ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΗΛΕΚΤΡΙΚΩΝ ΒΙΟΜΗΧΑΝΙΚΩΝ ΔΙΑΤΑΞΕΩΝ

& ΣΥΣΤΗΜΑΤΩΝ ΑΠΟΦΑΣΕΩΝ

Ενισχυτική Μάθηση και Φυσικά Μοντέλα για Πρόβλεψη Κατάστασης Φόρτισης σε Μπαταρίες Ιόντων Λιθίου

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΚΩΝΣΤΑΝΤΙΝΟΣ ΒΑΣΙΛΑΚΗΣ

Επιβλέπων : Γ. Μέντζας
Καθηγητής Ε.Μ.Π.

Αθήνα, 24 Φεβρουαρίου 2025

Η σελίδα αυτή είναι σκόπιμα λευκή.

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΗΛΕΚΤΡΙΚΩΝ ΒΙΟΜΗΧΑΝΙΚΩΝ
ΔΙΑΤΑΞΕΩΝ & ΣΥΣΤΗΜΑΤΩΝ ΑΠΟΦΑΣΕΩΝ

Reinforcement Learning and Physics-based Models for State of Charge Prediction of Li-Ion Batteries

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΚΩΝΣΤΑΝΤΙΝΟΣ ΒΑΣΙΛΑΚΗΣ

Επιβλέπων : Γ. Μέντζας
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την - Φεβρουαρίου 2025.

.....	.....	.....
Γ. Μέντζας	Δ. Ασκούνης	Ε. Μαρινάκης
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Επ. Καθηγητής Ε.Μ.Π.

Αθήνα, 24 Φεβρουαρίου 2025

Κωνσταντίνος Βασιλάκης

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Ο σκοπός της διπλωματικής εργασίας είναι η ανάπτυξη μιας μεθοδολογίας η οποία μπορεί να χρησιμοποιηθεί για τον υπολογισμό της Κατάστασης Φόρτισης μπαταριών Ιόντων Λιθίου. Γενικότερα δύο βασικές μέθοδοι μπορούν να χρησιμοποιηθούν για την μοντελοποίηση ενός συστήματος, η πρώτη βασίζεται σε κάποιο φυσικό μοντέλο το οποίο αποτελείται από εξισώσεις που περιγράφουν τη συμπεριφορά του συστήματος και η δεύτερη σε προσεγγίσεις βασισμένες σε τεχνολογίες μηχανικής μάθησης όπου η συμπεριφορά μαθαίνεται από κάποιο σύνολο μετρήσεων που ήδη υπάρχει. Σε αυτήν την εργασία χρησιμοποιήθηκε ένας συνδυασμός των δύο παραπάνω μεθόδων όπου ένα φυσικό μοντέλο της μπαταρίας δημιουργείται με βάση ένα Μοντέλο Ισοδύναμου Κυκλώματος ενώ οι παράμετροι του μοντέλου αυτού υπολογίζονται σε πραγματικό χρόνο από ένα νευρωνικό δίκτυο το οποίο εκπαιδεύεται σε ένα σύνολο πειραματικών μετρήσεων χρησιμοποιώντας μεθόδους Ενισχυτικής Μάθησης.

Η μέθοδος που περιγράφεται στοχεύει στο να συνδυάσει πλεονεκτήματα και από τις δύο επιμέρους μεθοδολογίες χρησιμοποιώντας το φυσικό μοντέλο για να προσεγγίσει τη βασική δυναμική του συστήματος και το νευρωνικό δίκτυο για να υπολογίσει τις παραμέτρους του και να βελτιώσει τη συμπεριφορά περιγράφοντας πιθανές μη γραμμικότητες που εμπεριέχονται στα πειραματικά δεδομένα. Στόχος είναι η δημιουργία ενός συνολικού μοντέλου με έναν αποδεκτό συμβιβασμό μεταξύ ακρίβειας και πολυπλοκότητας το οποίο θα μπορεί να χρησιμοποιηθεί σε εφαρμογές που απαιτείται υπολογισμός σε πραγματικό χρόνο.

Λέξεις Κλειδιά: ενισχυτική μάθηση, μοντέλο ισοδύναμου κυκλώματος μπαταρίας, υπολογισμός κατάστασης φόρτισης, μοντελοποίηση μπαταρίας

Η σελίδα αυτή είναι σκόπιμα λευκή.

Abstract

The scope of the thesis is the development of a methodology that can be used in order to estimate the State of Charge of Li-ion batteries. In general two basic methods can be used when modeling a system, physics-based approaches, where the underlying equations of the system are derived and solved, and data-based approaches, where the behavior is learned from a dataset using machine learning methods. In this work a hybrid approach is utilized where a physics-based model for a battery is derived based on the battery's Equivalent Circuit Model (ECM) while the parameters of the model are estimated in real time by a neural network that has been trained on a dataset of real measurements using Reinforcement Learning.

The approach described is focused on combining advantages from both methodologies by using a physics-based model to describe the base dynamics of the system while estimating the model parameters and capturing any additional nonlinearities present by utilizing a data driven approach. There is also a strong focus to produce a combined model with a good trade off between accuracy and complexity that can be used in applications with real time requirements.

Keywords: reinforcement learning, battery equivalent circuit model, state of charge estimation, battery modeling

Η σελίδα αυτή είναι σκόπιμα λευκή.

Πίνακας περιεχομένων

List Of Figures

List Of Tables

Abbreviations

BM	Battery Model
BMS	Battery Management System
CC	Coulomb Counting
DDM	Data Driven Methods
ECM	Equivalent Circuit Model
MAE	Mean Absolute Error
ML	Machine Learning
NN	Neural Network
OCV	Open Circuit Voltage
RF	Reward Function
RL	Reinforcement Learning
RMSE	Root Mean Square Error
SAC	Soft Actor Critic
SoC	State of Charge
SoH	State of Health

1 *Εκτενής Περίληψη*

Εισαγωγή

Ο σκοπός της παρούσας εργασίας είναι η προσομοίωση της λειτουργίας μπαταριών ιόντων λιθίου (Li-ion) με βασικό σκοπό τον υπολογισμό της φόρτισης τους την κάθε χρονική στιγμή (State of Charge). Η μεθοδολογία είναι ένας συνδυασμός παραδοσιακής μοντελοποίησης ενός συστήματος με την χρήση ενός μαθηματικού μοντέλου και μεθόδων ενισχυτικής μάθησης. Το τελικό μοντέλο αποτελείται από ένα ισοδύναμο μαθηματικό σύστημα για την μπαταρία (Equivalent Circuit Model, ECM - Μοντέλο Ισοδύναμου Κυκλώματος) και ένα νευρωνικό δίκτυο που έχει εκπαιδευτεί μέσω ενισχυτικής μάθησης (Reinforcement Learning, RL) το οποίο επιδρά πάνω στο φυσικό μοντέλο αλλάζοντας τις παραμέτρους του (Model Calibration). Η βασική ιδέα είναι ότι καθώς οι παράμετροι του μοντέλου δεν είναι γνωστές και μπορεί να αλλάζουν κατά την λειτουργία ο υπολογισμός τους γίνεται από το νευρωνικό δίκτυο RL το οποίο έχει εκπαιδευτεί πάνω σε πραγματικά πειραματικά δεδομένα.

Μεθοδολογία

1. Επιλογή Μοντέλου Μπαταρίας

Ένα από τα πλέον χρησιμοποιούμενα μοντέλα μπαταρίας στη βιβλιογραφία είναι το 2nd Order Battery ECM (Equivalent Circuit Model) το οποίο φαίνεται στο παρακάτω σχήμα

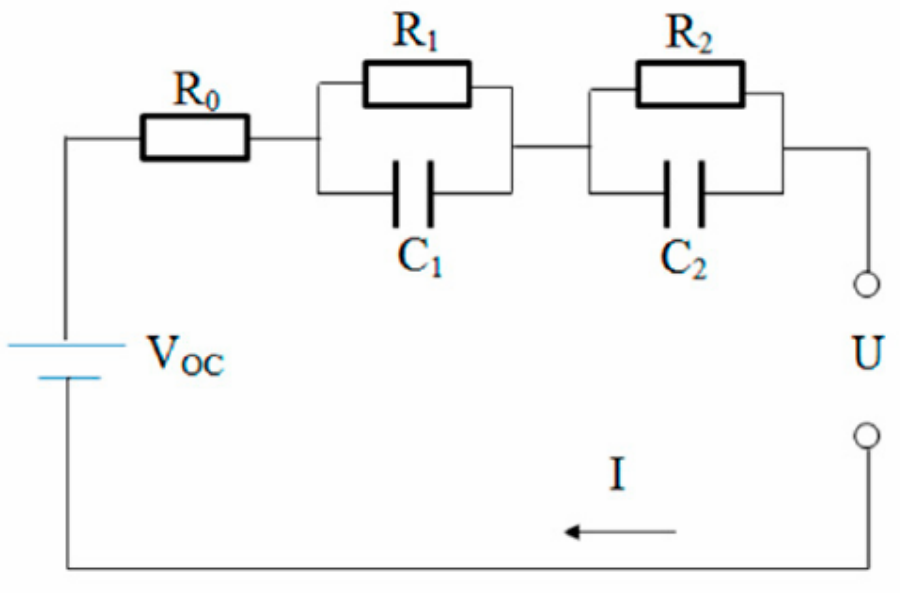


Figure 1: ECM Circuit

Το μοντέλο αυτό χρησιμοποιείται για τον υπολογισμό της V_{oc} (Open Circuit Voltage) η οποία είναι η τάση της μπαταρίας όταν αυτή δεν είναι συνδεδεμένη σε κάποιο κύκλωμα. Οι αντιστάσεις και πυκνωτές αποτελούν ένα απλοποιημένο μοντέλο των φυσικοχημικών διεργασιών μέσα στη μπαταρία (εσωτερική αντίσταση, υστέρηση και πόλωση). Η V_{oc} μπορεί να μετρηθεί πειραματικά αν η μπαταρία αποσυνδεθεί, αφεθεί χωρίς φορτίο για κάποιο χρονικό διάστημα και μετά μετρηθεί ξανά η τάση της.

2. Επιλογή μεθόδου προσέγγισης του SoC

Γνωρίζοντας τη V_{oc} από το παραπάνω σύστημα μπορούμε να υπολογίσουμε τη φόρτιση της μπαταρίας (SoC) επειδή υπάρχει συσχέτιση μεταξύ των δύο (καμπύλη SoC - OCV). Η πραγματική σχέση τους είναι πολύπλοκη και εξαρτάται και από τον τύπο της μπαταρίας. Γενικότερα υπάρχουν διάφορες προσεγγίσεις που μπορούμε χρησιμοποιήσουμε. Μία από αυτές είναι η προσέγγιση της V_{oc} σαν ένα πολυώνυμο της SoC. Μπορούμε επίσης αν θέλουμε να εισάγουμε και την θερμοκρασία σαν παράμετρο. Μία προσέγγιση είναι η παρακάτω

$$V_{oc} = p_1(SoC) + p_2(T)$$

όπου p1 και p2 είναι πολυώνυμα της φόρτισης και θερμοκρασίας αντίστοιχα, παράδειγμα

$$p_1(SoC) = a_n SoC^n + a_{n-1} SoC^{n-1} + \cdots + a_1 SoC + a_0$$

Έχοντας τα παραπάνω μπορούμε από μετρήσεις τάσης και ρεύματος στα άκρα μιας πραγματικής μπαταρίας να υπολογίσουμε τη φόρτιση.

3. Υπολογισμών των παραμέτρων του μοντέλου

Σε αυτό το σημείο για να γίνουν οι παραπάνω υπολογισμοί πρέπει να έχουμε υπολογίσει πειραματικά τις διάφορες παραμέτρους του μοντέλου δηλαδή τις αντιστάσεις και πυκνωτές (R_o, R_1, R_2, C_1, C_2) του ECM καθώς και όλες τις σταθερές του πολυωνύμου που επιλέξαμε στο βήμα 2 για την προσέγγιση του SoC. Αυτό γενικά είναι μία πολύπλοκη διαδικασία, απαιτεί πολλές πειραματικές μετρήσεις και οι τιμές που υπολογίζονται είναι σταθερές, οι οποίες όμως μπορεί και να αλλάζουν κατά το συνολικό χρόνο ζωής μία πραγματικής μπαταρίας.

Σε αυτό το σημείο η μεθοδολογία που παρουσιάζεται προτείνει την αντικατάσταση του βήματος αυτού με την χρήση ενός νευρωνικού δικτύου το οποίο για κάθε τιμή τάσης, ρεύματος και θερμοκρασία παράγει ως έξοδο τις παραμέτρους αυτές. Με κατάλληλη εκπαίδευση το μοντέλο αυτό επιλέγει κατάλληλα τις παραμέτρους προσπαθώντας να ελαχιστοποιήσει το σφάλμα της τιμής του SoC που υπολογίζεται.

4. Εκπαίδευση με Ενισχυτική Μάθηση

Όπως περιγράφηκε παραπάνω οι παράμετροι εκτιμώνται από ένα νευρωνικό δίκτυο που έχει εκπαιδευτεί μέσω ενός αλγόριθμου ενισχυτικής μάθησης. Το μοντέλο αυτό χρησιμοποιεί ένα dataset το οποίο περιέχει μετρήσεις για V, I, T και SoC_{real} που έχουν μετρηθεί κατά τη λειτουργία μίας μπαταρίας. Δηλαδή κατά τη διάρκεια ενός

πειράματος μετράμε την τάση και το ρεύμα που δίνει η μπαταρία σε ένα φορτίο, την θερμοκρασία της και την φόρτιση της. Η φόρτιση (SoC_{real}) αρκεί να προσδιοριστεί με κάποιο πειραματικό τρόπο κατά το ίδιο πείραμα ή ακόμα να είναι η έξοδος ενός άλλου νευρωνικού δικτύου το οποίο έχει εκπαιδευτεί μέσω μηχανικής μάθησης (το οποίο για εισόδους V, I, T δίνει σαν έξοδο ένα SoC) στα αντίστοιχα δεδομένα. Χρησιμοποιώντας το μοντέλο των βημάτων 2 και 3 μαζί με τα πειραματικά δεδομένα εκπαιδεύουμε το μοντέλο ενισχυτικής μάθησης με βάση την παρακάτω διαδικασία

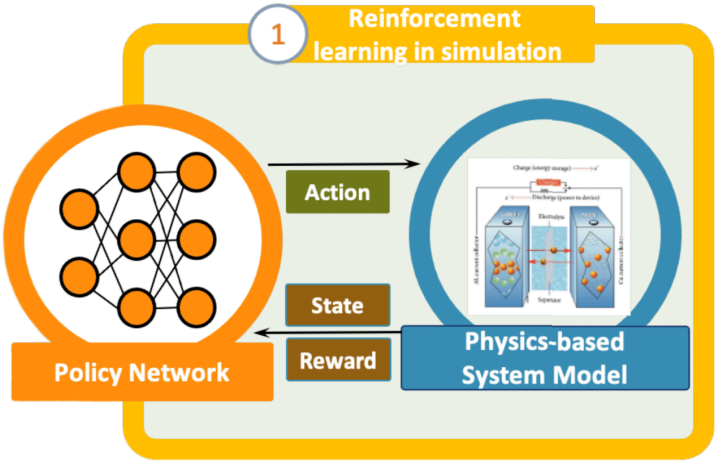


Figure 2: Neural Network - ECM Interaction In Training

1. Ορίζουμε ένα action και state space για τον νευρωνικό δίκτυο που θέλουμε να εκπαιδεύσουμε
2. Με είσοδο μετρήσεις V, I, T αυτό μας προτείνει κάποιες παραμέτρους για το μαθηματικό μοντέλο
3. Με τις ίδιες μετρήσεις V, I, T και τις παραμέτρους που έχει εκτιμάει το νευρωνικό δίκτυο λύνουμε το μαθηματικό μοντέλο και υπολογίζουμε ένα SoC_{calc}
4. Συγκρίνουμε το SoC_{calc} με το SoC_{real} που έχουμε ήδη για τη διαδικασία (από πειραματικές μετρήσεις) και υπολογίζουμε ένα reward function.
5. Ανατροφοδοτούμε το αποτέλεσμα του reward function στον αλγόριθμο ενισχυτικής μάθησης ο οποίος ρυθμίζει το νευρωνικό δίκτυο προσπαθώντας να επιλέξει παραμέτρους που ελαχιστοποιούν την διαφορά αυτών των δύο τιμών του SoC

6. Συνεχίζουμε για όλες τις τιμές των dataset που έχουμε. Τελικά το νευρωνικό δίκτυο για δεδομένες τιμες V, I και T μαθαίνει να επιλέγει τις τιμές του φυσικού μοντέλου έτσι ώστε η τιμή SoC_{calc} να προσεγγίζει την πραγματική.

4. Πραγματική Λειτουργία

Μετά την εκπαίδευση χρειαζόμαστε μόνο το μαθηματικό μοντέλο (2nd Order ECM) μαζί με το μοντέλο RL που έχουμε εκπαιδεύσει. Στην πραγματική λειτουργία έχουμε τα παρακάτω βήματα

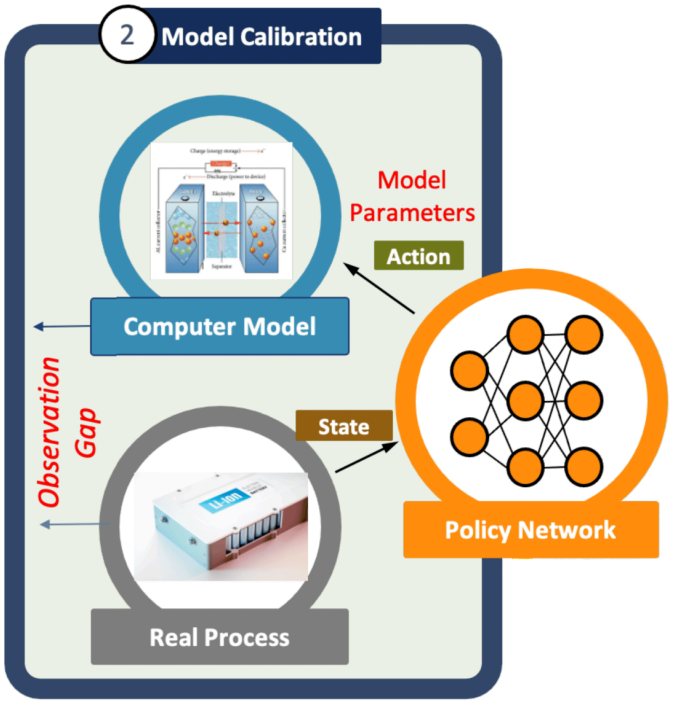


Figure 3: Deployed Neural Network - ECM

1. Κατά την λειτουργία της μπαταρίας παίρνουμε πραγματικές μετρήσεις τάσης, ρεύματος και θερμοκρασίας (V, I, T)
2. Με είσοδο τις τιμές V, I, T για κάποια χρονική στιγμή το νευρωνικό δίκτυο μας δίνει ως έξοδο παραμέτρους για το φυσικό μοντέλο. Οι παράμετροι αυτές όπως αναφέρθηκε και παραπάνω είναι τιμές των αντιστάσεων, πυκνωτών και των υπόλοιπων παραμέτρων του συστήματος.

3. Χρησιμοποιώντας τις παραμέτρους αυτές του νευρωνικού δικτύου μαζί με τις πραγματικές μετρήσεις V, I, T μπορούμε να λύσουμε το μοντέλο και να υπολογίσουμε την κατάσταση φόρτισης της μπαταρίας
4. Δεδομένου ότι το νευρωνικό δίκτυο έχει εκπαιδευτεί σωστά η τιμή αυτή θα πρέπει να συγκλίνει στην πραγματική τιμή της φόρτισης της μπαταρίας

Πλεονεκτήματα

Κάποια από τα πλεονεκτήματα της μεθόδου αυτής είναι τα παρακάτω

1. Μπορεί να χρησιμοποιηθεί με ένα μαθηματικό μοντέλο όπου αυτό κρίνεται σκόπιμο, για παράδειγμα σε προβλήματα που ήδη υπάρχει δουλειά μαθηματικής μοντελοποίησης ενός συστήματος από το παρελθόν
2. Συνδυάζει την παραδοσιακή μοντελοποίηση με μεθόδους Ενισχυτικής Μάθησης το οποίο μπορεί να βελτιώσει την απόδοση μαθηματικών μοντέλων που ήδη υπάρχουν.
3. Το μοντέλο ενισχυτική μάθησης (RL) δουλεύει παράλληλα και αλλάζει τις τιμές του φυσικού μοντέλου (ECM) κατά την λειτουργία (Online Calibration). Αυτό μπορεί να οδηγήσει σε απλοποιήσεις του φυσικού μοντέλου αφού πλέον, δεδομένου ότι οι παράμετροι αλλάζουν, αυτό μπορεί να προσεγγίσει μη γραμμικές συμπεριφορές που πριν ίσως δεν ήταν δυνατόν ή χρειάζονταν μία ακόμα πιο πολύπλοκη μαθηματική περιγραφή.
4. Σε κάποιους τύπους προβλημάτων ο συνδυασμός των μοντέλων μπορεί να βελτιώνει τη συνολική συμπεριφορά αφού το νευρωνικό δίκτυο κάνει calibration του φυσικού μοντέλου ενώ αυτό με τη βοήθεια των εξισώσεων μπορεί να προσεγγίζει καλύτερα τη δυναμική του συστήματος σε συγκεκριμένα σημεία λειτουργίας.

2 Introduction

2.1 Overview

Today there is an increased reliance on batteries across different types of applications, such as electric vehicles and renewable energy storage systems that requires an efficient management of the battery's health and performance. A key metric in battery management systems (BMS) is the State of Charge (SoC) which is a measure of the energy remaining in the battery that can be used by our application. An accurate estimation of SoC plays a vital role in optimizing utilization of the battery system and extending the battery's lifespan. Different ways of SoC estimation have been proposed, which are generally divided into two big categories, the first one involving the utilization of physics-based battery models and the second various machine learning methods that are trained on experimental datasets. Equivalent Circuit Models (ECMs) for batteries are commonly employed for this purpose due to their simplicity but their static parameters often fail to capture the changing operating conditions of a real system. This thesis proposes a novel approach that integrates reinforcement learning (RL) to dynamically adjust the parameters of the ECM in real time and trying to improve the accuracy of SoC estimation under various conditions.

2.2 Motivation

Traditional methods for SoC estimation rely on physics-based models of batteries that mostly have static parameters that have been estimated during the model identification. These parameters in general may not adequately account for variations in the behavior of the battery caused by temperature changes, aging or nonlinearities that are inherent in the process. Different techniques have been implemented in order to account for the differences in battery behavior, such as Kalman filters, but these methods often struggle with the model nonlinearities, require manual tuning and they are computationally more intensive if employed in a real time scenario.

Reinforcement learning techniques offer an alternative approach that allows the system to learn and adapt autonomously. Using them, a neural network can be trained that can update the parameters of the physics-based model online eliminating the need for manual calibration or the recalibration to fit different scenarios. This can offer an attractive alternative for commercial application where the battery is expected to work under different working conditions, such as warm and cold climates and across the different stages of the battery's life.

2.3 Problem Statement

Current battery management systems have limitations in accurately estimating the State of Charge (SoC) due to either the inability of the models to capture the behavior for different working conditions or the inherent non linearities of the battery system without becoming too complex and computationally intensive. In general

1. Static model parameters can lead to increasing estimation errors under different scenarios of operation.
2. Manual recalibration of the models can be time consuming and also impractical for commercial applications.

3. Other techniques can struggle when handling model nonlinearities (Kalman filters) or be computationally intensive for a real time application (Electrochemical battery models).

This thesis addresses these limitations by developing a reinforcement learning based neural network that adjusts the model parameters of a Li-ion battery ECM (Equivalent Circuit Model) attempting to combine traditional modeling with learning techniques thus improving the accuracy and robustness of the overall model.

2.4 Objectives

The primary objectives are the following:

1. Select a battery model, from the ones available in the literature, that can adequately capture the behavior of Li-ion batteries and be used for the estimation of the battery's SoC.
2. Develop a reinforcement learning methodology that can be used to train a neural network capable of online calibration of the battery model.
3. Train the neural network using datasets of real Li-ion batteries under different working conditions.
4. Validate the proposed approach through simulations of different working conditions based on real data.
5. Explore how slight variations of the selected physical model can affect the resulting accuracy.

3 Literature Review

3.1 State of Charge (SoC) Estimation Method

Battery Management Systems (BMS) are utilized in order to enhance battery lifetime, reliability and efficiency [1]. Their main objective is, amongst other things, to monitor the State of Health (SoH) which estimates the battery' condition and State of Charge (SoC) which is an important parameter that measures the remaining capacity on a battery. One drawback when estimating these parameters is that they cannot be measured directly from the available sensors (in an electric vehicle for example) and are also dependent on many factors such as current, voltage, temperature, aging etc. which raises the need to develop accurate estimation algorithms.

There are a number of different approaches for estimating the State of Charge such as Coulomb Counting (CC), open-circuit voltage (OCV) methods, electrochemical methods, machine learning based methods, Kalman Filter based methods and others [2]. Some of the methods such as Coulomb Counting or machine learning methods do not need to explicitly define a battery model, while others can use an equivalent model (models based on ECM or Kalman filtering) or a model that describes the inner battery processes (electrochemical models). Different methods have different computational complexities either upfront (learning or hybrid methods) or during the solution of the models (electrochemical methods).

One popular approach is the modeling of the battery's State of Charge through its relationship to the Open Circuit Voltage (SoC - OCV curve) which is an internal characteristic of a specific type of battery. This approach can be generalized to

different batteries but it should be noted that modeling and parameter estimation should be repeated for different types. The SoC - OCV curve is generally a nonlinear monotone function and a good estimation of this curve leads to more accurate models. The estimation of this curve is generally involved and needs to find a good function to approximate the curve, parameter identification and also a good way to model the Open Circuit Voltage of the battery as this quantity is also not directly measured to a battery that is in use. (To measure OCV experimentally for a specific battery state we need to disconnect the battery and let it settle for some time and then measure the voltage again). Several functions have been proposed that model the SoC - OCV curve that include polynomial functions of various degrees, logarithmic, exponential or a combination of them. The function becomes even more complex if we need to include other variables such as battery temperature or aging.

3.2 Battery Models

The battery models that exist in the literature in general can be divided into the following categories: physics-based electrochemical models, electrical equivalent circuit models (ECMs) and data driven models utilizing artificial intelligence algorithms such as neural networks and support vector machines [3].

3.2.1 Physics-based electrochemical models

The most mature of the physics based electrochemical models is the single-particle model where the concentration distribution in the electrode is done through a single particle. Though simple this model has a relatively low accuracy which has led to the extension of it by other models, for example SPMe that utilize more complex modeling as well as partial differential equations to describe parts of the electrochemical process. Other models such as the P2D have also been proposed that treat the anode and cathode of the cell as porous electrodes and the spaces between particles are filled with electrolyte. This model utilizes a system of coupled partial differential equations PDEs which makes it necessary to introduce simplifications from the perspective of engineering practice. Even more complex models have been proposed, for example a P2D electrochemical-thermal-capacity coupled model that

accounts for several mechanical and electrochemical factors. In general all these models are difficult to apply because they have a large number of unknown variables and are also computationally intensive to solve especially in a real time scenario. They are also highly sensitive to accurate estimation of the parameters which in many cases are internal to the battery and hard to identify which can lead to simulation results that are not ideal.

3.2.2 Equivalent Circuit Models

The electrical equivalent circuit models model the battery behavior using electrical components [4]. They have attracted interest due to their simplified structure with respect to other approaches. One of the simplest such models is the Rint model which approximates the battery with a real voltage source with a resistor in series. An extension of this model is the first order ECM which also includes an additional resistor and capacitor connected in parallel which aims at modelling the behavior during charging and discharging. A more complex approach is the second order ECM that employs an additional RC component that can better approximate the polarization and diffusion phenomena that happen during the battery's behavior. More complex approaches also exist that extend the model to higher orders.

3.2.3 Data Driven Methods

A variety of data driven methods (DDM) also exist that use neural networks and various learning algorithms to simulate the behavior of the battery. These methods in general have very good performance when it comes to the nonlinear phenomena without requiring a detailed knowledge of the underlying electrochemical system. When trained using comprehensive datasets can achieve high accuracy but this depends a lot on the quality and the quantity of the provided data. On the other hand they can struggle when approximating scenarios outside the dataset which can include different temperatures, loads or aging effects of a battery. In addition they can also be influenced by the training method and algorithm used which also need to be taken into account.

3.2.4 Hybrid Methods

As described above an approach where the SoC - OCV curve is modeled will require the selection of a complex function as well as the identification of a number of parameters used in it. This is an involved process and requires dedicated parameter identification methods also referred to as model calibration. The inferred model parameters often correspond to real physical quantities that are not directly observable and thus cannot be measured during operation. The relevance of the model calibration process in the final accuracy of the predictions while also making it robust to uncertainty in observations makes this process inherently challenging. This can introduce computational issues such as time consuming simulations, requirements for big datasets of high quality data and highly complex dynamic models. Also the existence of multiple solutions which increases with model complexity should also be taken into account.

Several methods have been proposed to tackle this problem . If a physics-based model already exists and is well founded on known underlying physical laws the available methods can include parameter inference using probabilistic or estimation approaches based on techniques from optimal control theory and statistics (for example Kalman Filters, extended Kalman Filters, particle filters, Bayesian inference methods and other). These methods have all been deployed in some form or another in real applications but still have limitations such as high processing power requirements or sensitivity to any inaccuracies in the modeled dynamics.

Data Driven approaches have also been proposed as an alternative method for calibrating physics-based models which results in a more probabilistic approach when solving the calibration problem. A common approach is to use supervised learning where a neural network is trained where a correlation is derived from observation and model parameters. These methods provide real time calibration possibilities but are also more sensitive to the absence of high quality data in the training datasets. Also in order to adapt to new scenarios, retraining of the neural network is necessary.

Recently developments in model-free reinforcement learning have achieved progress towards addressing control problems in essence effectively being successful in finding optima control policies. RL has proven effective in finding optimal control policies for nonlinear systems with dynamics that have a degree of uncertainty [5]. These methodologies give some flexibility to adapt to new scenarios while they can be employed in real time and can be utilized for the inference of the parameters of physics-based models. One approach is the formulation of the parameter estimation as a tracking problem where an agent is trained to keep the model response matching the observations [6].

4 *Approach of the Thesis*

In this work our goal is to train a Neural Network using Reinforcement Learning to learn to make optimal predictions of the parameters of a Physics-Based Model which in turn will be solved to calculate the State of Charge of the battery. After the training the models could be used to estimate the SoC during the operation. The general approach to the training process is summarized in the figure below.

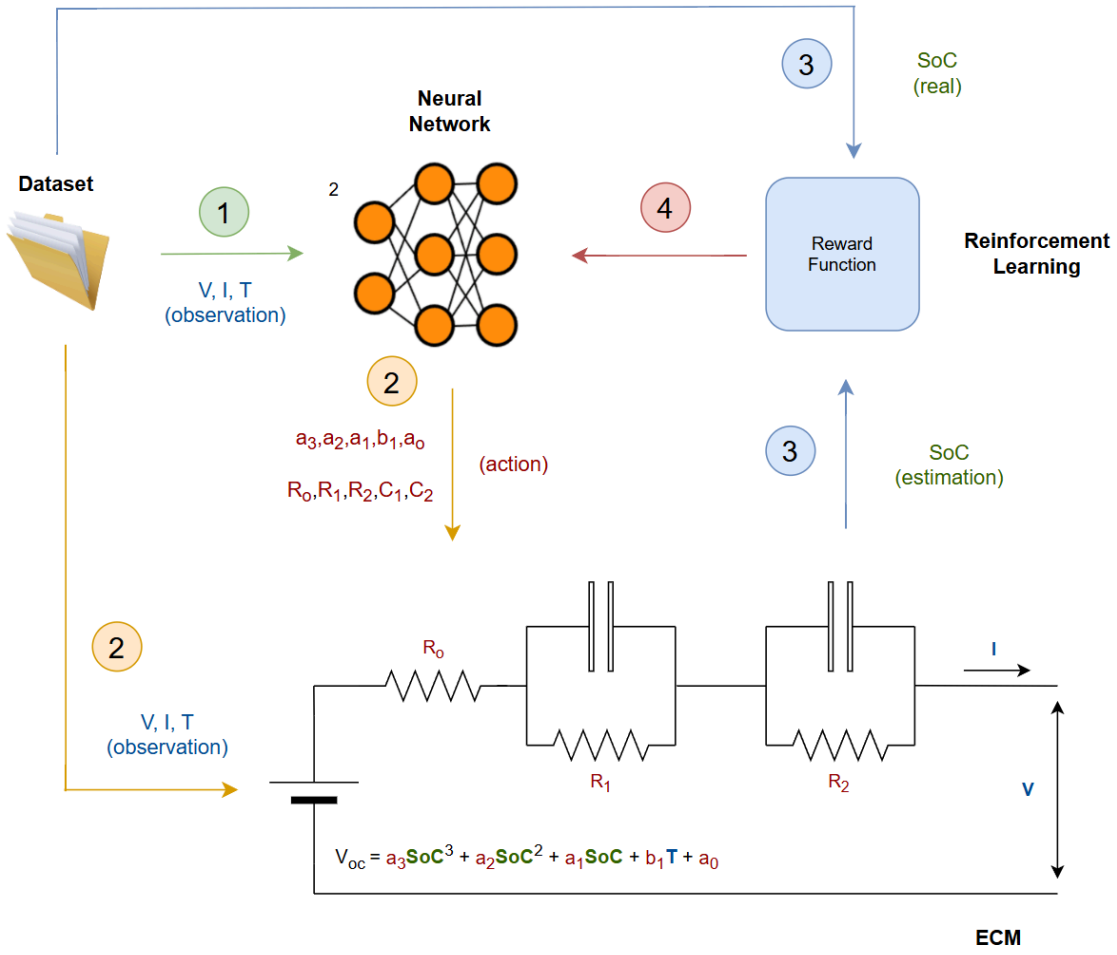


Figure 4: Training Approach

The steps are described here briefly and explained in detail in the following chapters

1. Measurements (observation) from the Dataset are given as inputs to the Neural Network (NN)
2. The Neural Network then estimates the parameters (action) of the Physics-Based model (the ECM in this case)
3. The ECM then, using the measurements and parameters, is solved to produce an estimation of the State of Charge of the battery at the current time step which is then, along with the real State of Charge from the measurements, inputted in a Reward Function
4. The result of the Reward Function is used by the Reinforcement Learning algorithm to calibrate the Neural Network and minimize the estimation error.

4.1 Selection of the Battery Model

The first step of the methodology is to select a battery model. The purpose of this model is to describe the internal state of the battery for the purposes of this methodology. This essentially means to give us a way to estimate the open circuit voltage (V_{oc}). This is the voltage measured when the battery is not connected to any circuit either to provide power or to get charged. In order to measure this voltage the battery needs to be disconnected and be left idle for some amount of time, for example half an hour, so it can settle to a voltage which represents the battery's Voc. In general this can vary with the individual characteristics and type of battery and it is not feasible to be measured in a real time scenario as it needs to be disconnected. So for our purposes we treat V_{oc} as a variable internal to the battery that cannot be directly measured.

In general V_{oc} is a variable directly connected to the State of Charge of the battery so a model needs to be used to estimate it from the available measurements that we have, i.e. voltage on the battery terminals, circuit current and temperature of the battery. A

number of physical models can be found in the literature that belong to broad categories, electrochemical models and equivalent circuit models. In general electrochemical models are complex models that model the internal structure of the battery and thus require a lot of parameters and possibly more complex mathematical representation which is unsuitable for real time calculations. Equivalent Circuit Models (ECMs) are considered a good middle ground between accuracy and complexity requiring only electrical components and can approximate the battery's behaviour adequately. A number of models also exist in this category from very simple one resistor models (Rint Model) to different orders of ECMs. The most commonly used model is the 2nd Order ECM which is the one used in the following analysis. This can adequately model the battery's internal voltage as well as some important dynamics of the battery such as polarization and hysteresis with the use of two additional capacitors and resistors.

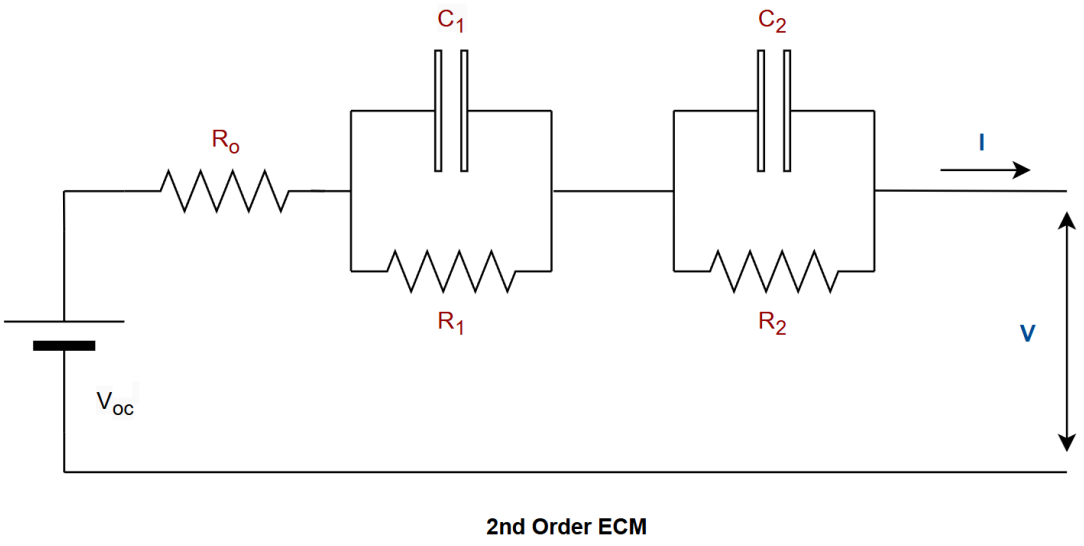


Figure 5: Circuit of Battery Model

As we can see in the picture we have two resistor/capacitor pairs modeling the dynamic behavior of the battery and an additional internal resistance. The value I and V_c are the battery measurements and V_{oc} is the open circuit voltage of the battery.

4.2 State of Charge Estimation Method

The model above gives us the means to calculate the battery's internal variable V_{oc} given the measurements I and V_c. An extra step is needed in order to be able to calculate the state of charge by adding an equation for it to the model. Several SoC approximations [7] can be found in the literature that connect the open circuit voltage V_{oc} with the current State of Charge. There is a plethora of models from different order polynomials to models also containing logarithmic or exponential factors. Since this is a non linear relationship the model can be chosen depending on the battery type and specific dataset in order to achieve the best fit. In our case the advantage of varying the model parameters in real time means that using a simpler model can capture this nonlinearity which is indirectly modeled inside the neural network trained using reinforcement learning.

There is also another factor that is taken into account and that is the battery's current temperature. In general the temperature is also included in the State of Charge model and can also vary depending on the approximation and it is common to also be modeled as a polynomial .

In our case a trade off needs to be made. Choosing a higher order model can improve accuracy but it does have diminishing returns as we add more parameters. On the other hand, more complex models can make the training of the RL based policy resource intensive, as more parameters need to be checked, increasing the dimensions of the action space. In our approach different SoC approximations have been studied, one simpler with a linear approximation for both SoC and temperature and one with a cubic approximation for SoC and a linear for temperature. More complex approaches can also be studied but in our case we achieve accurate results as the RL policy also captured part of the nonlinearities.

Model 1

$$V_{oc} = a_1 SoC + b_1 T + a_o$$

Model 2

$$V_{oc} = a_3 SoC^3 + a_2 SoC^2 + a_1 SoC + a_o$$

4.3 Mathematical Representation

The State of Charge estimation combined with the Equivalent Circuit Model of the battery described in the previous two chapters gives us a complete model that can be used in order to calculate the State of Charge of a battery based on the measured voltage, current and temperature. The combined model consists of the following system of algebraic and differential equations

For each capacitor resistor pair we have that

$$I = I_c + I_R = C \frac{dV_c}{dt} + \frac{V_R}{R}$$

$$V_R = V_c$$

Also from the whole circuit by applying the Kirchoff's law of voltages we have that

$$V_{oc} = V + IR_o + V_{R1} + V_{R2}$$

Two additions to the model have also been made in order to improve the overall behavior taking into account also the integration with the neural network

Firstly a limit function has been placed in the output of the model that bounds the SoC between 0 and 1, which are the values that it can normally take. This is done to remove possible peaks and undershoots created during the operation. Secondly taking

into account that the neural network changes the parameters of the model in real time this can in effect create discontinuities internally in the ECM circuit. The NN also has the ability to select any parameter within the action space which can have two effects. Firstly it can change a capacitor to a very low value in which the capacitor voltage (i.e. the capacitor voltage in the previous timestep) is higher than the one that could be achieved in the new circuit. Secondly it can change the capacitor to a very high value and because the capacitances are selected with high upper bounds, this can make the circuit very slow in reacting (since the capacitor will need to be charged requiring a lot of timesteps). The same applies in changes to the values of resistances. To solve these two problems a check has been applied so if the previous capacitor voltage is higher than the upper limit or lower than 70% of the max limit it is adjusted to 85% of the max value. These percentages have been empirically selected but work well in practice.

Taking into account all the above the equations of the complete model are given below

ECM

$$C_1 \frac{dV_{c1}}{dt} + \frac{V_{R1}}{R_1} = I$$

$$C_2 \frac{dV_{c2}}{dt} + \frac{V_{R2}}{R_2} = I$$

$$V_{R1} = V_{c1},\ V_{R2} = V_{c2}$$

$$V_{oc} = V\ + IR_o + V_{R1} + V_{R2}$$

SoC (one of the two equations)

$$V_{oc} = a_1 SoC + b_1 T + a_o$$

$$V_{oc} = a_3 SoC^3 + a_2 SoC^2 + a_1 SoC + a_o$$

Limit Functions

$$SoC = 1, SoC > 1$$
$$SoC = 0, SoC < 0$$

$$V_{c1} = 0.85IR_1, V_{c1} > IR_1 \; or \; V_{c1} < 0.7IR_1$$
$$V_{c2} = 0.85IR_2, V_{c2} > IR_2 \; or \; V_{c2} < 0.7IR_2$$

4.4 Parameter Description

As we can see the above model has parameters that need to be identified or estimated. The parameters are the following

2nd Order ECM: R_o, R_1, R_2, C_1, C_2
SoC Approximation: a_3, a_2, a_1, b_1, a_0 *or* a_1, b_1, a_0

so we have to identify 8 to 10 parameters depending on the SoC approximation we choose.

The ECM parameters are the values of the internal circuit components that model the behavior of the battery. In essence Ro is the internal resistance of the battery and the R1 - C1, R2 - C2 pairs model internal battery dynamics such as polarization and hysteresis. For example if the battery is in use and the charging cycle has started it takes some time for the internal voltages to settle because of the internal electrochemical phenomena that are here modeled by the capacitors.

The table below contains a range of typical values for the parameters based on their order of magnitude

Table 1: Parameter Ranges for Equivalent Circuit Models

	Minimum Value	**Maximum Value**
$\mathbf{R_o}$	0.001Ω	0.1Ω
$\mathbf{R_1}$	0.01Ω	0.5Ω
$\mathbf{R_2}$	0.05Ω	1Ω
$\mathbf{C_1}$	500F	5000F
$\mathbf{C_2}$	5000F	10000F
$\mathbf{a_3}$	-0.1	0.1
$\mathbf{a_2}$	-1	1
$\mathbf{a_1}$	0	5
$\mathbf{b_1}$	-0.5	0.5
$\mathbf{a_0}$	-10	10

Please note that the above are roughly the orders of magnitudes for the variables and are used as limits during the training process of the NN. They do not necessarily have any strict limitation and should be tinkered with during the modeling process to achieve the best results. For example a range could be increased slightly or a subset of the above ranges could be used to improve training time if it is judged that a reduced range is adequate for the modeling process.

4.5 Reinforcement Learning

As discussed in the previous chapters the above model is adequate, given the directly measurable inputs V, I and T to calculate the current State of Charge of the battery. The only missing part is the estimation of the parameters of the model. This traditionally is achieved by identification techniques in the laboratory and in general when the parameters are identified they remain fixed to specific values. A different approach that is used in the current thesis is instead to use a neural network trained

using reinforcement learning to learn a policy that itself will be responsible for dynamically updating the parameters of the model during operation. This offers the advantage that no identification is required as the neural network learns to propose optimal values of the parameters in real time while the parameters are updated at each timestep, not forcing us to adhere to a single set like in a more typical identification process.

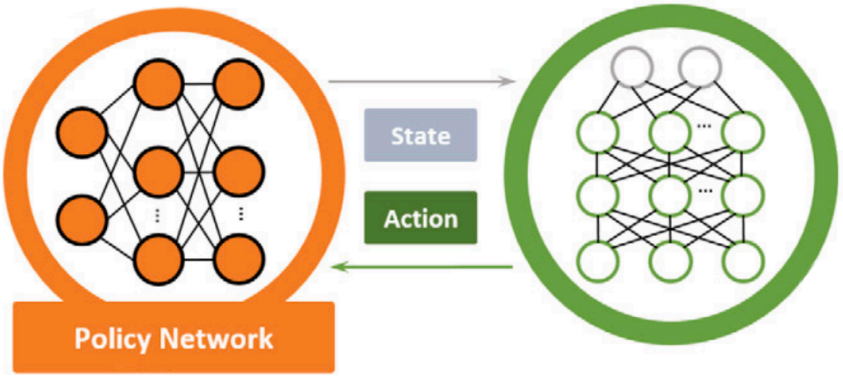


Figure 6: High Level Neural Network Training Process

As seen in the figure above a neural network is trained and an optimal action is learned. In our case a state is represented by the actual measurements that are acquired in real time or through a dataset, i.e. the voltage in the battery terminals, the current and the battery temperature. This NN in turn is responsible to produce an optimal action, which in our case is an estimation of the parameters described in the previous chapter, that will minimize the reward function. The goal of the Reward Function (RF) as defined in this problem is the minimization of the error of the calculated and the real State of Charge of the battery.

There is no specific requirement about the RL algorithm that is needed to be used in the problem so experimentation with different algorithms that meet the problems criteria can also be done. In our case a Soft Actor Critic (SAC) algorithm has been used that has the following advantages

1. Encourages the agent to explore more by adding an entropy term in the reward function thus reducing the chance to get stuck in local minima
2. Increases training efficiency by using a replay buffer that keeps past training states
3. Has improved critic performance by using two separate neural networks mitigating overestimation bias.

In any case though other RL algorithms could also achieve similar results so this is open for experimentation.

4.6 Description of the Methodology

Integrating all the above we have the final system that can be used for the estimation of the State of Charge of the battery. The first step that needs to be done is the training of the NN by the RL algorithm so it learns the optimal policy for parameter estimation. In our approach we used two different SoC models (as described above) which results in systems with different numbers of parameters. Below we will describe one of the two systems but the approach is similar for both. It should be noted that for each change of the physical model, i.e. the ECM or the SoC approximation, a new NN must be trained.

Training

First we train the neural network. This process takes the most time and a quality dataset is necessary for good results. The process is a follows:

1. In this step the model starts from an initial action that is produced by the neural network, that is arbitrary in the beginning as the network is untrained and converges to the optimal values as the training continues. The input (observation) to the NN comes from the dataset which is the measurement values for voltage, current and battery temperature.
2. The model then generates a new range of parameters (action) that are all bound by the action space during the problem formulation. This essentially is the expected range for the parameters (described in the previous chapters).

Figure 7: Input from Dataset to Neural Network

3. The parameters (action) along with the measurements (observation) are then fed to the physics-based model i.e. the ECM and SoC approximation models which can now be solved arithmetically.

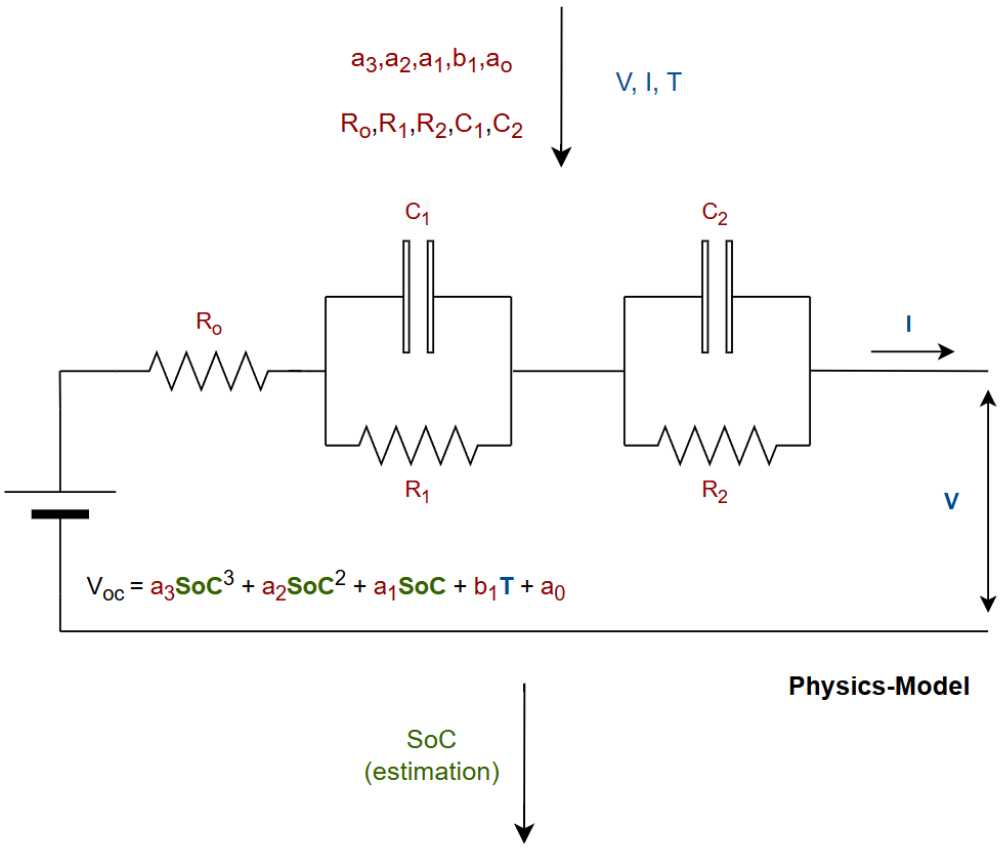


Figure 8: Inputs/Outputs of Physics-Based Model

4. The physics-based model then produces a new SoC estimation that corresponds to the current observation. This estimation is then compared with the real SoC that is included in the dataset, using a reward function and the result is used by the reinforcement learning algorithm to calibrate the neural network which learns the correct policy for minimizing the error between the two values.

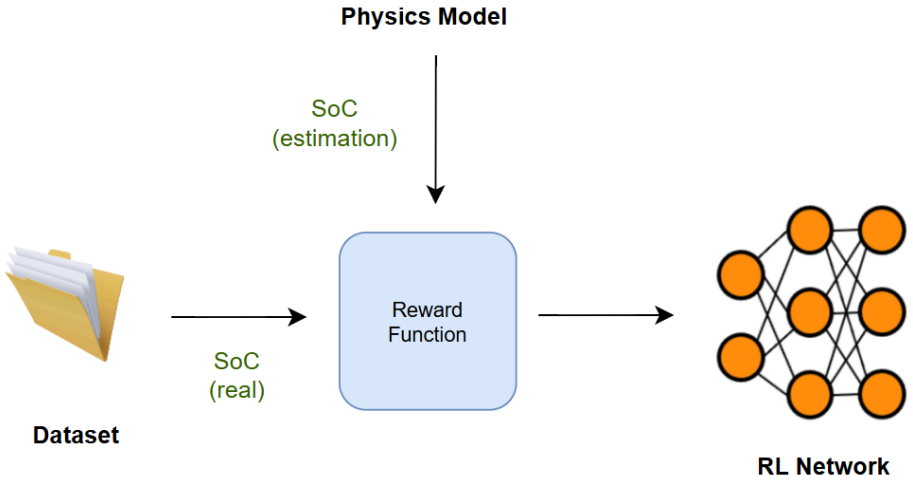


Figure 9: Reward Function of Reinforcement Learning Algorithm

Deployment

When the training process is completed a neural network is produced which has learned an optimal policy that attempts to reduce the SoC estimation error of the ECM. In general this policy is the best one found during training that minimizes this error. That said there could be different policies that produce similar results that could be learned by the neural network or even suboptimal policies if the model converges to possible local minima that may exist. The goal is to keep a model simple enough to

minimize this effect, as well as use RL algorithms that encourage exploration even if they have found a possibly good solution. Now that the NN has learned an optimal policy it can be used for a specific observation to give the best action (i.e. combination of parameters to be used in our case in the following way)

1. Each time new measurements are available, in our case the voltage, current and temperature of the battery, these values are used as an input to the NN to produce an optimal action, i.e. the parameters of the ECM.
2. These parameters along with the original measurements are used to solve the ECM and produce a new SoC estimation. It should be noted here that both the NN as well as the ECM work in combination to try and capture the behavior of the battery and in this sense both of these models contain part of the system's dynamics.
3. The ECM is solved and a prediction for the State of Charge of the current timestep is produced. This should be an optimal prediction with respect to the training of the Neural Network in the sense that its error from the real is minimized.

The figure below shows the process of using the model during a real time scenario. The NN now is the one that has been trained and along with the ECM can be used in real time as a battery is operating.

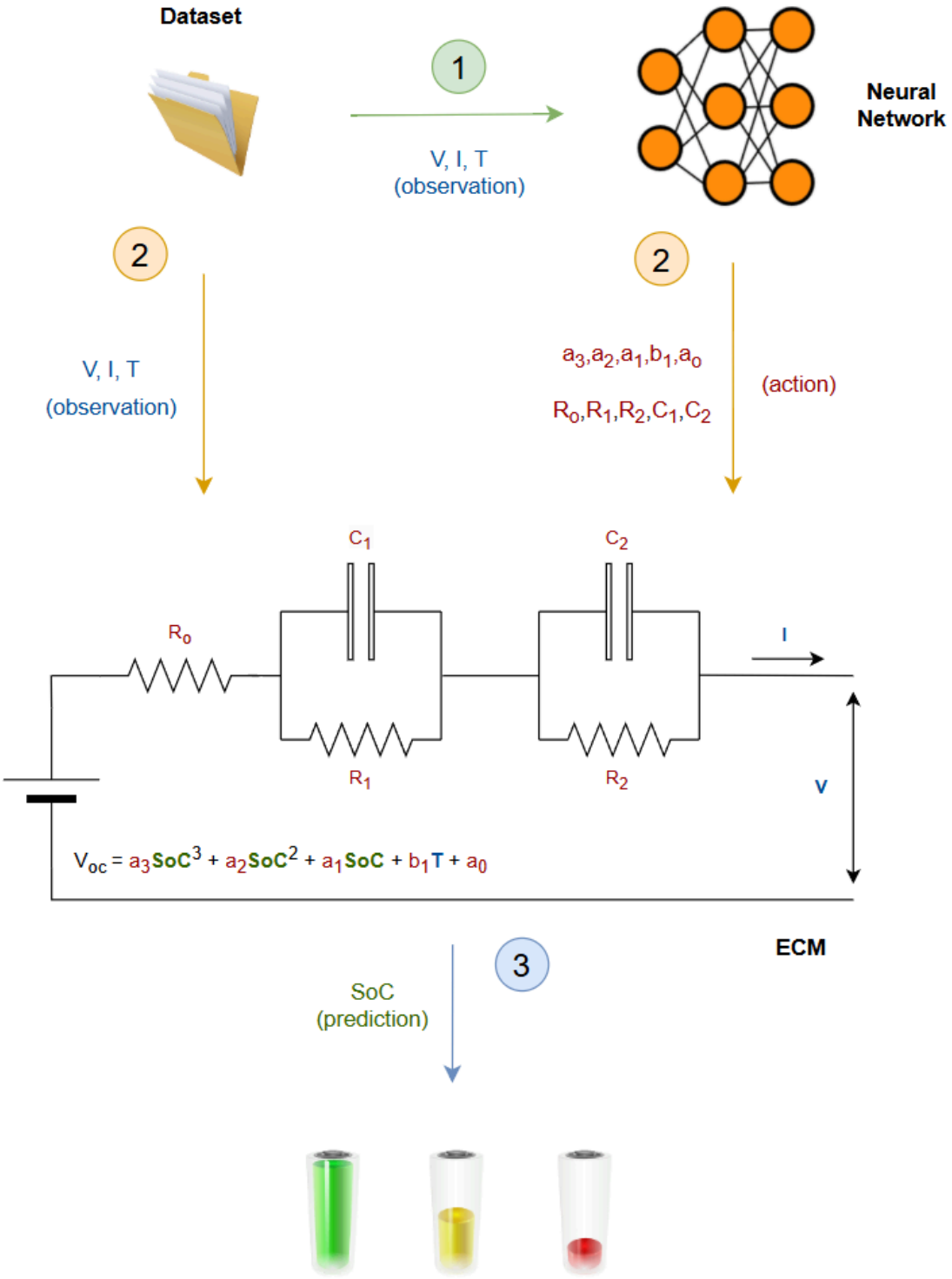


Figure 10: Model Deployment Methodology

In the two figures below we have a more high-level summary of the process. As a first step we train the NN using a dataset and solving the Physics-Based model, the output of which is used by the RL algorithm to calibrate the NN which learns the optimal policy.

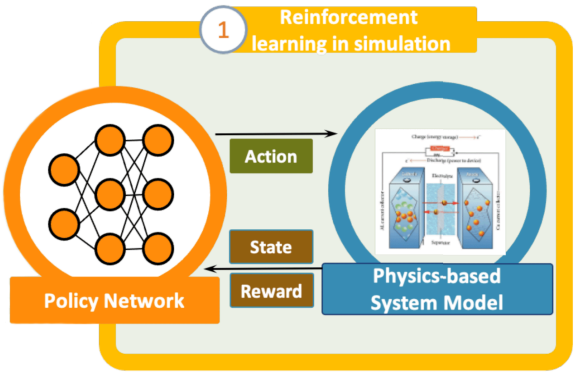


Figure 11: Neural Network - Physics Based Model Interaction In Training

After the training is completed we can deploy the network to an application in a real time environment. Here the NN is used as an optimal estimator of the best system parameters for a specific set of measurements. These measurements are then used with the parameters to solve the physics based model and get the estimation of the current State of Charge of the battery.

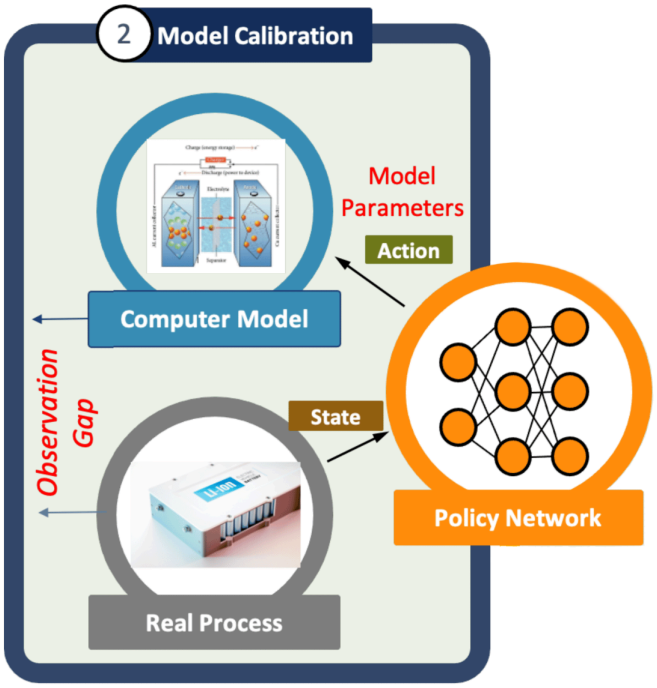


Figure 12: Neural Network - Physics Based Model In Deployment

5 Implementation

5.1 Datasets

5.1.1 Dataset Description

The datasets used [8] consist of 124 commercial lithium-ion batteries which have been charged/discharged till failure using fast charging. These lithium-ion cells were cycled in horizontal cylindrical fixtures on a 48-channel Arbin LBT potentiostat in a forced convection temperature chamber set to 30°C. The cells have a nominal capacity of 1.1 Ah and a nominal voltage of 3.3 V. There are also temperature measurements that have been performed by a Type T thermocouple. It should be taken into account that because of the nature of the experiment the temperature measurements could exhibit reduced accuracy since the thermal contact between the thermocouple and the cell may vary or the thermocouple could lose contract during some cycles.

The datasets are divided into three different groups each consisting of about 48 cells. Each group has a separate date denoting when the tests started. All data used were available as csv files and loaded into python during the training and validation phases.

Measurement Group 1

Experiment Information

1. All cells were cycled with one-step or two-step charging policies. The charging time varies from ~8 to 13.3 minutes (0-80% SOC). There are generally two cells tested per policy, with the exception of 3.6C (80%).
2. 1 minute and 1 second rests were placed after reaching 80% SOC during charging and after discharging, respectively.
3. We cycle to 80% of nominal capacity (0.88 Ah).
4. An initial C/10 cycle was performed in the beginning of each test.
5. The cutoff currents for the constant-voltage steps were C/50 for both charge and discharge.
6. The pulse width of the IR test is 30 ms.

Measurement Group 2

Experiment Information

1. All cells were cycled with one-step or two-step charging policies. The charging time is fixed at 10 minutes (0-80% SOC). There is generally only one cell tested per policy, with the exception of 4.8C(80%) (three cells).
2. We resumed 5 cells from the 2017-05-12 batch that didn't complete yet - 3.6C and 4.0C.
3. We cycle to 75% of nominal capacity (0.88 Ah).
4. 5 minute rests were placed both after reaching 80% SOC during charging and after discharging.
5. An initial C/10 cycle was performed in the beginning of each test.
6. The cutoff currents for the constant-voltage steps were C/50 for both charge and discharge.
7. The pulse width of the IR test is 30 ms.

Measurement Group 3

Experiment Information

1. All cells were cycled with two-step charging policies. The charging time fixed at 10 minutes (0-80% SOC). We test multiple cells per policy (3-8x per policy).
2. We cycle to 80% of nominal capacity (0.88 Ah).
3. Four 5-second rests were placed after reaching 80% SOC during charging, after the IR test, before discharging, and after discharging.

4. A final C/10 cycle was performed at 80% of nominal capacity.
5. The cutoff currents for the constant-voltage steps were C/20 for both charge and discharge.
6. The pulse width of the IR test is 33 ms.

Example files available for this dataset are the following (total 46 files of multiple cycles on different batteries)

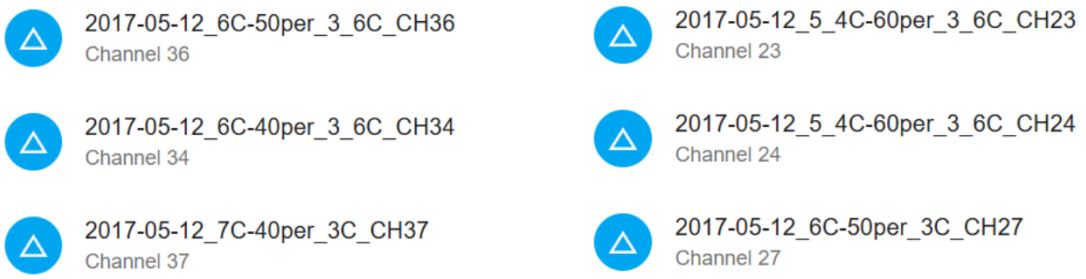
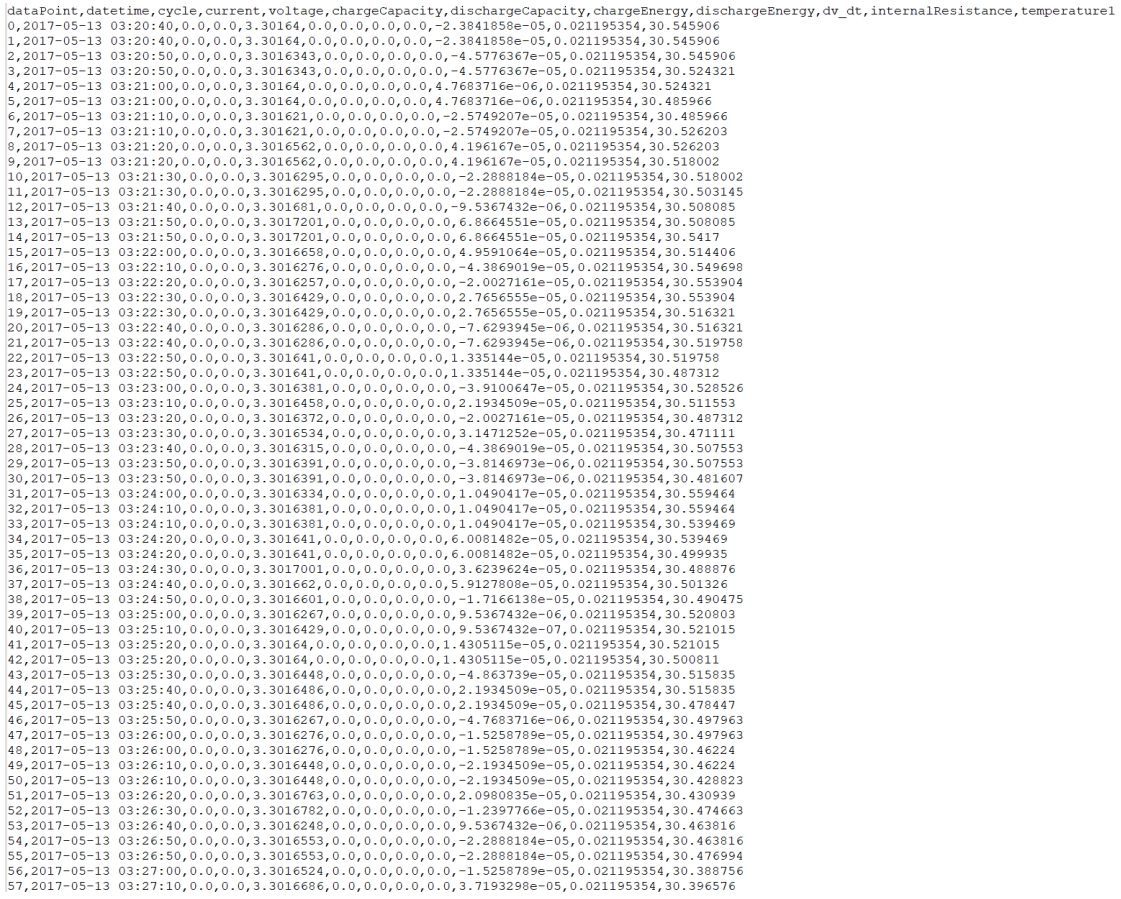


Figure 13: Example of Dataset Filenames

5.1.2 Dataset Overview

As we can see above the datasets have multiple charging/discharging cycles for multiple batteries. Also there are three groups that have different experimental designs with varying parameters. Moreover each group consists of multiple files each containing different load profiles for the battery (i.e. different Voltage and Current inputs during the charging and discharging process). This provides a rich dataset covering multiple different conditions for Li-ion batteries that in turn is expected to enhance the number of different real time scenarios that a model trained on these data can cover.



```
dataPoint,datetime,cycle,current,voltage,chargeCapacity,dischargeCapacity,chargeEnergy,dischargeEnergy,dv_dt,internalResistance,temperature1
0,2017-05-13 03:20:40,0.0,0.0,3.30164,0.0,0.0,0.0,0.0,-2.3841858e-05,0.021195354,30.545906
1,2017-05-13 03:20:40,0.0,0.0,3.30164,0.0,0.0,0.0,0.0,-2.3841858e-05,0.021195354,30.545906
2,2017-05-13 03:20:50,0.0,0.0,3.3016343,0.0,0.0,0.0,0.0,-4.5776367e-05,0.021195354,30.545906
3,2017-05-13 03:20:50,0.0,0.0,3.3016343,0.0,0.0,0.0,0.0,-4.5776367e-05,0.021195354,30.524321
4,2017-05-13 03:21:00,0.0,0.0,3.30164,0.0,0.0,0.0,0.0,4.7683716e-06,0.021195354,30.524321
5,2017-05-13 03:21:00,0.0,0.0,3.30164,0.0,0.0,0.0,0.0,4.7683716e-06,0.021195354,30.485966
6,2017-05-13 03:21:10,0.0,0.0,3.301621,0.0,0.0,0.0,0.0,-2.5749207e-05,0.021195354,30.485966
7,2017-05-13 03:21:10,0.0,0.0,3.301621,0.0,0.0,0.0,0.0,-2.5749207e-05,0.021195354,30.526203
8,2017-05-13 03:21:20,0.0,0.0,3.3016562,0.0,0.0,0.0,0.0,4.196167e-05,0.021195354,30.526203
9,2017-05-13 03:21:20,0.0,0.0,3.3016562,0.0,0.0,0.0,0.0,4.196167e-05,0.021195354,30.518002
10,2017-05-13 03:21:30,0.0,0.0,3.3016295,0.0,0.0,0.0,0.0,-2.2888184e-05,0.021195354,30.518002
11,2017-05-13 03:21:30,0.0,0.0,3.3016295,0.0,0.0,0.0,0.0,-2.2888184e-05,0.021195354,30.503145
12,2017-05-13 03:21:40,0.0,0.0,3.301681,0.0,0.0,0.0,0.0,-9.5367432e-06,0.021195354,30.508085
13,2017-05-13 03:21:50,0.0,0.0,3.3017201,0.0,0.0,0.0,0.0,6.8664551e-05,0.021195354,30.508085
14,2017-05-13 03:21:50,0.0,0.0,3.3017201,0.0,0.0,0.0,0.0,6.8664551e-05,0.021195354,30.5417
15,2017-05-13 03:22:00,0.0,0.0,3.3016658,0.0,0.0,0.0,0.0,4.9591064e-05,0.021195354,30.514406
16,2017-05-13 03:22:10,0.0,0.0,3.3016276,0.0,0.0,0.0,0.0,-4.3869019e-05,0.021195354,30.549698
17,2017-05-13 03:22:20,0.0,0.0,3.3016257,0.0,0.0,0.0,0.0,-2.0027161e-05,0.021195354,30.553904
18,2017-05-13 03:22:30,0.0,0.0,3.3016429,0.0,0.0,0.0,0.0,2.7656555e-05,0.021195354,30.553904
19,2017-05-13 03:22:30,0.0,0.0,3.3016429,0.0,0.0,0.0,0.0,2.7656555e-05,0.021195354,30.516321
20,2017-05-13 03:22:40,0.0,0.0,3.3016286,0.0,0.0,0.0,0.0,-7.6293945e-06,0.021195354,30.516321
21,2017-05-13 03:22:40,0.0,0.0,3.3016286,0.0,0.0,0.0,0.0,-7.6293945e-06,0.021195354,30.519758
22,2017-05-13 03:22:50,0.0,0.0,3.301641,0.0,0.0,0.0,0.0,1.335144e-05,0.021195354,30.519758
23,2017-05-13 03:22:50,0.0,0.0,3.301641,0.0,0.0,0.0,0.0,1.335144e-05,0.021195354,30.487312
24,2017-05-13 03:23:00,0.0,0.0,3.3016381,0.0,0.0,0.0,0.0,-3.9100647e-05,0.021195354,30.528526
25,2017-05-13 03:23:10,0.0,0.0,3.3016458,0.0,0.0,0.0,0.0,2.1934509e-05,0.021195354,30.511553
26,2017-05-13 03:23:20,0.0,0.0,3.3016372,0.0,0.0,0.0,0.0,-2.0027161e-05,0.021195354,30.487312
27,2017-05-13 03:23:30,0.0,0.0,3.3016534,0.0,0.0,0.0,0.0,3.1471252e-05,0.021195354,30.471111
28,2017-05-13 03:23:40,0.0,0.0,3.3016315,0.0,0.0,0.0,0.0,-4.3869019e-05,0.021195354,30.507553
29,2017-05-13 03:23:50,0.0,0.0,3.3016391,0.0,0.0,0.0,0.0,-3.8146973e-06,0.021195354,30.507553
30,2017-05-13 03:23:50,0.0,0.0,3.3016391,0.0,0.0,0.0,0.0,-3.8146973e-06,0.021195354,30.481607
31,2017-05-13 03:24:00,0.0,0.0,3.3016334,0.0,0.0,0.0,0.0,1.0490417e-05,0.021195354,30.559464
32,2017-05-13 03:24:10,0.0,0.0,3.3016381,0.0,0.0,0.0,0.0,1.0490417e-05,0.021195354,30.559464
33,2017-05-13 03:24:10,0.0,0.0,3.3016381,0.0,0.0,0.0,0.0,1.0490417e-05,0.021195354,30.539469
34,2017-05-13 03:24:20,0.0,0.0,3.301641,0.0,0.0,0.0,0.0,6.0081482e-05,0.021195354,30.539469
35,2017-05-13 03:24:20,0.0,0.0,3.301641,0.0,0.0,0.0,0.0,6.0081482e-05,0.021195354,30.499935
36,2017-05-13 03:24:30,0.0,0.0,3.3017001,0.0,0.0,0.0,0.0,3.6239624e-05,0.021195354,30.488876
37,2017-05-13 03:24:40,0.0,0.0,3.301662,0.0,0.0,0.0,0.0,5.9127808e-05,0.021195354,30.501326
38,2017-05-13 03:24:50,0.0,0.0,3.3016601,0.0,0.0,0.0,0.0,-1.7166138e-05,0.021195354,30.490475
39,2017-05-13 03:25:00,0.0,0.0,3.3016267,0.0,0.0,0.0,0.0,9.5367432e-06,0.021195354,30.520803
40,2017-05-13 03:25:10,0.0,0.0,3.3016429,0.0,0.0,0.0,0.0,9.5367432e-07,0.021195354,30.521015
41,2017-05-13 03:25:20,0.0,0.0,3.30164,0.0,0.0,0.0,0.0,1.4305115e-05,0.021195354,30.521015
42,2017-05-13 03:25:20,0.0,0.0,3.30164,0.0,0.0,0.0,0.0,1.4305115e-05,0.021195354,30.500811
43,2017-05-13 03:25:30,0.0,0.0,3.3016448,0.0,0.0,0.0,0.0,-4.863739e-05,0.021195354,30.515835
44,2017-05-13 03:25:40,0.0,0.0,3.3016486,0.0,0.0,0.0,0.0,2.1934509e-05,0.021195354,30.515835
45,2017-05-13 03:25:40,0.0,0.0,3.3016486,0.0,0.0,0.0,0.0,2.1934509e-05,0.021195354,30.478447
46,2017-05-13 03:25:50,0.0,0.0,3.3016267,0.0,0.0,0.0,0.0,-4.7683716e-06,0.021195354,30.497963
47,2017-05-13 03:26:00,0.0,0.0,3.3016276,0.0,0.0,0.0,0.0,-1.5258789e-05,0.021195354,30.497963
48,2017-05-13 03:26:00,0.0,0.0,3.3016276,0.0,0.0,0.0,0.0,-1.5258789e-05,0.021195354,30.46224
49,2017-05-13 03:26:10,0.0,0.0,3.3016448,0.0,0.0,0.0,0.0,-2.1934509e-05,0.021195354,30.46224
50,2017-05-13 03:26:10,0.0,0.0,3.3016448,0.0,0.0,0.0,0.0,-2.1934509e-05,0.021195354,30.428823
51,2017-05-13 03:26:20,0.0,0.0,3.3016763,0.0,0.0,0.0,0.0,2.0980835e-05,0.021195354,30.430939
52,2017-05-13 03:26:30,0.0,0.0,3.3016782,0.0,0.0,0.0,0.0,-1.2397766e-05,0.021195354,30.474663
53,2017-05-13 03:26:40,0.0,0.0,3.3016248,0.0,0.0,0.0,0.0,9.5367432e-06,0.021195354,30.463816
54,2017-05-13 03:26:50,0.0,0.0,3.3016553,0.0,0.0,0.0,0.0,-2.2888184e-05,0.021195354,30.463816
55,2017-05-13 03:26:50,0.0,0.0,3.3016553,0.0,0.0,0.0,0.0,-2.2888184e-05,0.021195354,30.476994
56,2017-05-13 03:27:00,0.0,0.0,3.3016524,0.0,0.0,0.0,0.0,-1.5258789e-05,0.021195354,30.388756
57,2017-05-13 03:27:10,0.0,0.0,3.3016686,0.0,0.0,0.0,0.0,3.7193298e-05,0.021195354,30.396576
```

Figure 14: Dataset File Internal Structure

Here we can see the structure of a single file of the above datasets. It is formatted as a CSV and provides the measurements for each experiment. From the above measurements in for our analysis we use the following cycle, current, voltage,

1. **dataPoint:** The experiment's current datapoint

2. **datetime:** The date and time for the specific measurement. Here we are mostly interested about the time which gives us the second that this measurement was taken. This is useful to find the total time that has elapsed between data points that is mainly needed during the simulation of the ECM when the differential equations of the model are solved.

3. **cycle:** This informs us of the specific cycle in the current experiment. As will be described below the cycle is used for differentiating the datasets into separate subgroups to be used as the training and test sets respectively.

4. **current:** The current supplied (during charging) or drawn (during discharging) from the battery. This is used as an input to the battery's Equivalent Circuit Model and

in a real time scenario can be the last measurement taken from a battery during its operation.

4. **voltage:** Similarly the voltage measured at the battery's terminals. This is also used as an input to the model and is the voltage that can be measured in real time from a battery.

5. **chargeCapacity:** The current chargeCapacity of the battery. This along with the dischargeCapacity represent the total State of Charge of the battery which is the model's output variable and the value that we are trying to estimate. This is a value that cannot be measured directly in a real time scenario.

6. **dischargeCapacity:** The current dischargeCapacity of the battery.

7. **temperature1**: The battery's current temperature. This is also used as an input to the model as different temperatures can affect its behavior.

5.1.3 Data Selection

This is a vast dataset that includes many different experiments each with multiple cycles. In order to use the whole dataset much processing power and memory is needed as well as very big processing times that a normal consumer grade pc may not suitable to handle in a reasonable time. Thus a selection of a subset of all the data has been made. This selection has been done using a broad range of cycles from different files during different conditions and from all the available datasets. The goal was to have a diverse selection covering multiple different cases in order to confidently validate the results of the methodology. In any case if the necessary resources are available in a company environment for example all data could be used to produce an even more versatile model.

In our case 7 different experiment files have been selected from across all the datasets groups where the batteries have been exerted to different load conditions, days and temperatures. From each file 300 cycles have been used for a total of about 2100 cycles. From these 80% (1680 cycles) have been used during the training phase and 20% (420 cycles) have been used as the test dataset for the validation. In order to have

a uniform distribution and because the battery's behavior is changing with temperature and aging the following process has been followed.

1. Load the data from all the dataset files (that represent a different experiment scenario).
2. Take the first 300 cycles from each file. (here we could expect for each scenario the later cycles to potentially have slightly different behavior due to aging).
3. Rearrange the data so the first cycles from files 1 to 7 come first then the second cycles etc. This ensures that we have a dataset where a battery has different load conditions in each subsequent cycle to better simulate a real world scenario.
4. Every 5th iteration, separate the data to a different group that is the test group so it is only used for validation and not training.

So in summary we would have the following structure

Training dataset

(cycle 1 from dataset files 1 to 7),
(cycle 2 from dataset files 1 to 7),
(cycle 3 from dataset files 1 to 7),
(cycle 4 from dataset files 1 to 7),
cycle 5 has been excluded and added to the test dataset
(cycle 6 from dataset files 1 to 7),
…

Test dataset

(cycle 5 from dataset files 1 to 7),
(cycle 10 from dataset files 1 to 7),
(cycle 15 from dataset files 1 to 7),
…

This ensures a more realistic battery usage scenario, shuffles the data so the training algorithm is exposed to all the different load conditions from the beginning and takes cycles for the test dataset uniformly from the start to the later stages of the experiments.

5.1.4 Exploratory Data Analysis

Below we have a plot of the dataset for 7 consecutive cycles each belonging to a different dataset. With blue we have the current that is applied to or drawn from the battery (depending if we are currently charging or discharging). Here positive currents denote that the battery is charging. With orange we have the voltage as measured in the battery terminals. With green we have the real State Of Charge as measured in the experiment (ranges from 0 to 1). As we can see, we have different load conditions that are exerted on the battery depending on the experiment selected, especially during the charging stage where different charging profiles have been implemented. After that we also include the temperature graph for the same experiments. The graph is in Kelvin degrees and we can see that more or less the temperature is within the range of 30 - 40 degrees Celsius. We can also see that the temperature is rising as the experiments progress. As also mentioned above the temperature might sometimes be inaccurate due to the way the measurement setup has been implemented.

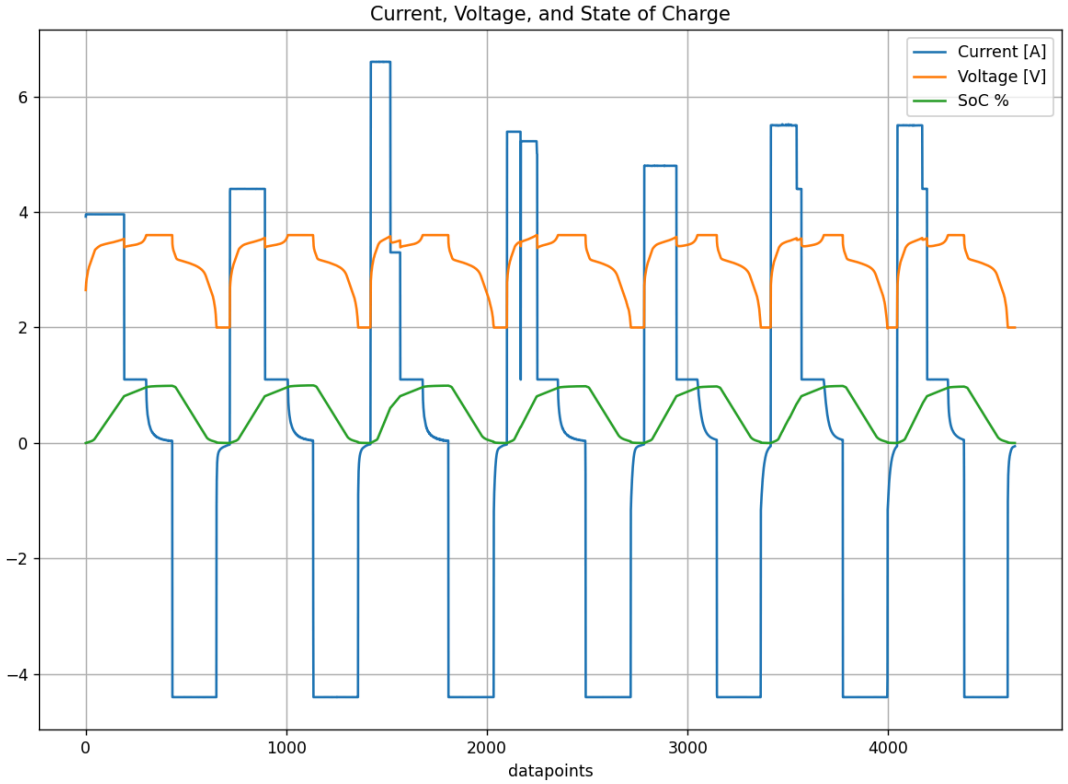


Figure 15: Battery's Current, Voltage and State of Charge

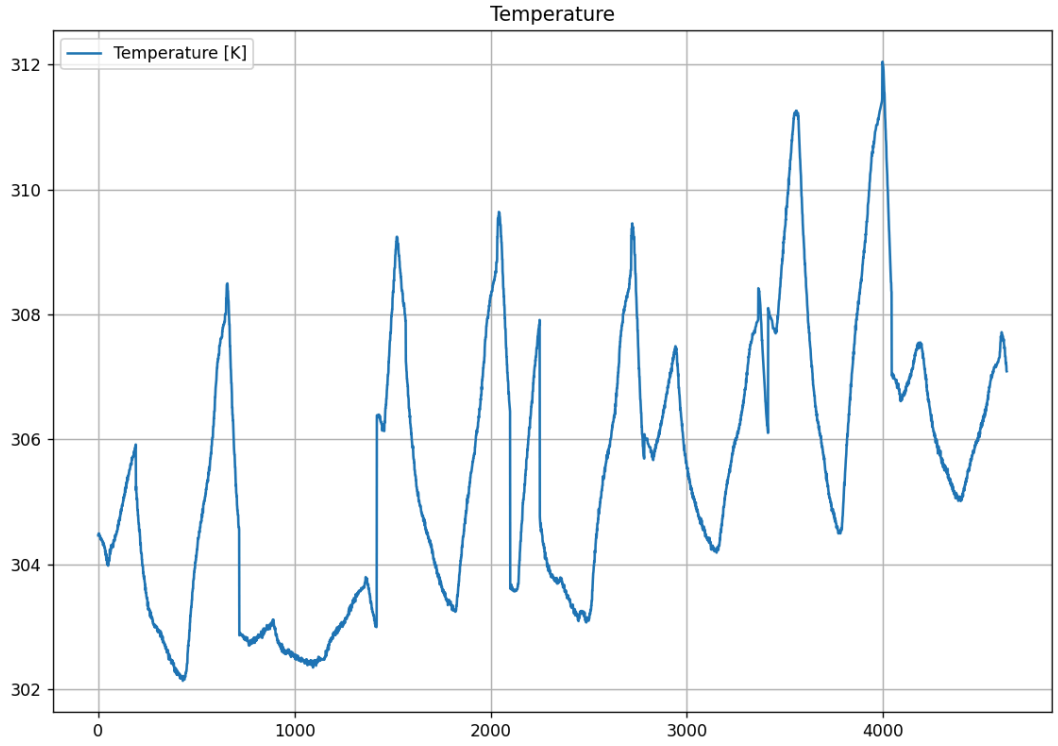


Figure 16: Battery's Temperature

Below we also present two measurement cycles from different datasets for a more detailed look. As we can see the cycles have a constant discharge current of about 4.4 A and a charging stage with varying currents with a maximum of about 6.6 A. We can see that the terminal voltage of the battery also changes, but as expected, much less than the current in the two cycles. Within each cycle the voltage drops or increases with respect to its SoC (denoting the voltage drop as a battery depletes). The SoC of each cycle also has differences between the cycles if plotted against each other but overall the curve appears similar.

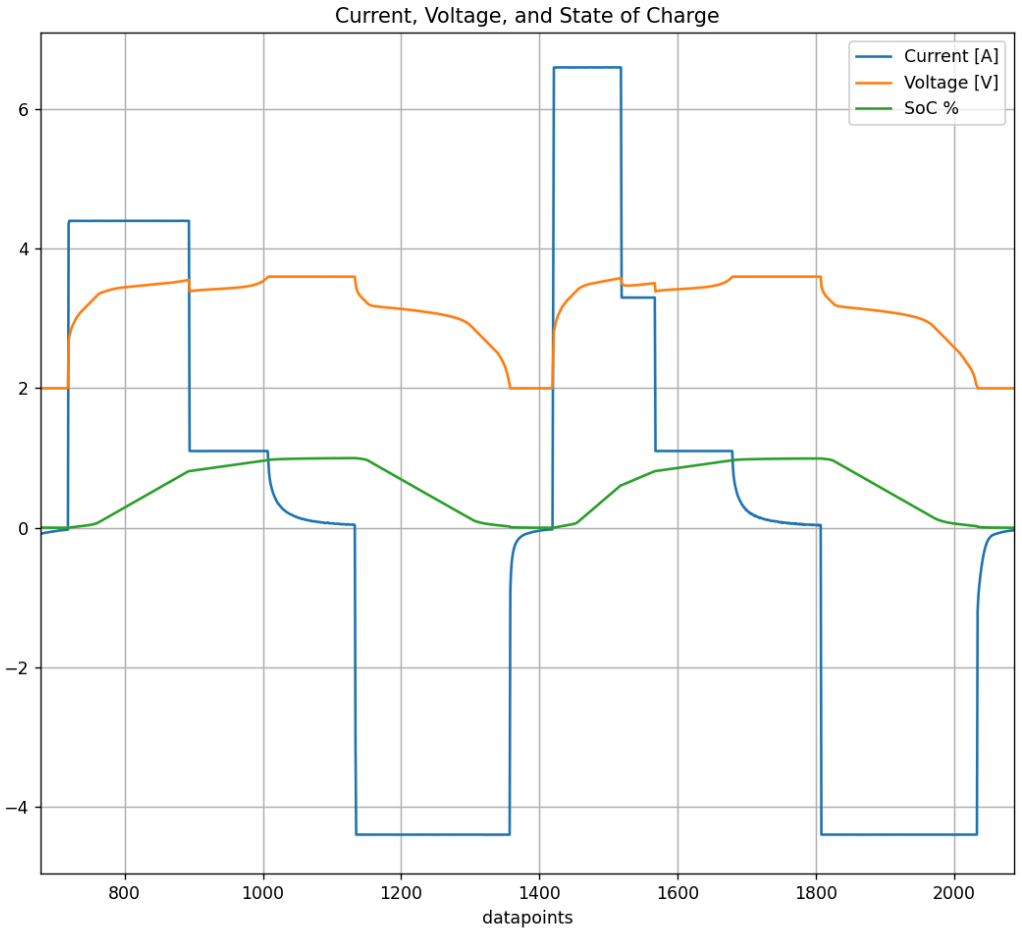


Figure 17: Battery's Current, Voltage and State of Charge (1 Cycle)

5.2 Software Framework Overview

For the training, testing as well as the data management python code has been used. Below we will provide an overview of the framework and libraries as well as a brief description of them. In general we can divide the code into four main functionalities: *data management, battery model simulation, reinforcement learning algorithm and dataset visualization*. Below we will give a brief description of each category and how they were all organized together.

5.2.1 Data Management

Data management is the first step that needs to be addressed as the datasets that have been described above need to be read and transformed to a usable format by the following algorithms. For this two main reusable functions used by the rest of the code have been created.

auxiliary.py

This is a collection of helper functions mainly responsible for loading the datasets from the csv files and converting them to dataframes using the **pandas** library. Depending on the use case separate functions have been created that load either the training or test datasets, all of them or selected cycles as needed. The need to load a selected cycle is mostly used for evaluation and visualization purposes. Here we also do some preprocessing of the input data. We first open the correct csv files, and we keep all the columns that we need by adding them to a pandas dataframe. We also convert Celsius to Kelvin for temperature, calculate the time elapsed from the beginning of the experiments and also calculate the time differential between two consecutive measurements that is needed during the calculations. Lastly we can do some dataset trimming like remove some regions where there are transitions between the experiments that we don't necessarily need during the training process because it can produce unneeded calculation artifacts.

```
import pandas as pd

used_dataset_names = [
    '2017-05-12_3_6C-80per_3_6C_CH1.csv',
    '2017-05-12_4C-80per_4C_CH5.csv',
    '2017-05-12_6C-60per_3C_CH29.csv',
    '2017-06-30_4_9C-27per_4_75C_CH24.csv',
    '2018-04-12_4_8C-80per_4_8C_CH1.csv',
    '2018-04-12_5C-67per_4C_CH7.csv',
    '2018-04-12_5C-67per_4C_CH42.csv',
]

def load_train_datasets():
    selected_cycles = [n for n in range(1, 301) if n % 5 != 0]
    return load_selected_datasets(used_dataset_names, selected_cycles)
```

```
def load_test_datasets():
    selected_cycles = [n for n in range(1, 301) if n % 5 == 0]
    return load_selected_datasets(used_dataset_names, selected_cycles)

def load_all_datasets():
    selected_cycles = [n for n in range(1, 301)]
    return load_selected_datasets(all_dataset_names, selected_cycles)

def load_single_cycle_datasets():
    selected_cycles = [120]
    return load_selected_datasets(used_dataset_names, selected_cycles)

def load_selected_datasets(dataset_names, selected_cycles):
    merged_dataset = load_dataset(dataset_names[0], selected_cycles)

    for dataset_name in dataset_names[1:]:
        dataset = load_dataset(dataset_name, selected_cycles)
        merged_dataset = pd.concat([merged_dataset, dataset], ignore_index=True)

    merged_dataset = merged_dataset.reset_index(drop=True)

    return merged_dataset

def load_dataset(dataset_name, selected_cycles):
    path_prefix = '../'

    dataset_path = path_prefix + 'datasets/data/' + dataset_name

    dataset = pd.read_csv(dataset_path)

    dataset = dataset[dataset['cycle'].isin(selected_cycles)]
    dataset = dataset.reset_index()
    dataset['time'] = pd.to_datetime(dataset['datetime']).apply(lambda x:
x.timestamp())
    dataset['time'] = dataset['time'] - dataset['time'].iloc[0]
    dataset['temperature1'] = dataset['temperature1'] + 273
    dataset['dt'] = dataset['time'].diff().fillna(-1)
    dataset['soc'] = (dataset['chargeCapacity'] - dataset['dischargeCapacity'])

    dataset = dataset[dataset['dt'] > 0]
    dataset = dataset[
        (abs(dataset['current']) > 3.0) | ((abs(dataset['current']) < 1.2) &
(abs(dataset['current']) > 0.01))]

    dataset.dropna(inplace=True)
    dataset = dataset.reset_index(drop=True)

    return dataset
```

5.2.2 Battery Simulation

This is the first part of the total model. Here we model the physical battery using a 2nd Order ECM combined with a SoC approximation. This produces a system of differential and algebraic equations that can be solved in real time using arithmetic

methods. This essentially allows us to calculate given the three inputs (voltage, current and temperature) the intermediate open circuit voltage (V_{oc}) which is internal to the battery and from this the current State of Charge (SoC) for this timestep. Essentially this has been modelled in python using a separate class which implements the system of equations that are needed to be solved along with functions to allow the neural network to be able to update in real time the internal model parameters. The class also holds the current state of the model (i.e. current parameters, state of charge and capacitor charge). Below we include the code for the 2nd order model with the *cubic SoC approximation* as well as the part of the code that differs for the one utilizing the *linear SoC approximation*. The Cubic SoC model also utilized the scipy library to arithmetically solve the 3rd order SoC polynomial. It should also be noted that the SoC approximation has been implemented as a separate function which enables us to change the function separately as needed. In this example also the filtering and scaling functions used in the end have been included.

ecm_2nd_order_model.py (Cubic SoC Approximation with Filtering)

```
import numpy as np
from scipy.optimize import fsolve


class Ecm2ndOrder:
    def __init__(self, VR1, VR2):
        self.update_parameters(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0)

        # State
        self.VR1, self.VR2 = VR1, VR2
        self.time, self.Voc, self.SoC = 0.0, 0.0, 0.0
        self.dt = 1.0

    def update_parameters(self, Ro, R1, R2, C1, C2, a3, a2, a1_soc, a1_temp, a0):
        self.Ro, self.R1, self.R2, self.C1, self.C2 = Ro, R1, R2, C1, C2
        self.a3, self.a2, self.a1_soc, self.a1_temp, self.a0 = a3, a2, a1_soc,
a1_temp, a0

    def update(self, Vout, I, T, dt):
        if (abs(self.VR1) > abs(I*self.R1)) or (abs(self.VR1) < abs(0.7*I*self.R1)):
            self.VR1 = 0.85*I*self.R1

        if (abs(self.VR2) > abs(I*self.R2)) or (abs(self.VR2) < abs(0.7*I*self.R2)):
            self.VR2 = 0.85*I*self.R2

        dVR1dt = (I / self.C1) - (self.VR1 / (self.R1 * self.C1))
        self.VR1 += dVR1dt * dt

        dVR2dt = (I / self.C2) - (self.VR2 / (self.R2 * self.C2))
```

```
        self.VR2 += dVR2dt * dt

        self.Voc = Vout + I * self.Ro + self.VR1 + self.VR2

        self.time += dt
        self.dt = dt
        self.SoC = self.estimate_soc(T)

    def estimate_soc(self, T):
        def poly_func(SoC):
            return self.a3 * (SoC ** 3) + self.a2 * (SoC ** 2) + self.a1_soc * SoC +
self.a1_temp * T + self.a0 - self.Voc

        SoC_solution, info, ier, mesg = fsolve(poly_func, self.SoC, full_output=True)

        SoC_solution = self.SoC + (1/25)*(SoC_solution - self.SoC)
        SoC_solution = 1.0008*(SoC_solution-0.5) + 0.5

        SoC_solution = np.clip(SoC_solution, 0, 1)

        return SoC_solution
```

ecm_2nd_order_model.py (Linear SoC Approximation)

```
def estimate_soc(self, T):
    SoC_solution = (self.Voc - T*self.a1_temp - self.a0) / self.a1_soc

    SoC_solution = np.clip(SoC_solution, 0, 1.0)

    return SoC_solution
```

5.2.3 Reinforcement Learning Algorithm

This is the second part of the model which utilizes reinforcement learning to train the neural network and then combine it with the physical system (in this case modeled by the code presented in the previous section) to change its parameters in real time. This code is split into two different files one for training and one for the testing of the final system. This code utilizes stable-baselines3 which is a python library implementing reinforcement learning algorithms based on PyTorch and can be used as a framework for training and testing neural networks trained using RL. It in turn uses gymnasium library which is an interface that can be used for the representation of RL problems as well as numpy for any arithmetic calculations needed.

Training Code

This code is responsible for the training of the neural network. This is based on an implementation of a Soft Actor Critic algorithm provided in stable-baselines3. The Soft Actor Critic algorithm (SAC) is an off-policy maximum entropy deep reinforcement learning algorithm with a stochastic actor. This algorithm uses an actor that takes actions which in turn a critic evaluates trying to minimize the mean square error for the objective function. In our case this is the minimization of the difference between the real and calculated State of Charge. This part of the code runs independently and internally utilizes the ECM and SoC models presented before. This is by far the most computationally intensive part of the code as it is responsible for the training of the neural network and is required to run for a long time to provide meaningful results (in our each of the different models that are presenting in the next chapter has been training for at least a day but many other models have also been trained to various times while evaluating the process). This code in turn provides a neural network that is saved as a .zip file in an internally recognized format and can then be used to make predictions for the model parameters.

Below we include the main code of the training functionality. As already mentioned this code is responsible for the training of the neural network. As we can see it implements an interface as required by the used libraries to describe the RL problem, the action space, the observation space as well as the reward function. In our case the system model is the ECM/SoC Approximation models that we mentioned above. This model is solved in real time using running in each timestep the code provided in the previous sections which in turn produces an estimation of the current state of charge. Following is a brief description of the most important parts of the code.

__init__ function: As we can see during the initialization of the environment we define an action space (which is the range of the actions the Actor can make an represent the parameters of the system), the observation space (which is the range of the inputs to the NN in our case the input of the system that is the measured voltage, current and temperature of the battery) as well as custom variables defined by us and

needed by the current implementation (such as episode length, capacitor initial voltage etc).

step function: This is the main function where the training is performed. This runs after every system update which in our case is at every timestep of the discretized system. During each step a new action is selected by the Actor which in turn is used to update the ECM/SoC models described in the previous section. These models in turn calculate a new State of Charge estimation by solving the system of equations of the internal model. Based on this the reward function is calculated which is the negative square error of the estimated and real State of Charge. This value is then returned along with the current observation (the measurements of voltage, current and temperature of the battery). These values are in turn processed by the Critic part of the reinforcement learning algorithm which tries to minimise the total estimation error.

reset function: This function is required to reset the environment in the beginning of the training as well as when each episode finished.

The rest of the code initializes the environment described above, sets any initial variables needed and starts the training. In the end a file is produced containing the trained network that can be used for the estimation of the parameters. This model can also be loaded if needed to continue its training.

ecm_2nd_order_train.py

```
import gymnasium as gym
from gymnasium import spaces
import numpy as np
from stable_baselines3 import SAC
from stable_baselines3.common.vec_env import DummyVecEnv
import src.auxiliary.auxiliary as auxiliary
from src.rl.second_order_ecm.soc_voc_cubic.ecm_2nd_order_model import Ecm2ndOrder


class Ecm_env(gym.Env):
    def __init__(self, dataset, Vc1, Vc2):
        super(Ecm_env, self).__init__()

        self.action_space = spaces.Box(low=np.array([0.001, 0.01, 0.05, 500, 5000,
```

```
                                                    0.0, 0.0, 1.0, -0.0005, 0.0]),
                                        high=np.array([0.01, 0.2, 0.5, 5000, 10000,
                                                       0.01, 0.01, 3.0, 0.0005, 10.0]),
                                                       shape=(10,),dtype=np.float32)
        self.observation_space = spaces.Box(low=np.array([1.8, -5.0, 300]),
                                            high=np.array([3.8, 7.0, 315]),
                                            shape=(3,), dtype=np.float32)
        self.dataset = dataset
        self.dataset_length = len(dataset)
        self.Vc1 = Vc1
        self.Vc2 = Vc2
        self.ecm_model = Ecm2ndOrder(self.Vc1, self.Vc2)
        self.episode_length = 1
        self.counter = 0
        self.dataset_parses = 0

    def step(self, action):
        Ro, R1, R2, C1, C2 = action[0], action[1], action[2], action[3], action[4]
        a3, a2, a1_soc, a1_temp, a0 = action[5], action[6], action[7], action[8],
                                      action[9]

        self.ecm_model.update_parameters(Ro, R1, R2, C1, C2, a3, a2,
                                         a1_soc, a1_temp, a0)

        Vout, I = dataset['voltage'].iloc[self.counter],
                  -dataset['current'].iloc[self.counter]

        T = dataset['temperature1'].iloc[self.counter]
        SoC_real = dataset['soc'].iloc[self.counter]
        dt = dataset['dt'].iloc[self.counter]

        self.ecm_model.update(Vout, I, T, dt)
        SoC_estimated = self.ecm_model.SoC

        reward = -np.square(SoC_real - SoC_estimated)

        observation = np.array([Vout, I, T]).astype(np.float32)
        truncated = False
        info = {}

        self.counter = (self.counter + 1) % self.dataset_length

        if self.counter == 0:
            self.dataset_parses += 1
            print("Dataset parse", self.dataset_parses, "finished")

        if (self.counter % self.episode_length) == 0:
            terminated = True
        else:
            terminated = False

        return observation, reward, terminated, truncated, info

    def reset(self, seed=None, options=None):
        super().reset(seed=seed)
        Vout, I = dataset['voltage'].iloc[self.counter],
                  -dataset['current'].iloc[self.counter]
        T = dataset['temperature1'].iloc[self.counter]

        return np.array([Vout, I, T], dtype=np.float32), {}
```

```
dataset = auxiliary.load_train_datasets()

Vc1, Vc2 = -0.8, -1.4
dataset_length = len(dataset)

env = DummyVecEnv([lambda: Ecm_env(dataset, Vc1, Vc2)])
env.reset()

model_name = "model-ep1-cubic"

model = SAC("MlpPolicy", env=env, verbose=0)

model.learn(total_timesteps=dataset_length)

model.save("models/" + model_name)
model.save_replay_buffer("models/" + model_name + "_buffer")

env.close()
```

Evaluation Code/Result Analysis

This part of the source code is responsible for running the tests and evaluating the trained neural network. In essence it loads the model presented before which is the physical system (2nd Order ECM and SoC model) and simulates the system assuming real time operation. This is achieved by loading the dataset files and using the measurements (that were used before for training) as if they were real time inputs measured in a battery. These inputs (i.e. battery voltage, current and temperature) are fed in the physical model as well as the trained neural network which in turn produces the optimal model parameters at each time instant. All the results are stored in separate lists and are then used for plotting as a way to visualize and evaluate the results. This code is mainly used with the test datasets to evaluate the results but can also be used with the training dataset to check the fitting there or any other subset of them for visualization or analysis needs.

ecm_2nd_order_test.py

```
import numpy as np
from stable_baselines3 import SAC
from src.auxiliary import auxiliary
import matplotlib.pyplot as plt
from src.rl.second_order_ecm.soc_voc_cubic.ecm_2nd_order_model import Ecm2ndOrder


Vc1, Vc2 = -0.8, -1.4
ecm_model = Ecm2ndOrder(Vc1, Vc2)

model_name = "model-ep1-cubic"

dataset = auxiliary.load_test_datasets()

dataset_length = len(dataset)
model = SAC.load("models/" + model_name)
SoC_calculated = []
Voc_calculated = []
Ro_list, R1_list, R2_list, C1_list, C2_list, a3_list, a2_list, a1_soc_list,
a1_temp_list, a0_list = [], [], [], [], [], [], [], [], [], []

for i in range(dataset_length):
    Vout, I = dataset['voltage'].iloc[i], -dataset['current'].iloc[i]
    T = dataset['temperature1'].iloc[i]
    action, _states = model.predict(np.array([Vout, I, T]), deterministic=True)

    Ro, R1, R2, C1, C2 = action[0], action[1], action[2], action[3], action[4]
    a3, a2, a1_soc, a1_temp, a0 = action[5], action[6], action[7], action[8], action[9]

    ecm_model.update_parameters(Ro, R1, R2, C1, C2, a3, a2, a1_soc, a1_temp, a0)

    dt = dataset['dt'].iloc[i]
    ecm_model.update(Vout, I, T, dt)

    SoC_calculated.append(ecm_model.SoC[0])
    Voc_calculated.append(ecm_model.Voc)
    Ro_list.append(Ro)
    R1_list.append(R1)
    R2_list.append(R2)
    C1_list.append(C1)
    C2_list.append(C2)
    a3_list.append(a3)
    a2_list.append(a2)
    a1_soc_list.append(a1_soc)
    a1_temp_list.append(a1_temp)
    a0_list.append(a0)


index = dataset.index.values.tolist()
soc_index = index
SoC_real = dataset['soc'].iloc[:].tolist()
Vout_real = dataset['voltage'].iloc[:].tolist()
I_real = dataset['current'].iloc[:].tolist()

mse = sum((a - b) ** 2 for a, b in zip(SoC_calculated, SoC_real)) / len(SoC_calculated)
print("Mean Square Error:", mse)


plt.figure(1)
```

```
plt.plot(index, Ro_list, label='Ro [Ohm]')
plt.plot(index, R1_list, label='R1 [Ohm]')
plt.plot(index, R2_list, label='R2 [Ohm]')
plt.xlabel('datapoints')
plt.title('Parameters from RL, ' + model_name)
plt.legend()
plt.grid()

plt.figure(2)
plt.plot(index, C1_list, label='C1 [F]')
plt.plot(index, C2_list, label='C2 [F]')
plt.xlabel('datapoints')
plt.title('Parameters from RL, ' + model_name)
plt.legend()
plt.grid()

plt.figure(3)
plt.plot(index, a3_list, label='a3')
plt.plot(index, a2_list, label='a2')
plt.plot(index, a1_soc_list, label='a1_soc')
plt.plot(index, a1_temp_list, label='a1_temp')
plt.plot(index, a0_list, label='a0')
plt.xlabel('datapoints')
plt.title('Parameters from RL, ' + model_name)
plt.legend()
plt.grid()

plt.figure(4)
plt.plot(index, Vout_real, label='V')
plt.plot(index, I_real, label='I')
plt.plot(index, Voc_calculated, label='Voc')

plt.xlabel('datapoints')
plt.legend()
plt.grid()

plt.figure(5)
plt.plot(soc_index, SoC_calculated, label='SoC Estimation %')
plt.plot(soc_index, SoC_real, label='SoC Real %')
plt.xlabel('datapoints')
plt.title('State of Charge, ' + model_name)
plt.legend()
plt.grid()

plt.show()
```

5.2.4 Dataset Visualization

Lastly we have a helper function that can plot selections of the datasets and is mainly used for visualization and design purposes (for example inspecting a dataset during the initial design of the system). Here the auxiliary.py presented above is used for

loading the selected datasets as well as that **matplotlib** library for creating the plots. This is a useful functionality especially in the early stages of the training where visual inspection of the datasets is needed. It also helps judge the effect that any preprocessing of the datasets, such as trimming or downsampling, can have.

plot-dataset.py

```
import matplotlib.pyplot as plt
import auxiliary

dataset = auxiliary.load_test_datasets()

index = dataset.index.values.tolist()
current = dataset['current'].tolist()
voltage = dataset['voltage'].tolist()
soc = dataset['soc'].tolist()
temperature = dataset['temperature1'].tolist()

plt.figure(1)
plt.plot(index, current, label='Current [A]')
plt.plot(index, voltage, label='Voltage [V]')
plt.plot(index, soc, label='SoC %')
plt.xlabel('datapoints')
plt.title('Current, Voltage, and State of Charge')
plt.legend()
plt.grid()

plt.figure(2)
plt.plot(index, temperature, label='Temperature [K]')
plt.title('Temperature')
plt.legend()
plt.grid()

plt.show()
```

5.3 Training Methodology

5.3.1 Alternative Models

The training methodology that has been used is given below. In general this is the part that takes the most time as the training depending on the available system and selected datasets can vary vastly. In this work good results have been achieved using the dataset selection described in the previous chapters. A lot of different models have also been evaluated raging from different battery models (Rint and 1st Order ECM

has also been evaluated) as well as different SoC approximations (in this work a linear and cubic approximation with temperature are presented but also the same approximations without temperature as well as other SoC approximations have been evaluated briefly). Specifically in the literature different SoC approximations exist that include higher order polynomials with respect to SoC and/or temperature as well as different functions that consist of logarithmic and exponential terms. Models like these have also been evaluated (trained for at least a few hours each) during this process to various degrees of success resulting in the selection presented in the next chapter that produced the best results. It should be noted that the fitting to any of these models is a result of training time but because of real time constraints not all models could be run for a very long time so for the ones not selected it does not mean that these models cannot necessarily be used successfully for other similar applications or datasets. It should also be noted that the more complex the model gets the training time can increase exponentially as new parameters are introduced for which the Soft Actor Critic (SAC) algorithm of the reinforcement learning need to try combinations as well as more complex models could introduce more local minima describing suboptimal solutions that the algorithm can converge to, so a good trade off between acceptable accuracy and model complexity needs to be achieved.

5.3.2 Training Process

During training and after a model has been selected two main things firstly need to be defined which are the range of the action and observation space.

Action Space

The action space is the range of values that can be proposed by the Actor of the RL algorithm (the physical system's parameters in our case). This needs to be selected using a reasonable range of values because if care is not taken we could exclude the optimal solutions altogether as we constraint the algorithm not to search in those ranges. On the other hand if the range is too big we can increase the training time needlessly as the RL algorithm will evaluate parameters that are not realistic for our

system thus wasting processing cycles. In our case a range of parameters have been used as have been proposed in the literature (for example [9], [10]). Typical range of parameters include 0.001Ω to 0.5Ω for the resistors and 500F to 10000F for capacitors. The parameters for the SoC polynomial range from a little below zero to around 10.

Observation Space

The observation space is in turn the range of values the inputs to the SAC algorithm are going to take (in essence the value domain). These ranges are defined by the datasets available and are the ranges of the inputs that have been measured during the experiments. So to set the observation space we look at those ranges in the dataset (and we also make the range slightly bigger to have some extra room). In our case the voltage is between 1.8V - 3.8V, current is between -5.0A and 7.0A and temperature between 300°K and 315°K

After the initial selection of the parameters the training is conducted. As already mentioned a lot of different models have been evaluated. A candidate model is initially run for a few hours to investigate if any fit is achieved. If it is, the model is reloaded and training is continued. The models that provided the best fits have been training for 20+ hours at least although it needs to be mentioned that most of them did not achieve much better results if trained for longer. The output either stayed the same or improved slightly (while in some cases there was a deterioration in some areas and improvement in others).

Training Flow

During the training and after the initialization mentioned above the following process is conducted:

1. The Actor of the RL training algorithm produces a set of actions to be used by the system. These actions in our case represent all the parameters of the physical system.

2. The ECM model with the SoC Approximation described above are updated with the proposed parameters from the RL

3. A new State of Charge estimation is calculated

4. Using this estimation with the real SoC from the dataset the reward function is calculated

5. The result of the reward function, the current observation (V, I, T of the current time step as measured in the dataset) are passed to the Critic of the RL algorithm which evaluates the result and updates the neural network.

6. The process continues for all the available data.

In the end we have a trained neural network representing a policy that its goal is to minimize the SoC error calculated from our model given the input conditions. The training is conducted in the train datasets described (80% of all available measurements).

5.4 Evaluation Methodology

When the training completes the neural network produced can be used to produce optimal actions (meaning parameters of the physical model) given inputs V, I and T. This can happen in real time as new measurements are provided. In our case we use the datasets in a step process described below.

Evaluation Flow

1. A trained neural network describing an optimal parameter policy is loaded

2. We prepare the test datasets which in our case in which each timestep is treated sequentially as if it were real time measurements.

3. For each measurement the real measurements for V, I and T are fed to the neural network which in turn provided the optimal action for the current stage (i.e. the parameters of the system)

4. These parameters are updated to the physical model consisting of the 2nd Order ECM and the chosen SoC Approximation Model.

5. The system of differential and algebraic equations are solved for the current timestep using the parameters provided by the RL providing a SoC estimation.

6. We return to number 3 and we continue the same process for all the timesteps.

This process produces a State of Charge curve that is the estimation of the actual SoC measured in the experiment. These values are then used to calculate the error metrics of the estimation as well as plot the different simulation curves for visualization and further analysis.

In general the datasets are constructed in a way so the battery is simulated like it is running on multiple charging/discharging cycles one after the other which have different load profiles. This happens in order to make the training and testing data seem like they come from a real scenario where the battery will be working in different conditions in its successive cycles.

6 Results

6.1 Experiments

Below we will present the simulations for different models that have been done and also a comparison between them. Two different main models have been validated. As has been presented before the base battery model for all the use cases is a 2nd Order ECM which provides a good trade off between model complexity and accuracy capturing the main battery dynamics such as battery hysteresis and polarization with an adequate approximation. The difference between the use cases is the approximation used for the modeling of the State of Charge of the battery which is combined with the 2nd Order ECM to calculate the charge and discharge of a Li-ion battery. For the State of Charge two approximations have been considered. The first is a linear approximation of the V_{oc} and SoC and the second is a cubic approximation of the V_{oc} and SoC that also includes a linear term to take into account the battery's temperature.

6.1.1 2nd Order ECM - Linear SoC Approximation

This model uses a 2nd Order ECM and a linear approximation with respect to SoC and temperature. Below we can see the output of the model for 7 consecutive charge and discharge cycles of a Li-ion battery. The model can in general achieve a good approximation of the state of charge with the exception of the top part of the cycle where it has a flatter output.

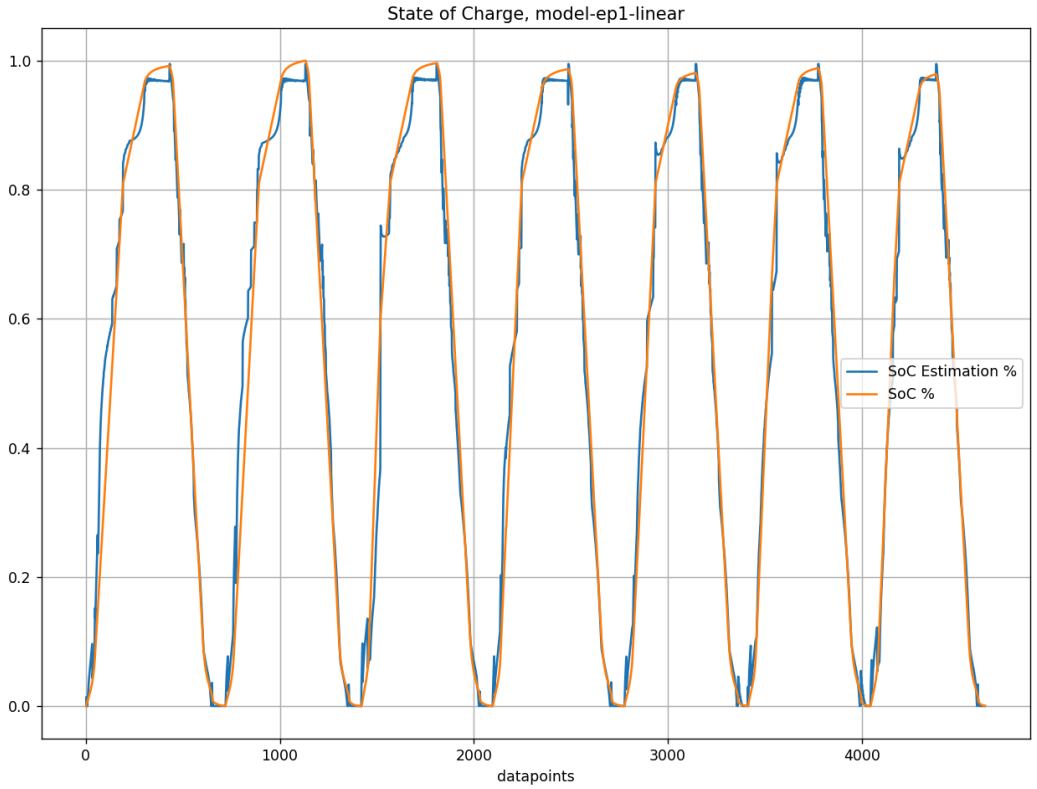


Figure 18: Model 1 - State of Charge Results

In the graph below we have the output of the neural network. This is a selection from all the parameters that are provided by the model (for example also the resistances and capacitances of the model are provided). We observe that the model actively changes the parameters capturing the non linear behaviour of the system. In the third graph we also provide the measured parameters V and I that are used as an input to the circuit and the open circuit voltage V_{oc} which is an intermediate variable in the system and the output of the 2nd order ECM.

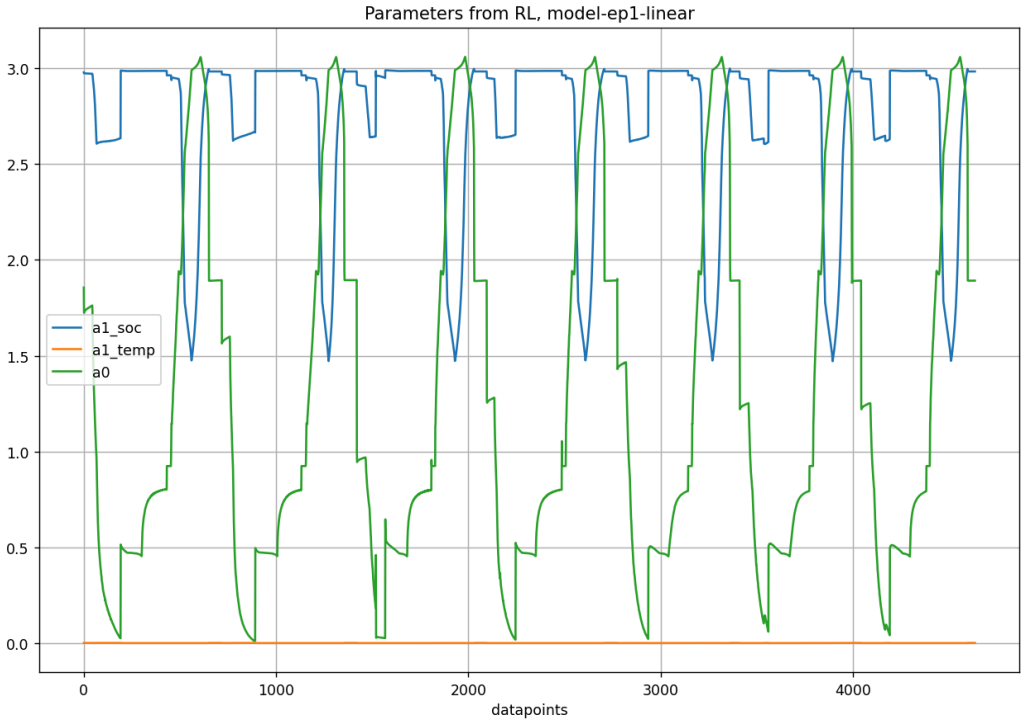


Figure 19: Model 1 - Neural Network Parameters

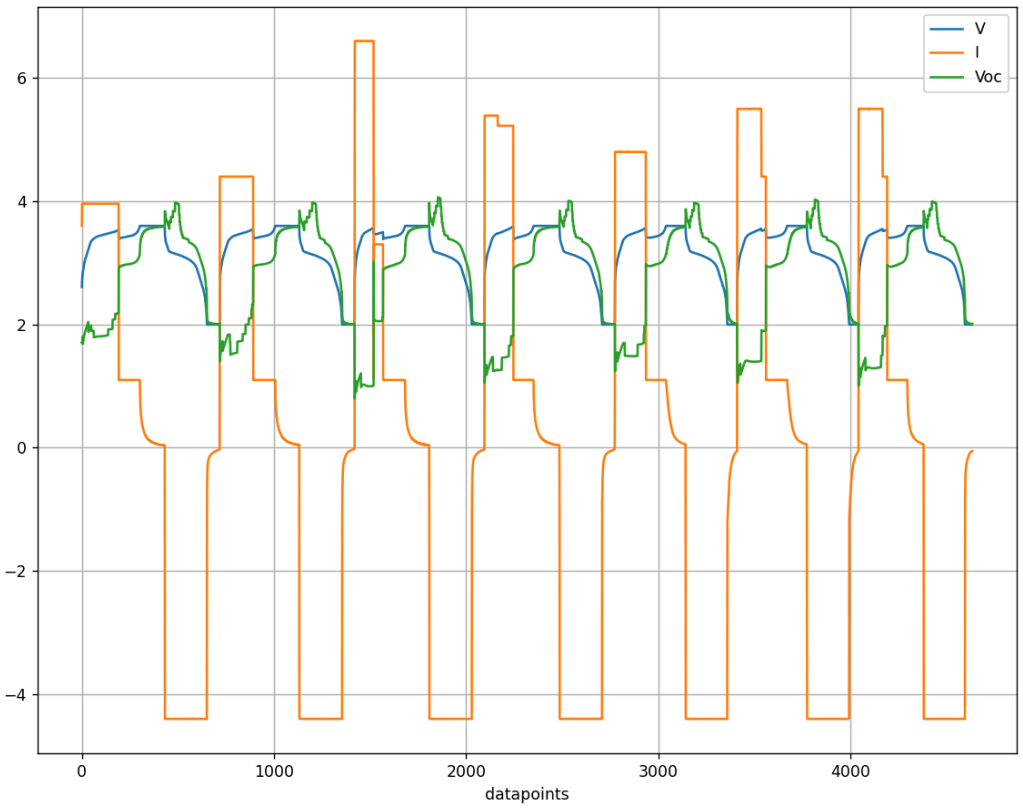


Figure 20: Model 1 - ECM Inputs and Outputs

The error metrics of the output are given below. We can see that the errors are low in spite of some regions (the top of the curve in this case) that the error increased. Also a high R Squared score close to one shows that the curve has overall a good fit.

Table 2: 2nd Order ECM + Linear SoC Results

Error Metrics 2nd Order ECM + Linear SoC	
RMSE	5.66%
MAE	3.66%
R-Squared	0.976

6.1.2 2nd Order ECM - Cubic SoC Approximation - Model 1

This is the second model that has been evaluated. This is also using a 2nd Order ECM for the battery but it also uses a cubic approximation for the SoC of the battery as well as a linear term for the temperature. This model also has a good overall approximation of the SoC although we can observe some flatter regions on the top of each charge cycle. The error of those flat regions is reduced with respect to the linear model presented above.

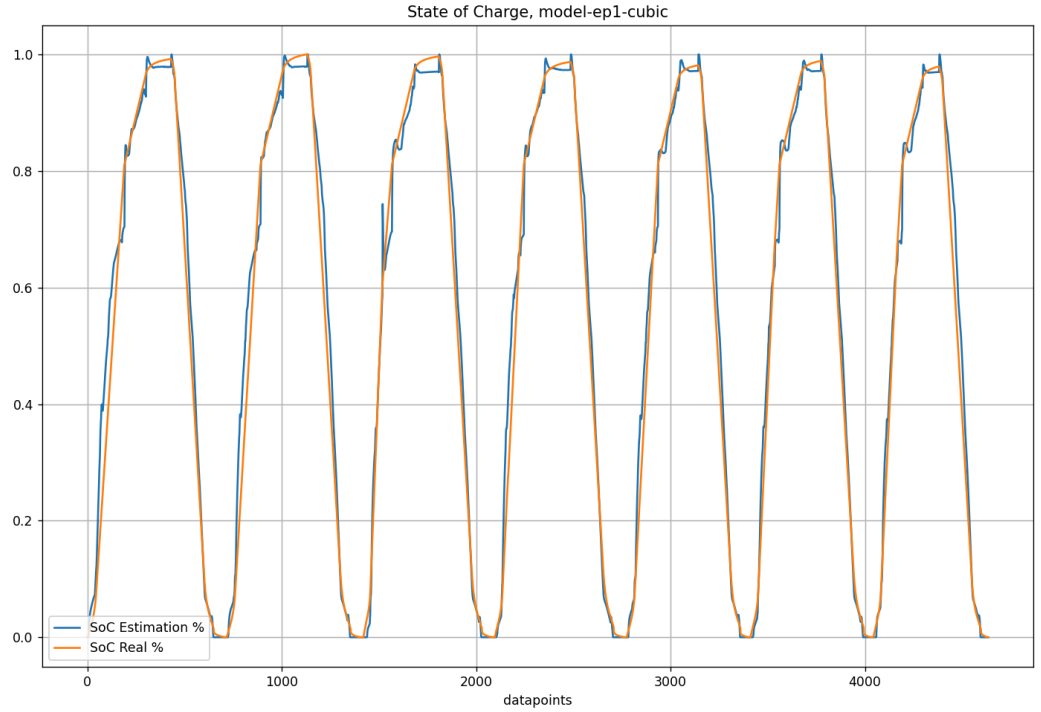


Figure 21: Model 2 - State of Charge Results

In the two graphs below we present a selection of the parameters of the output of the neural network. We can see here that since the approximation is cubic we have two more parameters that have been introduced to represent the coefficients for the square and cubic terms of the equation. We also present the graph of the physical measurements and ECM output (V_{oc}).

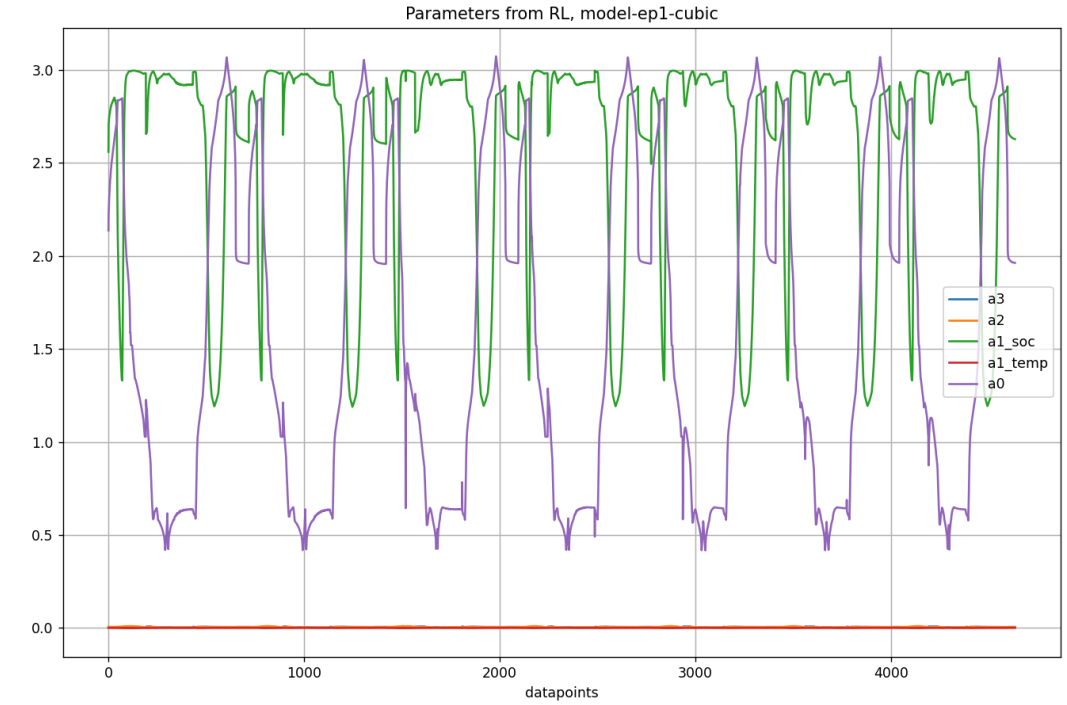


Figure 22: Model 2 - Neural Network Parameters

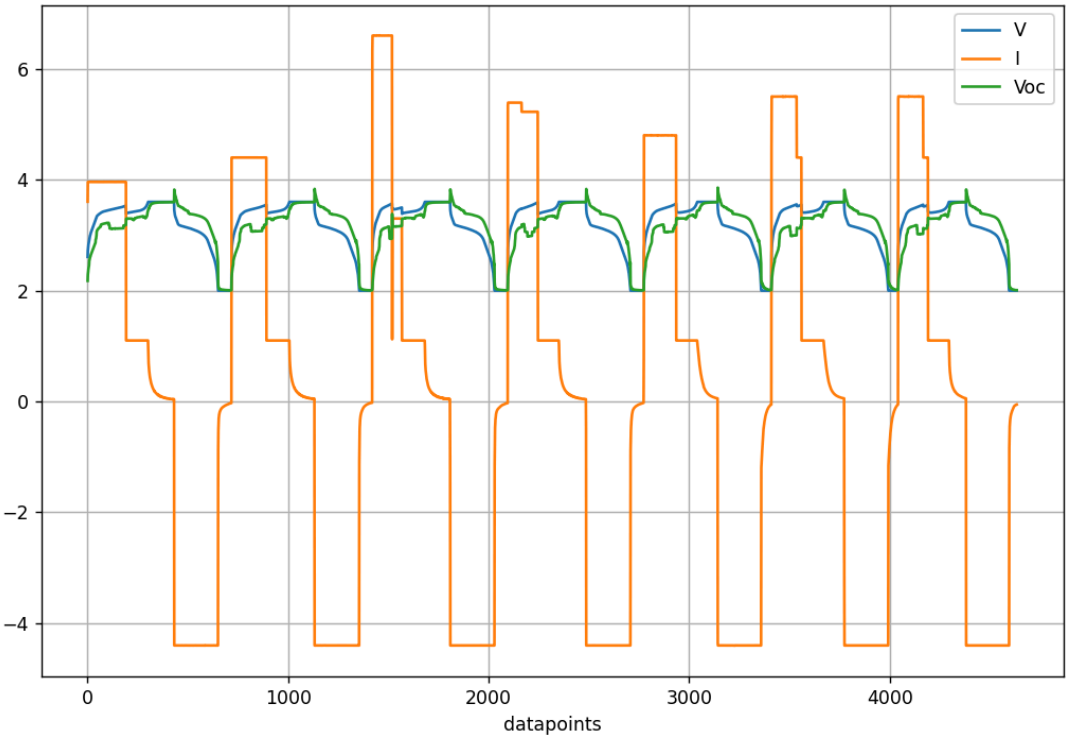


Figure 23: Model 2 - ECM Inputs and Outputs

We also see that in this model which uses the higher order approximation the overall error is reduced for all the error metrics with respect to the linear model.

Table 3: 2nd Order ECM + Cubic SoC - Model 1 - Results

Error Metrics 2nd Order ECM + Cubic SoC 1	
RMSE	4.95%
MAE	3.17%
R-Squared	0.982

6.1.3 2nd Order ECM - Cubic SoC Approximation - Model 2

This model is identical with the model of the previous section. It uses a 2nd order ECM, a cubic approximation for SoC as well as a linear term for modeling the battery temperature. The difference between the two models is that they have been trained independently for similar times with the same methodology and on the same datasets to investigate the results. As we can see, the output of the whole model also achieves a reasonable approximation of the real SoC but is a bit different from the one of the previous one mostly in the top regions of the curve where the calculated output does not have as flat a response as before. As a trade off we can observe that there are some deviations in lower regions of the curve. In general with these comparisons we want to present that using the same methodology can produce slightly different models as the neural network that is produced by the learning process can converge to different values depending on the nature of the functions and local minima that it can find. In any case those models both approximate the SoC with similar accuracy so there are not big deviations in the total errors.

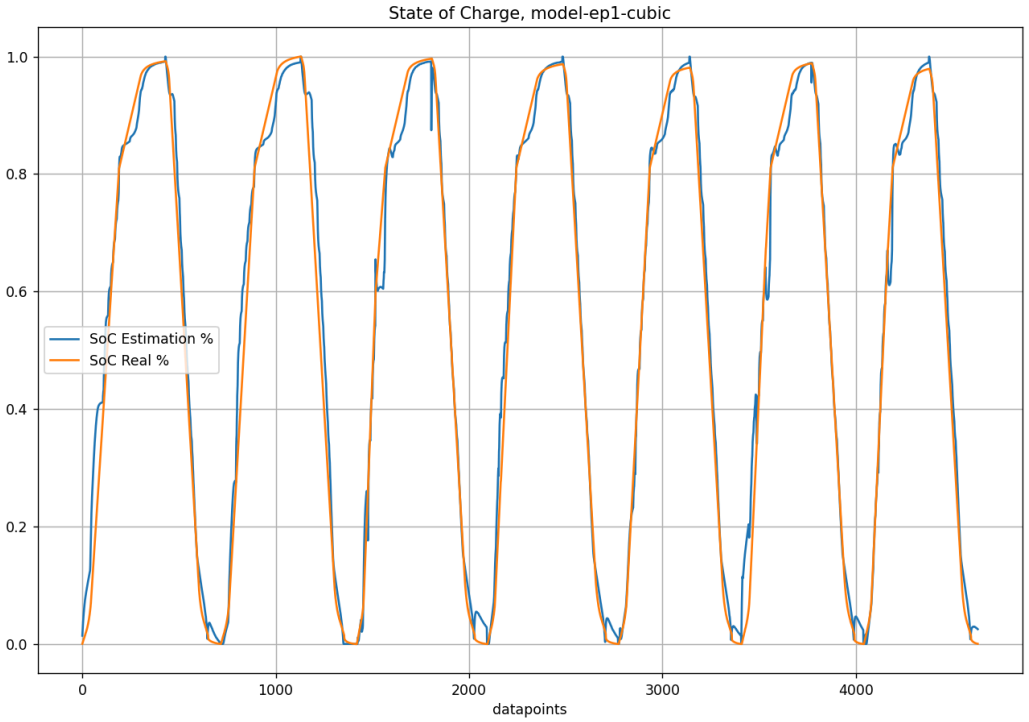


Figure 24: Model 3 - State of Charge Results

Like before we also provide the two graphs, one for the parameters as are outputted from the NN and one for the output of the ECM model along with the real parameters V and I as measured in the real battery. It is worth noting here that some visible differences occur between these models that lead to similar approximations of the SoC though. This makes evident the different minima that this approach can converge to, something that is expected to be amplified the more complex a model gets.

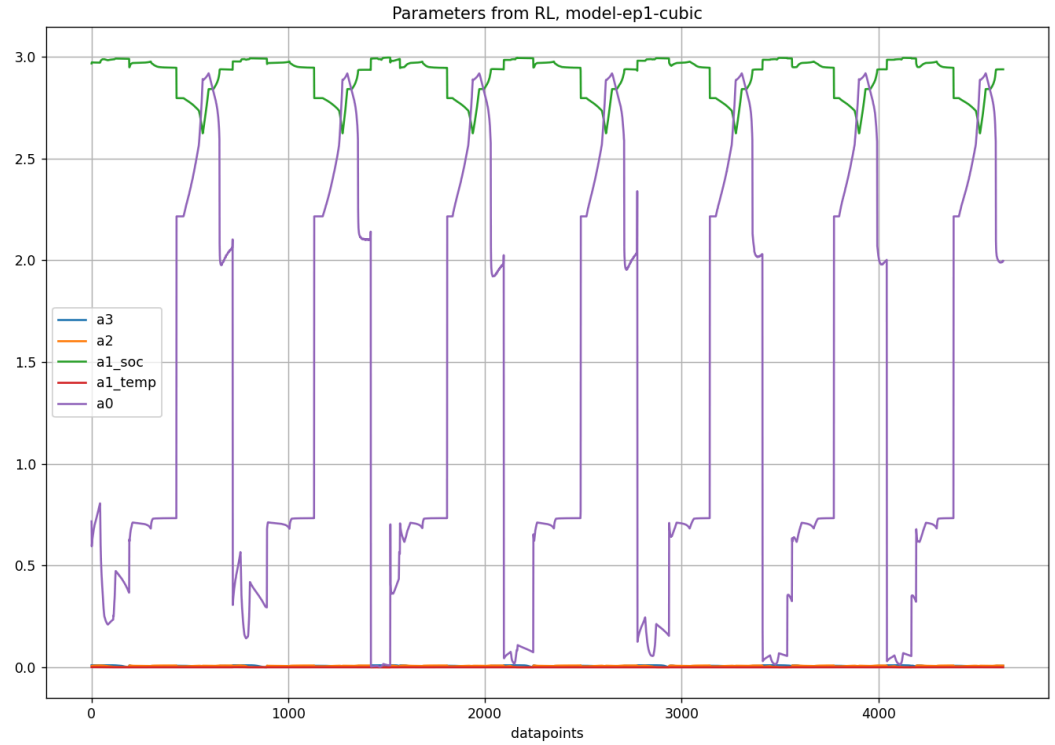


Figure 25: Model 3 - Neural Network Parameters

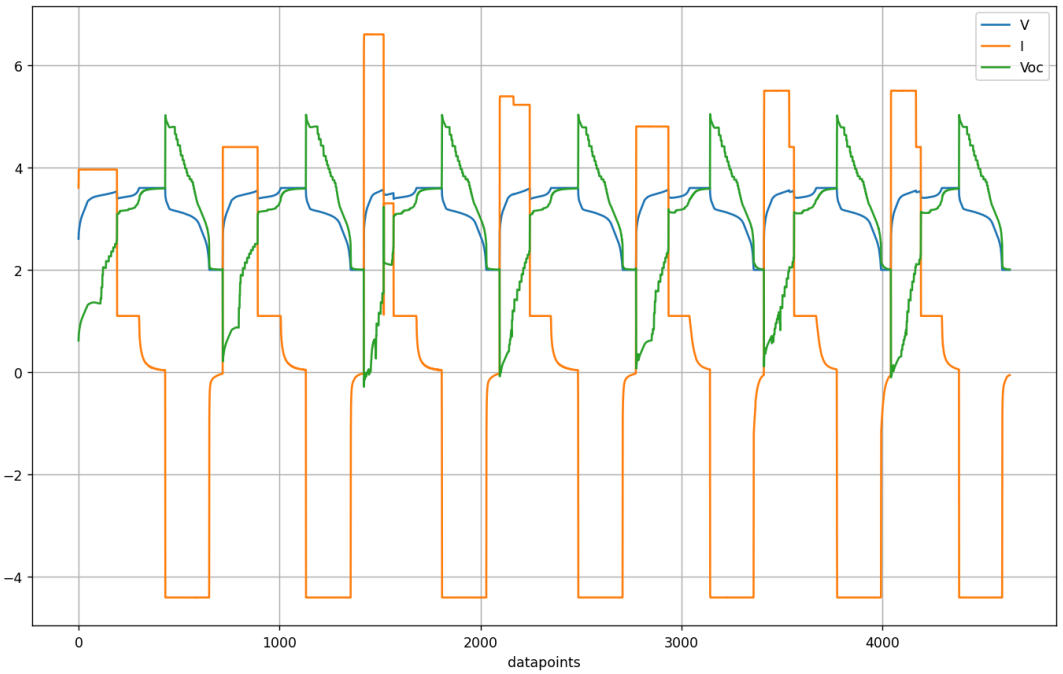


Figure 26: Model 3 - ECM Inputs and Outputs

Lastly the mean square error is provided below. This error is comparable to the first cubic model (with MAE being a bit higher). We can say that in practice this will not be a meaningful difference and the decision on which model should be used should be taken qualitatively based mostly on the general response of the computed SoC.

Table 4: 2nd Order ECM + Cubic SoC - Model 2 - Results

Error Metrics 2nd Order ECM + Cubic SoC 2	
RMSE	4.94%
MAE	3.41%
R-Squared	0.982

6.1.4 2nd Order ECM - Cubic SoC Approximation - Filter

All the above models present an acceptable approximation of the state of charge for different applications and alone or maybe with some minor tweaking can be used with acceptable accuracy. One step further can be to add one more stage to the whole model which is the filtering of the output. This does not require any changes to the ECM model or the NN which can be used as is. The idea here is that the 2nd Order ECM model, although a system of linear differential equations, can present some discontinuities at times when coupled with the NN. The reason is that since the NN changes the parameters of the model in real time, big changes will affect the internal behavior of the model instantly producing jumps and overshooting. This is slightly counteracted by the fact that the ECM model includes two capacitors that could slightly filter these changes as a side effect but they cannot do this for the SoC model that is also affected by the NN and also this is not their main function as they only need to model the internal behavior of the battery correctly.

Thus a simple filter has been added to the output of the model to investigate if the behavior will improve. Any filter could be used with different cut off frequencies but here a simple moving average has been used. This filter has three effects on the model output. Firstly it smoothens the output depending on the cut off frequency (or number of samples in the case of the moving average) that has been selected. As a byproduct of the filtering, which is affected by our previous selection, the output amplitude is reduced and a time lag is also introduced. The two effects are not desirable so special steps are taken so they can be compensated.

In this case we present the final model that produced the best results. This is a model identical to the 2nd Order ECMs with Cubic SoC approximation (discussed in the previous two sections) but also the proposed filtering has been added in the output. Both cubic models can produce similar results so any one of them can be selected (so the neural network that has been used is from one of the models that has been presented above and not a newly trained one).

In this model the proposed filtering has been applied to the output without affecting or changing the base model calculation presented above in any way. The filtering and compensation parameters have been selected once (here through trial and error) though more analytical methods could be employed if needed. The parameters of the filter do not change in any way during the simulations to avoid specifically tuning the model for each dataset. They are selected one time and used as is and stay fixed for every cycle and for all the different datasets.

Table 5: Model 4 - Filter Parameters

Filter Parameters		
Moving Average Samples	***Scale Factor***	***Delay Compensation***
25 samples	1.0008	22 samples

In the figure below we have the output of the model. We can see that the output is much smoother and it produces a better overall fit. Some small peaks exist at the highest point that is a remnant of the overshooting of the normal output but overall there are much smaller spikes in the graph. This could be expected as the state of charge of a battery is expected to change slowly in most normal scenarios. The two graphs for the NN estimated parameters as well as the ECM inputs and outputs are also given.

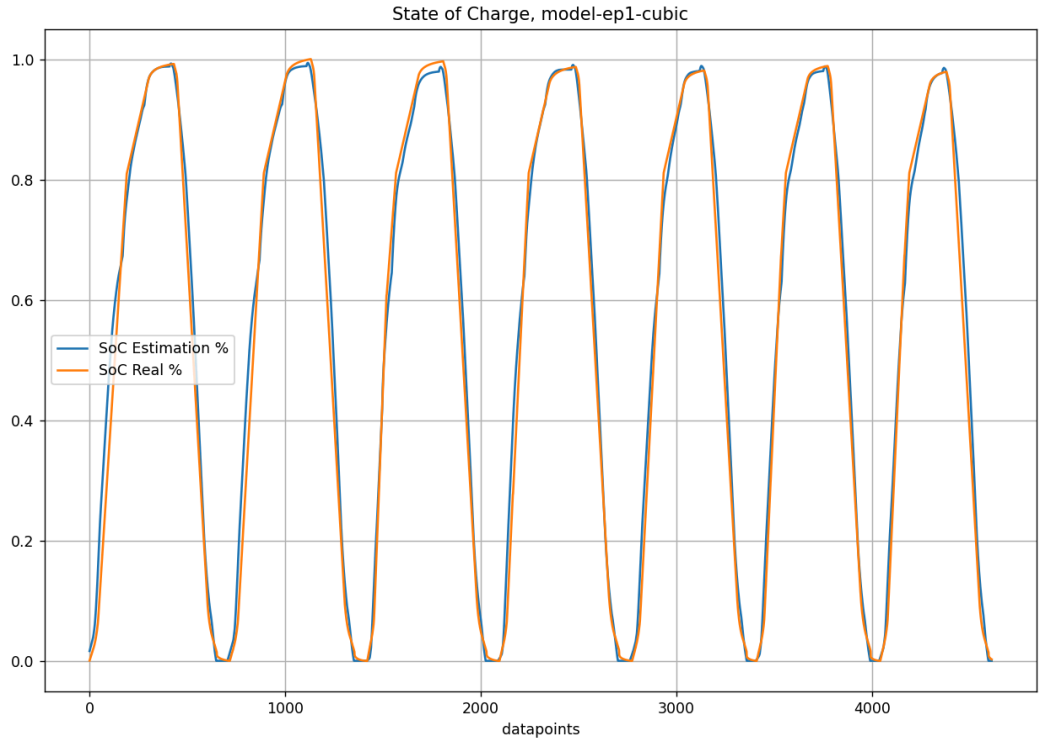


Figure 27: Model 4 - State of Charge Results

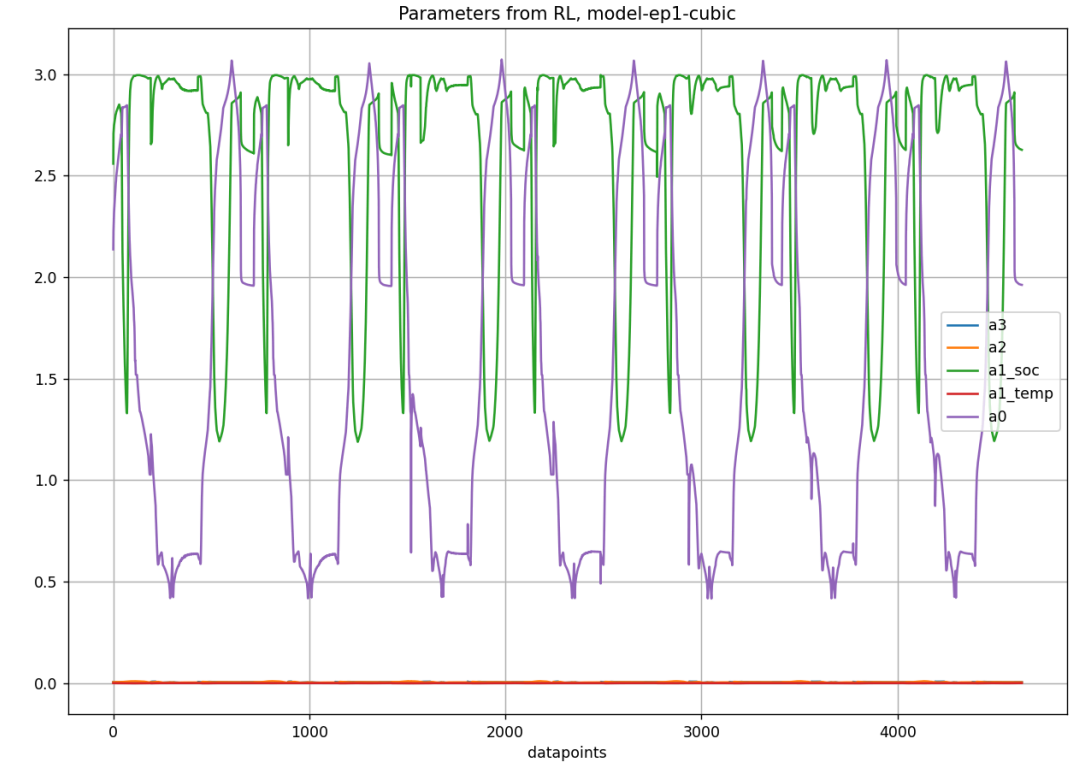


Figure 28: Model 4 - Neural Network Parameters

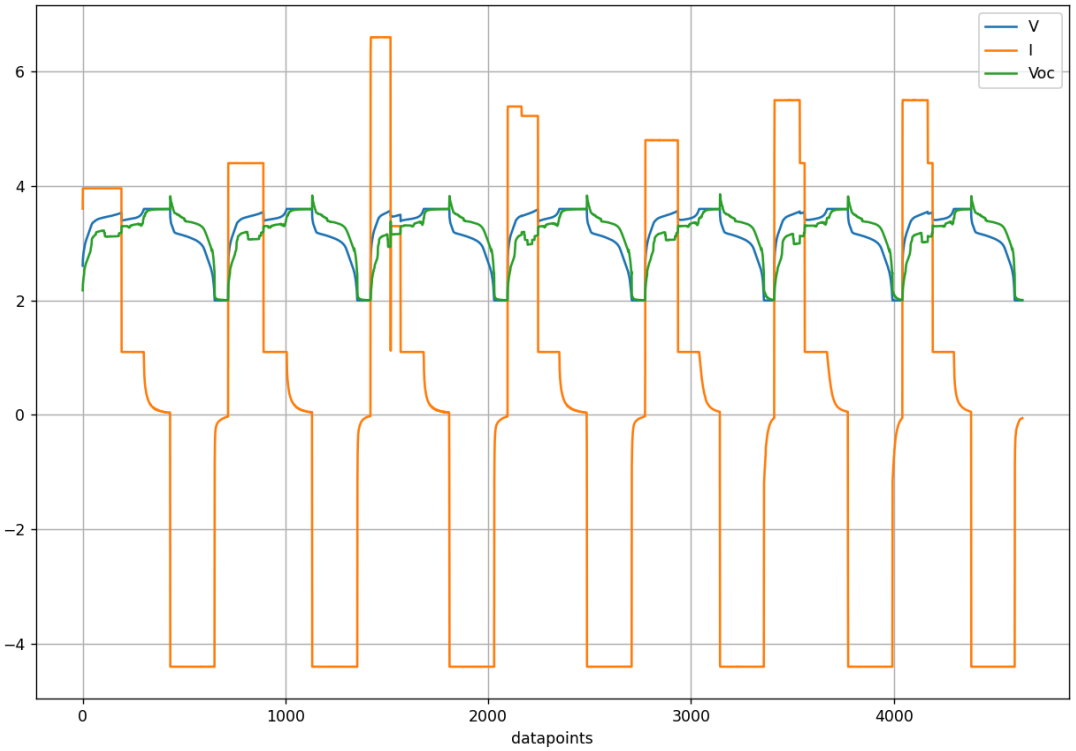


Figure 29: Model 4 - ECM Inputs and Outputs

From the error metrics of this model we can also find that this is a much better fit than the previous ones and the filtering does provide a noticeable improvement if applied on the output of a Cubic Model. As we mentioned above this filtering is done once for all datasets and no tuning for a specific cycle is done so in essence these parameters could be selected once and used as is for all possible scenarios. We can see here that both RMSE and MAE errors have been reduced an we have a slightly increased R Squared metric which show an improvement to the curve fitting.

Table 6: 2nd Order ECM + Cubic SoC + Filter - Results

Error Metrics 2nd Order ECM + Cubic SoC + Filter	
RMSE	4.49%
MAE	2.94%
R-Squared	0.985

6.2 Evaluation of Results

In general from the results presented above the accuracy of estimation has low error even for the simpler models. Between the models we can observe some improvements, which is desirable, but depending on the application these improvements may not necessarily be needed and a simpler model could provide adequate accuracy. Indeed even the worst performing model which is linear does achieve low error metrics by itself. The biggest inaccuracies of these models lie mostly in specific regions (mostly on the top and bottom regions of the state of charge curve).

Lastly we present the error metrics of all the models together. As we described above all models use the same 2nd Order ECM as the underlying battery model and have been trained on the same datasets for similar time. All models also include the temperature as a linear term in the SoC estimation. The difference lies in the order of

the SoC model with respect to the SoC variable itself. We can see that the linear model with respect to SoC achieves the lower accuracy (although still good in absolute terms). Then we have the two Cubic models that have been separately trained which provide an increase in the overall accuracy with respect to the linear model and similar performance with respect to each other. The final model is the Cubic model that has a filter applied to its output as described above with no alterations to the internal behavior of the model either the ECM or the NN which also provides an improved performance with respect to all the previous models.

Table 7: Comparison of Results

Error Metrics				
	2nd Order ECM + Linear SoC	2nd Order ECM + Cubic SoC 1	2nd Order ECM + Cubic SoC 2	2nd Order ECM + Cubic SoC + Filter
RMSE	5.66%	4.95%	4.94%	4.49%
MAE	3.66%	3.17%	3.41%	2.94%
R-Squared	0.976	0.982	0.982	0.985

6.3 Effects of Datasets

The datasets also play an important role in the final accuracy. As already mentioned each dataset (containing multiple cycles, i.e. multiple battery charges and discharges) is done under different conditions. This means that different loads are applied to the battery and which results in different current and voltage patterns during the charging and discharging. Also during an experiment where the battery is charged and discharged multiple times (something that can take multiple hours) also affects the temperature and thus the behavior of the battery. Lastly there is always the aging factor for a battery and the same battery can perform differently after a large number of charges and discharges even under the same load profiles. This makes the training of the neural network take into account different conditions and the resulting model, while it has a low error overall, it is expected to perform better or worse on specific datasets.

For example below we have three consecutive charging-discharging cycles of the battery using different load profiles from different datasets. As we can observe the fit differs between each dataset with some dataset being approximated better than others.

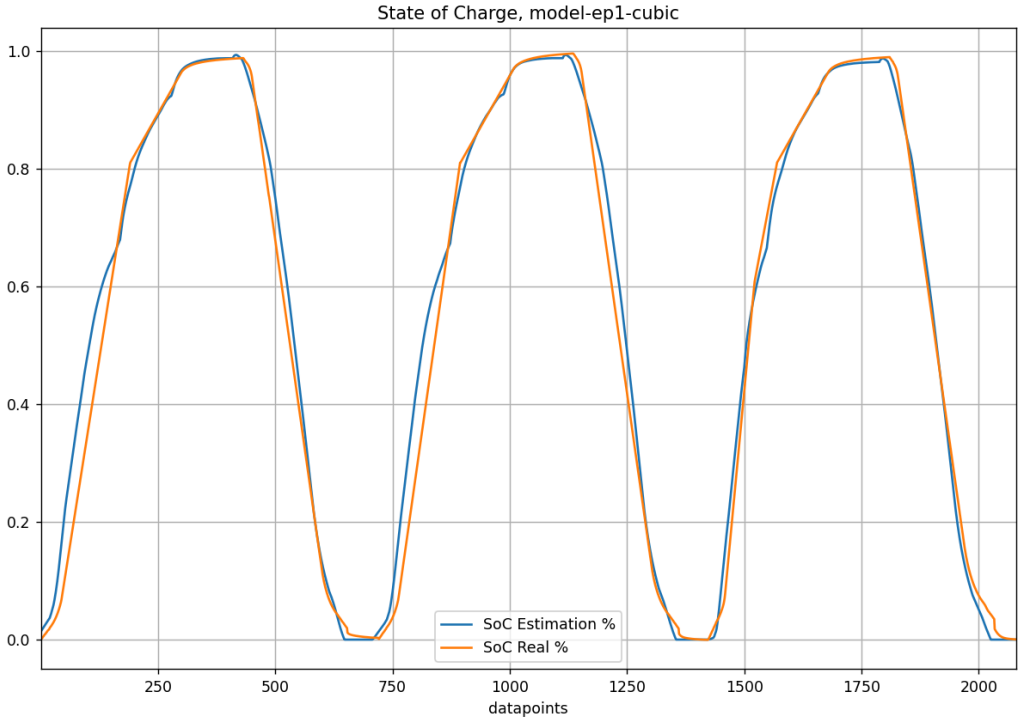


Figure 30: Model 4 - Results (3 Cycles)

Essentially the RL algorithm will try to find the optimal policy that fits a better curve on average to all the different datasets. This is also coupled with the fact that the base dynamics are modeled by the ECM which should capture the average behavior of a cycle regardless of the conditions or at least it should not diverge that much. That in itself provides a guarantee that even if different conditions are met between the datasets or during the deployment, the model should not diverge that much from an acceptable solution. If this is not the case the physics-based model should be revisited and improvements there should be considered.

7 Conclusion

7.1 Summary

In the above work we have presented the implementation of a hybrid approach that uses classical modeling along with reinforcement learning in order to model a Li-ion battery and estimate its State of Charge which is an internal quantity and cannot be directly measured. While several approaches have been proposed, most include either complex electrochemical models that are non trivial to model and solve, approaches based on physics-based models (Kalman Filters) or completely data driven methods based on machine learning. This approach aims to provide a framework where both a physics-based model can be used to capture the main dynamic behavior of the system as well as a neural network that acts as an optimal estimator for the system parameters and captures nonlinear dynamics that may exist while also providing a model that could run in real time during a battery's operation.

7.2 Key Findings and Evaluations

This approach can provide good accuracy for a real scenario of operation. The models have provided good approximations and low estimation errors for the different datasets that have been used. Moreover the data driven approach of the RL allows us to capture possible nonlinear behavior that is harder to be captured by more classical approaches while it also provides a model lightweight enough to be feasible to run in real time.

The main advantages of the method are summarized below

1. It utilizes a data driven approach while it also uses a physics-based model. This can be advantageous in applications where physical modeling has previously been done and a different approach could be desirable without abandoning any work that already exists.

2. The physical model already captures part of the dynamics which makes it easier for the RL algorithm to converge to a solution and improves accuracy. The NN trained by the RL can drive the model to an optimal state by introducing non linear behavior by changing its parameters while leaving the model to capture faster dynamics that could be harder to capture with a data driven approach..

3. Since the physics-based model can be thought to act as a constraint to the RL it could be argued that faster training or training that requires less data could be achieved.

4. It is fast enough to be used in a real time scenario.

5. Having a physics-based model could improve the generalization capabilities of the model to scenarios outside the provided datasets as a big part of the system dynamics is already captured.

7.3 Limitations

This methodology, while it provides good accuracy in the current use case, should be further investigated in more scenarios. Some disadvantages and limitations of the method are the following.

1. It utilizes a hybrid approach. This can be also a disadvantage as well as an advantage since competence for both classical modeling and data driven methods need to be present within a team which can prove a limiting factor.

2. While it is fast enough to run in real time, it is not necessarily faster than simpler methods that could be adequate in many cases especially in applications where not

that much processing power or memory is available. This can be a limiting factor as both a physical model as well as a neural network need to run concurrently.

3. Good quality datasets are needed. If these datasets do not already exist separate meticulous experiments should be done which requires time and the necessary experimental setup.

4. On the other hand if good quality datasets exist the time that is required to investigate physical models to be used with this approach may not be worthwhile if concrete advantages in the accuracy or generalization of a model are not provided.

We can see that a lot of the advantages of the method can also prove disadvantages so the method should be evaluated on a case by case basis taking into account the competences available within a team as well as the problem at hand.

7.4 Future Work

There are a number of things that could be investigated further mainly related to the advantages mentioned above.

Generalization Capabilities

It is worthwhile to investigate how good the method generalizes to other datasets of similar systems. Since part of the dynamics are captured by the physical model a study where a model is trained in a specific dataset but is also tested at experiments done at completely different battery conditions can be conducted to test how the model behaves. The results will be useful to be compared to the ones of a purely data driver approach in the same use case.

Improved Training

It could be argued that since the physical model behaves as a constraint to the RL network, reduced training time or a reduction in training data needed could be possible while achieving good results. A comparison of different data driven methods

along with this method could be done to investigate if this can be proven true. This can be especially useful in more complex systems.

Different Use Cases

This method could be applied to different use cases i.e. either modeling batteries using different models for the battery and state of charge estimation or modeling completely different systems and evaluating the results.

8 *References*

[1] Fadlaoui Elmahdi, Lagrat Ismail, Masaif Noureddine, Fitting the OCV-SOC relationship of a battery lithium-ion using genetic algorithm method, The International Conference on Innovation, Modern Applied Science & Environmental Studies

[2] Yong Tian, Dong Li, Jindong Tian, Bizhong Xia, State of charge estimation of lithium-ion batteries using an optimal adaptive gain nonlinear observer, Electrochimica Acta, Volume 225, 2017, Pages 225-234, ISSN 0013-4686

[3] Yujie Wang, Jiaqiang Tian, Zhendong Sun, Li Wang, Ruilong Xu, Mince Li, Zonghai Chen, A comprehensive review of battery modeling and state estimation approaches for advanced battery management systems, Renewable and Sustainable Energy Reviews

[4] Xiaosong Hu, Shengbo Li, Huei Peng, A comparative study of equivalent circuit models for Li-ion batteries, Journal of Power Sources, Volume 198, 2012, Pages 359-367, ISSN 0378-7753

[5] Yuan Tian, Manuel Arias Chao, Chetan Kulkarni, Kai Goebel, Olga Fink, Real-time model calibration with deep reinforcement learning, Mechanical Systems and Signal Processing, Volume 165, 2022, 108284, ISSN 0888-3270

[6] Unagar, A.; Tian, Y.; Chao, M.A.; Fink, O. Learning to Calibrate Battery Models in Real-Time with Deep Reinforcement Learning. Energies 2021, 14, 1361

[7] Yu, QQ., Xiong, R., Wang, LY. et al. A Comparative Study on Open Circuit Voltage Models for Lithium-ion Batteries. Chin. J. Mech. Eng. 31, 65 (2018)

[8] Severson et al. Data-driven prediction of battery cycle life before capacity degradation. Nature Energy volume 4, pages 383–391 (2019)

[9] Pang, H.; Zhang, F. Experimental Data-Driven Parameter Identification and State of Charge Estimation for a Li-Ion Battery Equivalent Circuit Model. Energies 2018, 11, 1033

[10] Ruba, Mircea & Nemes, Raul-Octavian & Ciornei, Sorina & Martis, Claudia. (2020). Parameter Identification, Modeling and Testing of Li-Ion Batteries Used in Electric Vehicles