Physics-Informed Learning for Li-Ion Battery Health Prediction

Final Project for ECE 228: Track #2
Group Number #31
Arturo Amaya, Anushka Sarode, Nathaniel Greenberg, Pete Quawas

Abstract

Accurately predicting lithium-ion (Li-ion) battery degradation is crucial for ensuring safety, optimizing energy use, and managing warranties. This study evaluates the use of physics-informed neural networks (PINNs) for estimating state of health (SOH) in Li-ion cells using NASA's publicly available four-cell dataset cycled under varying temperatures (4°C, 24°C, 43°C). Battery health prediction is crucial for safety, warranty, and efficient energy use. Existing methods for battery health prediction are challenged by diverse battery chemistries and vast operating procedures. Existing SOH estimations rely on capacity measurements from complete charge or discharge curves. However, the difficulty of obtaining full discharge curves in practice motivates this work to incorporate statistical features from short ranges in the charge and discharge cycles to explore the effects of decoupling SOH estimation from complete charging cycles.

1 Introduction

Lithium-ion batteries play a crucial role in powering modern technologies, from electric vehicles to aerospace systems. Over time, these batteries experience capacity fade, leading to reduced performance and eventual failure. Accurately forecasting when a battery will reach its end of life, typically defined as a 30% drop from nominal capacity, is essential for maximizing performance, ensuring safety, and managing operational costs.

The primary conventional technique for battery health diagnosis involves putting the battery through a full charge cycle, from 100% state of charge (SOC) to a near-empty state. Ironically, estimating battery health in this manner can be detrimental to its lifespan, especially if the battery is cycled frequently. In addition, operational limitations can make performing this procedure difficult or impossible, and putting the battery near 0% SOC can be dangerous.

Related Work

State of Health

The SOH of a battery is defined as the ratio of the full charge capacity of a battery in its current state and its nominal charge–aka, the full charge capacity of a battery when it is initially bought. SOH is a function of various measurable battery performance parameters (i.e, current, voltage, resistance, and self-discharge rate, etc.) and is generally expressed in terms of capacity while holding all other parameters constant.

Data-Driven Models

In the work by Saha et al., three features related to battery capacity degradation were extracted and used to compare two model architectures. The features included: 1) number of discharge cycles, 2)

Preprint. Under review.

time required for the voltage of different discharging cycles to reach 4.2 V and 3) time required for the temperature of different discharging cycles to reach a maximum value. A support vector regression (SVR) model was found to be the better classifier in comparison to a long-short term memory (LSTM) model, while additionally successfully learning and fitting several outlier data points in the dataset. While the LSTM architecture was hypothesized to help fit the outliers observed in data points and the general non-linearity of the dataset due to the dependency of capacity with respect to cycles, the model struggled with predicting sharp and sudden capacity upward trends that occurred at random intervals throughout the discharge lifecycle of the batteries. Overall, these data-driven models do not require physical knowledge and focus on the relationship between the extracted features and the output capacity; therefore, the extraction of features determines the performance of the SOH estimation, and the trained models are extremely dataset-specific[1].

Physics-Informed Models

Purely physics-based models are stable and accurate, but have high computational costs as batteries with different chemical compositions require different model parameters. The work by Wang *et al.* combines the strengths of physics-based and data-driven approaches in an exploration to overcome the issue of obtaining full charge capacity data in practice. The hybrid model employed in this work utilized neural networks to capture battery degradation dynamics from the perspective of empirical degradation and state space equations[2].

Project Contributions

As established by previous work in battery health estimation, feature extraction of charge cycles is a powerful method to obtain battery-specific data for model training. As this study seeks to improve upon existing methods for efficient battery SOH estimation, the extensively tested method of feature extraction for the generation of a robust input dataset is relied upon. Additionally, physics-informed models are seen to outperform data-driven models as illustrated by Wang *et al.* To leverage these experimentally-determined benefits, this model evaluates the performance of a PINN architecture against 17 extracted features used in the work by Wang *et al.* on a four-cell lithium-ion battery dataset publicly provided by NASA. To contribute improve upon these benchmark loss values, additional features are included in the training dataset to increase model robustness. These features include spectral entropy calculations on the current and voltage curves of the charging cycles, as well as second-derivative calculations of the original features. Extended feature calculation is performed in this project to probe whether an accurate model can be generated by using fewer battery cycling data if enough derived features are incorporated in the training dataset.

2 Methodology

Dataset

The dataset employed in this project has been collected from a battery testbed at the NASA Ames facility. In this four-cell battery dataset, Li-ion batteries were run through 3 different operational profiles (charge, discharge and electrochemical impedance spectroscopy) at 4 °C, 24 °C, and 43 °C. In this project, the majority of features for model training were extracted from the charge cycles only. Discharges of the batteries were carried out at varying current load levels until the battery voltage fell to at or below 2.7V. The experiments were stopped when the batteries reached the end-of-life criteria of 30% fade in rated capacity (from 2 Ah to 1.4 Ah). Discharge data was not used. For a further discussion see Appendix one.

Feature Extraction

Each of the batteries in the NASA dataset had varying discharge values depending on their rated capacities. Therefore, to maintain generalizability of features within the training dataset, this project adopted the method from Wang*et al.* to obtain ranges of voltage and current values that exist regardless of battery-specific discharges. To this end, two ranges were specified for all batteries in the dataset for feature extraction; the current data was chosen between 0.5A and 0.1A during the battery's constant voltage (CV) charging period (shown in Figure 2, and the voltage data was chosen between 4V and 4.2V during the constant current (CC) period (shown in Figure 1). These current and voltage

ranges exist within all battery charging cycles, and therefore simplified the dataset pre-processing for this project.

Figure 1: Voltage for one charge cycle. The portion circled in red corresponds to the CC mode voltage values we used for feature extraction.

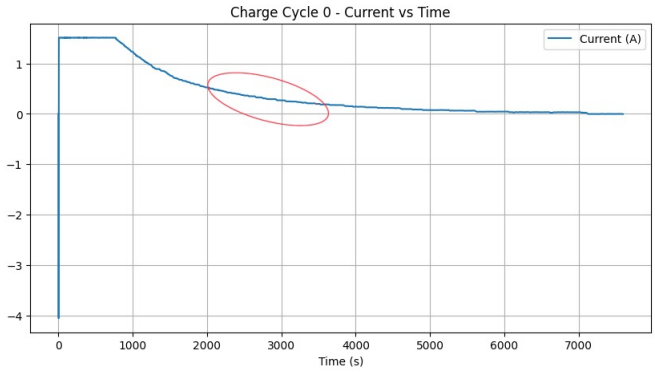
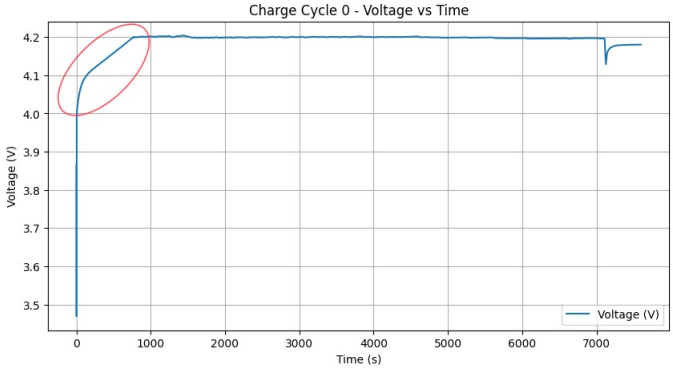


Figure 2: Current for one charge cycle. The portion circled in red corresponds to the CV mode current values we used for feature extraction.



Eight features were extracted from both ranges to form the initial training dataset. These features and a brief explanation of their purpose is as follows. Mean values were extracted from both charge and discharge cycles, as were standard deviation measurements. The meaning of both of these features is obvious, where the mean is the average over the selected time period and the standard deviation is calculated as

$$\sigma = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}\left(\mathbf{x}(i) - \bar{x}\right)^2}$$

where $\bar{x}$ represents the mean of the data in the given time range. The following features are extracted only from the charge cycles. The kurtosis indicates the width of the peak in the charge cycle (*aka*, how thin or broad the peak is), and is calculated as

$$\frac{\sum_{i=1}^{n}\left(\mathbf{x}(i) - \bar{x}\right)^4}{(n-1)\sigma^4}$$

where σ represents the standard deviation in the given time range. The skewness is a measure of the asymmetry around the mean, and is calculated as follows.

$$\frac{\sum_{i=1}^{n}\left(\mathbf{x}(i) - \bar{x}\right)^3}{(n-1)\sigma^3}$$

The charging time is simply the difference between the end and start of the current or voltage range and indicates the duration of the charging mode. The charge accumulated during the mode is given

by the area under the current curve, the slope is the approximate slope during the charging mode and the curve entropy is a rough measure of uncertainty calculated as

$$-\sum_{i=1}^{n} p_i \cdot \log(p_i)$$

where p_i is the normalized value of the curve.

In addition to these 17 features and the cycle index extracted from the CC and CV charging cycles, we also intended to add the dominant frequency of the current and voltage curves, as well as their spectral entropy and the root mean square (RMS) of each signal and the range[1]. We thought that adding the range would tell the model more information about the spread along with the standard deviation, the RMS would tell us the strength of the signal and spectral entropy, which is just the entropy equation applied to the signal after Fourier transformation, would add to our understanding of how much information was in the signal.

FFT-Based and Signal Features

In addition to the 17 statistical features and the cycle index extracted from the Constant Current (CC) and Constant Voltage (CV) charging phases, we engineered six additional features based on frequency-domain and signal-strength analysis:

- **Root Mean Square (RMS)** of voltage and current:

$$\text{RMS}(x) = \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2} \tag{1}$$

  This captures the *signal strength* or energy content of the voltage and current profiles.

- **Range** of voltage and current:

$$\text{Range}(x) = \max(x) - \min(x) \tag{2}$$

  The range provides a simple measure of *signal spread*. While the windowed signals are roughly bounded (e.g., voltage ≈ 0.2 V, current ≈ 0.4 A), we observed sufficient variation to justify explicitly including this metric.

- **Spectral Entropy** of voltage and current:

$$H(P) = -\sum_{i} P_i \log P_i \tag{3}$$

  where P is the normalized power spectral density computed from the FFT of the signal. Spectral entropy measures the *complexity* of the frequency distribution: higher entropy implies a more spread-out (complex or noisy) spectrum, while lower entropy indicates dominant, regular frequency components.

- **Dominant Frequency** (Omitted): Initially, we considered using the index of the frequency with the highest power in the FFT spectrum. However, due to low variability across samples, we found this feature to be non-discriminative and ultimately excluded it from our final feature set.

Implementation Notes. Spectral features were computed by applying the Fast Fourier Transform (FFT) to the zero-mean version of the voltage (from CC phase) and current (from CV phase) signals. We then squared the magnitudes to obtain the power spectrum, normalized it to form a probability distribution, and applied entropy. Features were only extracted when the signal length exceeded 5 and standard deviation was greater than 10^{-6}; otherwise, `NaN` values were assigned to maintain robustness.

The dominant frequency gave us a similar enough result for all data entries that we decided to omit it.

[1]The range is implied in the window that we took, i.e. all voltage ranges should be around 0.2 and all current ranges should be around 0.4 but we still thought it would be useful to include them, and there is indeed a certain degree of variation.

Model Architecture

Our work built on the PINN architecture proposed by Wang *et al.*. It leverages two multi-layer perceptrons that complement each other. The first is a regular MLP $\mathcal{F}$ that takes in $(\mathbf{x}, t)$ i.e. the extracted features and a time variable, which is the cycle index. It produces a result that Wang *et al.* refers to as u. This is the model's prediction for the SOH (i.e. the ratio of capacity and nominal capacity). That is,

$$u = f(\mathbf{x}, t) \approx \mathcal{F}(\mathbf{x}, t)$$

The second MLP is intended to simulate the PDE that describes the rate of decay of the battery capacity, i.e.

$$\frac{\partial u}{\partial t} = g(\mathbf{x}, t, u, u_t, u_x ...) \approx \mathcal{G}(\mathbf{x}, t, u, u_t, u_x)$$

The problem is that this g is difficult to describe in an explicit form, so we use a neural network $\mathcal{G}$ to approximate it. This $\mathcal{G}$ takes the original features, the cycle index and u, as well as the derivatives u_t and $u_{\mathbf{x}}$. This leads to a second loss term MSE_{PDE} that we enforce on top of the original MSE loss $\mathrm{MSE}_{\text{u, label}}$:

$$\mathcal{L} = \mathrm{MSE}_{\text{u, label}} + \alpha\, \mathrm{MSE}_{\text{PDE}}$$

where α is a weighting factor and

$$\mathrm{MSE}_{\text{PDE}} = \frac{1}{n} \sum_{i=1}^{n} (\frac{\partial u_i}{\partial t} - \mathcal{G}_i)^2$$

where $\mathcal{G}_i$ is the output of the second MLP for that particular sample i. The original paper added a third loss term called the physics/monotonicity loss but because of the format of our data[2] we have decided to omit it, or make its weight $\beta = 0$. We modified $\mathcal{G}$ to also take the second derivatives u_{xx} and u_{tt}. The original model chose not to include these because of added computation, but we thought that they would provide additional accuracy and the model is light enough that the added computation shouldn't slow the system down too much.

3 Experiments

3.1 In-depth experiment explanation

The dataset used was previously discussed in the methodology section. It is available on on AWS. It contains several folders with `.mat` files with the data. We uploaded that to google drive and proceeded to use Google Colab. Colab uses Python 3.11 and we decided to use `pytorch`. Colab provides access to GPUs, but we largely didn't need any GPU time.

We extracted features from the dataset by walking through the dataset's file structure and loading only the `.mat` files using Python's `scipy` package. Each `.mat` corresponds to a battery, and has a series of cycles. Each cycle is charge, discharge, or impedance. In line with Wang *et al.*, we only used charge cycle data. To extract the relevant CC and CV windows we used DataFrame filters available from Python's `pandas` package. We also filter out cycles that had strange data, like a duration of 0, or ones that never enter the voltage window of $[4.0, 4.2]$ in CC mode or the current window of $[0.5, 0.1]$ in CV mode. The features are extracted using `scipy`'s built-in functions for kurtosis, skew, entropy and Fourier Transform, as well as standard `numpy` functions like `std()` and `mean()`. The cycle index is the count of valid charge cycles so it is incremented every time we find valid data. The capacity is calculated using `numpy`$'s$ `trapezoid` function to integrate the area under the current curve across the entire charge cycle and the accumulated charge is calculated the same way just in the smaller

[2]Our data contains multiple batteries in one file, so the capacity does not monotonically decrease when you change from one battery to another.

window.[3] After applying this to every battery in the dataset we have a features DataFrame with all of our model's input data.

The model definition was written following the original implementation by Wang *et al.*, as described in the model architecture section. We augmented the model to take in more than the basic parameters to dynamically construct models that used the regular or extended feature datasets, or included/excluded the second derivatives. For testing all we have to do is create an `args` object and pass that into the model constructor. The model uses Adam optimizer for both MLPs and has an learning rate scheduler that runs a few warm-up epochs and sets a cosine learning rate schedule for the model[4]

The model's dataloader also does some data processing based on the implementation by Wang *et al.*. It removes values that are 3σ off of the average, as well as normalizing the data with either the z-score or min-max. The capacity is normalized with the nominal capacity, which in our case is 2Ah. The data is then loaded into Tensor form for the testing and training. The data loader has three different set ups - the first uses an 80/20 split for training and testing and a subsequent 80/20 split inside the training data to make training and validation data, one that has no test set and an 80/20 train/validation split, and one where there is only a test set. For the most part the first setup is used to train and the third setup is used for extracting final metrics.

We end up with:

State MLP Sol$_u$: Linear(in_features,60)→ `Sin` →Linear(60,60)→`Sin`→Dropout(0.2)→Linear(60,32) →Dropout(0.2)→Linear(32,32)→`Sin`→Linear(32,1)

Physics MLP F: Linear(in_features * γ +1, 60)→`Sin`→Linear(60,60)→`Sin`→Dropout(0.2)→Linear(60,1)

Loss: $\mathcal{L} = \mathrm{MSE}_{\text{data}} + \alpha\,\mathrm{MSE}_{\text{PDE}} + \beta\,\mathrm{MSE}_{\text{physics}},\ \alpha = 1, \beta = 0.$

Hyperparameters:

- batch size: 256
- normalization: z-score
- epochs: 10000
- learning rate: 1e-3
- warm-up epochs: 10
- warm-up learning rate: 5e-4
- final learning rate: 1e-4
- learning rate F: 1e-3
- early stop: 500

Most of these were just taken directly from Wang *et al.*'s implementation - we tried a grid search across the hyperparameter space but it took forever and didn't return any interesting changes [5]We did choose the epoch count as a purposefully large number to give the models the opportunity to find lower and lower minima - most of them stopped early anyway.

3.2 Results

Our work led to four possible model configurations we can compare using the NASA dataset - the base model, the model with extended features, the model with second derivatives but regular features, and the model with both second derivatives and extended features. The final statistics for each method are presented below in Table 1:

The model configuration that most positively impacted performance was adding u_{xx} and $u_t t$, the second derivatives. It improved results across MSE (mean squared error), MAE (mean absolute error) and MAPE (mean absolute percentage error). Extending the features used did not improve the MSE, but did improve the MAE and the MAPE with or without the second derivative - adding the second derivative improved the extended feature model's performance. Improving in MAE and MAPE but moving backwards would imply the model is by and large performing better, but fails at outliers in the data that are more heavily penalized by the MSE. It's not a huge improvement, but it is an improvement - possibly with more relevant features we could improve that performance some more.

[3]Note that this applies in both windows because capacity $Q = \int I dt$ regardless of whether we're looking at CC or CV mode.

[4]The dataloader and the learning rate scheduler mimic implementation by Wang *et al.* very closely as it made some design choices that were a bit hard for us to understand in terms of the code.

[5]Basically it said that changing the α to 1.1 would produce marginally better results by about 1e-5.

Table 1: Error-based results comparison between model configurations

Model Moniker	MSE	MAE	MAPE
regular features	0.000985199	0.019209078	0.025809931
extended features	0.001375389	0.018003048	0.024290869
second derivative regular features	0.000857127	0.017249381	0.023150161
second derivative extended features	0.001104505	0.017689174	0.023605143

Table 2: R2 results comparison between model configurations

Model Moniker	R2
regular features	0.929718911
extended features	0.899324871
second derivative regular features	0.938855181
second derivative extended features	0.919152962

Table 2 contains a comparison of the R2 score generated from each model's predictions compared with the ground truth. The R2 score is basically a measure of how well the predictions explain the ground truth. A score of 1 would mean the model perfectly predicts results that match the ground truth. All three models score quite highly - implying they can predict the results well. That being said, only the model with second derivatives and regular features outperformed the base model.

Table 3: Comparing final model size

Model Moniker	Parameter Count
regular features	13662
extended features	14742
second derivative regular features	14682
second derivative extended features	16122

Table 3 compares the number of parameters present in each model. Obviously our configurations increased the number of parameters, and combining both for the model with second derivatives and extended features increased the number of parameters dramatically. In terms of benefit-per-added parameter, the second derivative is the superior configuration compared to our other two expansions on the base model - the number of parameters increases less than just extending the features and it outperforms that configuration in every metric, as seen in tables 1 and 2

Figure 3: The loss curve of all model configurations

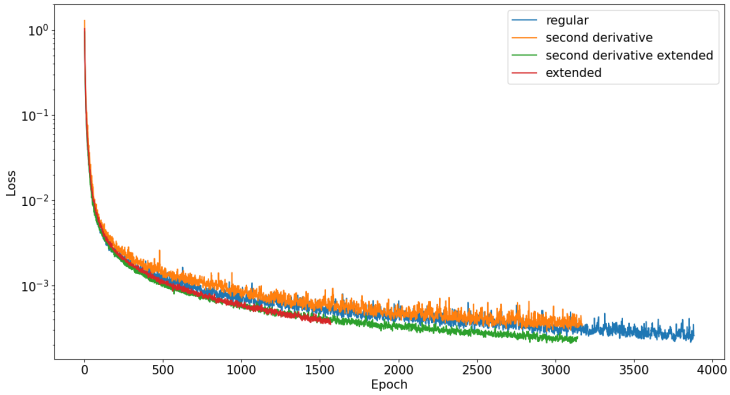


Figure 3 plots the loss curves of the four configurations together. There are some interesting results here. First of all, the regular model took the longest to train, followed by the second derivative model and the second derivative extended model. The regular model taking the longest to train and having a noisy curve would point to the possibility of it overfitting to the data, which may explain its excellent error scores. The loss curves for the models with extended features are noticeably less noisy than those without them. That may also be an indicator of overfitting from the regular-featured models. It would also justify the addition of the extended features as a noise or overfit deterrent. Finally, the extended model with first derivatives only finishes training a lot earlier than any of the other models - about twice as quickly as the next fastest. Given that the difference in each error metric with respect to the regular model is so small, this model may be preferred in use cases where quick training is preferred.

4 Conclusion

The goal of this project was to validate a PINN architecture for battery SOH estimation. To this end, a physics-informed loss term was generated from two fully connected MLPs to estimate the SOH of a battery and to estimate the dynamics of the battery by taking the time derivative of the first MLP. Both first- and second-order derivatives were employed in the model architecture to increase the robustness of the model. The dataset used in this project was NASA's four-cell lithium-ion battery dataset consistent of 56 different Batteries and their respective `.mat` files. The metrics used to evaluate the performance of the trained models include the MSE, MAE, and MAPE. We found that extending the features did not noticeably improve performance but that adding the second derivatives did - thought not by a large amount. Future work may focus on extracting more relevant features than the ones that we chose, as well as using a larger battery dataset.

5 GitHub Repository

The full code and documentation for this project are available at: https://github.com/pete-q/ECE228_Project

References

[1] B. Saha, K. Goebel, *NASA AMES prognostics data repository* **2007**.

[2] F. Wang, Z. Zhai, Z. Zhao, Y. Di, X. Chen, *Nature Communications* **2024**, *15*, 4332.

Appendices

A Discharge features

Discharge data was available to us, but we eventually opted against using it. Discharge information is user-specific, whereas almost all charge cycles use CV and CC modes. To illustrate this, consider a toy example. Most people charge their phone and then use it normally. Arturo charges his phone and then runs Pokemon Go in the background all day, streams soccer games with picture in picture, and generally blitzes through his phone's battery life. The discharge curves for Arturo and a regular person would look vastly different, at there would be no common window to extract data from. The point is that since they look vastly different, it would take a much larger amount of data than we had to attempt to approximate the f that describes SOH as a function of discharge curve features. Applying this method to our dataset would definitely improve the estimates of capacity, but only for our dataset. We prefer to be able to apply the model more generally. In addition, not every use case is measuring the discharge data of their batteries when they're in use, but those statistics are easily extracted during charging - any generalizable model will attempt to work with data that is readily available.