with respect to DNN) statistical inspectionable insights. This last property, let us have a better understanding on feature weights and importance resulting in a simpler and lighter/easy to use model which predicts battery capacities using features that are actually computable in real world scenarios and applications. In this approach we used the first 27 Random Walks entirely for training and left the 28th random walk for testing, skipping the validation phase. Actually, in our Google Drive directory much more tests have

been done, in particular we tried training and testing on different subsets of the dataset and features obtaining similar results. Back to the operative phase, we initially set the number of trees in the forest to 10 and then to 20. Another parameter changed by the default setting is the number of features to consider when looking for the best split. This value has been set to 3 and therefore considers 3 features at each split. As we said before the RandomForest regressor has been very useful for gaining precious info on the discriminitive feature values, we have in fact detected their importance.

Feature	Importance
Time	60.382
RelativeTime	8.622
Voltage	5.813
Current	6.563
Temperature	18.62

Table 1: *Features' importance*

Time is the most important feature, having a weight of 63.3% on the prediction, while the less relevant feature is represented by the voltage, with a weight of 5.8%. Here the results were slightly better than in the Deep Learning approach. We achieved an accuracy of 99.63% with a mean absolute error of 0.01 on the test set.

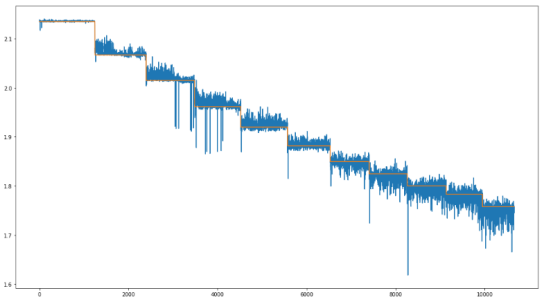

Figure 5: *Labelled capacity vs predicted capacity*

VI. Conclusions and future developments

In conclusion, both methods performed well but Random Forest's got some extra points. These can be attributed to the selected features and to the execution speed. As we previously said, Random Forest's feature are lighter and sensor easily extractable data and this make RF model suitable for real world applications. Conversely the DLM, although modelled data with slightly more precision is slower in execution and its feature can't be applied universally to all applications, for example, a smart phone battery might not have information about previous capacity. Finally,the objectives set have been achieved and the residual capacity of the batteries is predicted in a satisfactory manner. In the future, however, it is possible to improve the system through the development of:

- Online prediction applicability.
- Metrics accuracy augmentation.
- Execution time.
- More sophisticated data imputation technique with local mean. (e.g KNN)

Summing up, we think that deeplearning will perform better than RandomForest with a more accurate data imputation strategy and a wider dataset, but until there's no much difference in prediction performance Random Forest will be preferred for its semplicity and speed. Fig. 6 resumes the workflow implemented to investigate ML techniques for battery capacity prediction. We think that this kind of framework may be helpful in future.

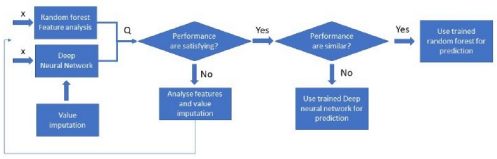

Figure 6: *Diagram*

References

[1] B.Bole, C. Kulkarni, and M. Daigle. "Randomized Battery Usage Data Set". In: *NASA Ames Prognostics Data Repository* (2019).

[2] Alexander Farmann et al. "Critical review of on-board capacity estimation techniques for lithium-ion batteries in electric and hybrid electric vehicles". In: *Journal of Power Sources* 281 (2015), pp. 114–130. ISSN: 0378-7753. DOI: https://doi.org/10.1016/j.jpowsour.2015.01.129. URL: http://www.sciencedirect.com/science/article/pii/S0378775315001457.

[3] Peiyao Guo, Ze Cheng, and Lei Yang. "A data-driven remaining capacity estimation approach for lithium-ion batteries based on charging health feature extraction". In: *Journal of Power Sources* 412 (2019), pp. 442–450. ISSN: 0378-7753. DOI: https://doi.org/10.1016/j.jpowsour.2018.11.072. URL: http://www.sciencedirect.com/science/article/pii/S0378775318313272.

[4] Gregory L. Plett. "Recursive approximate weighted total least squares estimation of battery cell total capacity". In: *Journal of Power Sources* 196.4 (2011), pp. 2319–2331. ISSN: 0378-7753. DOI: https://doi.org/10.1016/j.jpowsour.2010.09.048. URL: http://www.sciencedirect.com/science/article/pii/S037877531001654X.

[5] W. Yan et al. "A Battery Management System With a Lebesgue-Sampling-Based Extended Kalman Filter". In: *IEEE Transactions on Industrial Electronics* 66.4 (Apr. 2019), pp. 3227–3236. ISSN: 0278-0046. DOI: 10.1109/TIE.2018.2842782.

[6] Heng Zhang et al. "An improved unscented particle filter approach for lithium-ion battery remaining useful life prediction". In: *Microelectronics Reliability* 81 (2018), pp. 288–298. ISSN: 0026-2714. DOI: https://doi.org/10.1016/j.microrel.2017.12.036. URL: http://www.sciencedirect.com/science/article/pii/S0026271417306005.