POLITECNICO DI TORINO

Master's Degree in Data Science and Engineering

Master's Degree Thesis

A novel procedure for real-time SOH estimation of EV battery packs based on Time Series Extrinsic Regression

Supervisors

Prof. Edoardo PATTI

Prof. Alessandro ALIBERTI

Dott. Raimondo GALLO

Dott. Paolo TOSCO

Candidate

Tommaso MONOPOLI

December 2022

Abstract

With the increasing awareness on environmental issues, research and development revolving around automotive industry – one of the most important yet pollutant industries world-wide – has gained momentum. In recent years, electric vehicles have been widely accepted as a clean and reliable alternative to fossil fuel vehicles, both in private and public transportation sectors, and are expected to quickly take over the market in the upcoming years.

Lithium-ion batteries have emerged as the main enabling technology in the development of EVs, mainly due to their high energy density and long lifespan. One of the key challenges posed by the spread of EV LIB packs is the real-time estimation of their state of health (SOH), commonly regarded as the main indicator of EV aging. However, SOH estimation is still a challenging task due to the electro-chemical complexity of LIBs and their non-linear charge and discharge dynamics.

In this thesis, the current solutions employed for SOH estimation of EV battery packs are discussed, and a novel real-time and computationally-inexpensive machine learning procedure for on-board SOH estimation is introduced, requiring just a narrow time window of voltage, current and state of charge measurements taken while driving the vehicle. To make up for the lack of large-scale publicly available EV monitoring data, a synthetic dataset has been generated by simulating multiple driving sessions of a Simulink-based EV model. The performances of our procedure are investigated and compared with those of other state-of-the-art methods in the field of time series extrinsic regression. Performances are evaluated both on a test set of synthetic data and on a real driving session monitoring dataset acquired from a private EV fleet management company.

Despite achieving promising results on synthetically-generated data, the procedure designed is still not able to perform well on real EVs, suggesting that further research is required. However, we claim that this procedure has much room for improvement with regards to its generalization capability, and we propose a path for future works to be developed.

Ringraziamenti

A tutti i colleghi delle Officine Edison Torino, che hanno creduto in questo lavoro e che lo hanno sostenuto con la loro professionalità e le loro conoscenze, in particolare a Paolo, Rémi, Marco, Alessandro. Alle persone del Politecnico di Torino che mi hanno dato la possibilità di lavorarci: Edoardo, Alessandro, Raimondo.

A Claudia, collega e amica, con cui alle 13 andrò da *Fiore* per la pausa pranzo.

A Ruggiero, accanto al quale ho camminato in questo percorso accademico, che lo ha reso più leggero per me e che non poche volte mi ha indicato la direzione da seguire quando mi smarrivo.

A Mattia e Pixel, per la vostra ospitalità ed i vostri pranzi stellati.

Ai coristi e alle coriste del *Coro PoliEtnico*, con cui mi auguro di condividere ancora a lungo risate, musica, trasferte e abbuffate. In particolare a David, per la sua estrema generosità e la sua spontanea simpatia, e a Roberto, a cui mi sono fin da subito sentito affine – non solo per i capelli ricci.

Alle amicizie storiche, quelle non segnate dal tempo e dalla distanza geografica. Alla *Corteccia* – Vittorio, Fabio, Miki – per la complicità e l'affiatamento che ci legano, per il sostegno disinteressato che ci diamo e per tutte le esperienze che ancora abbiamo da vivere assieme (tra cui Praga 2028 che è sempre più vicina). A Miki, ancora, con cui condivido quotidianamente gioie e dolori dell'essere fuori sede. A Silvia, assieme alla quale non vedo l'ora di tornare a suonare e a ridere. A Marcello, con il quale ancora oggi posso dialogare sia di massimi sistemi che di scemenze colossali. Ai *Cittadini d'Abruzzo*, per la vostra spensierata compagnia.

Ai miei genitori e ad Andrea, che mi hanno sostenuto nei momenti peggiori e celebrato con me quelli migliori, e a cui voglio tanto bene.

Ad Alessandra, che riesce a risvegliare la parte più giocosa, vivace ed emotiva di me. Il tuo supporto e il tuo amore, che sai essere ricambiati con lo stesso impegno e la stessa forza, sono la mia grande fortuna.

A tutti voi, grazie.

Table of Contents

List of Tables

List of Figures

Abbreviations

BMS	battery management system
BTMS	battery thermal management system
CAN	controller area network
CC	constant current
CG	center of gravity
CNN	convolutional neural network
CV	constant voltage
DD	data-driven
DL	deep learning
DMB	direct measurement-based
DOD	depth of discharge
DVA	differential voltage analysis
ECM	equivalent circuit model
EChM	electrochemical model
EIS	electrochemical impedance spectroscopy
EM	empirical model
EOL	end of life
EV	electric vehicle
FC	fully connected
GBM	generic battery model
GRU	gated recurrent unit
I	current
ICA	incremental capacity analysis
IndRNN	independent recurrent neural network
LIB	lithium-ion battery
LSTM	long short-term memory
MAE	mean absolute error
MB	model-based
ML	machine learning
MR	MiniRocket
MSE	mean squared error
NN	feed-forward neural network
OBD	on-board diagnostics
OLS	ordinary least squares
PCA	principal component analysis
PI	proportional-integral
PPV	proportion of positive values
RF	random forest
RMSE	root mean squared error
RNN	recurrent neural network
RR	ridge regression
SEI	solid electrolyte interface
SOC	state of charge
SOH	state of health
SSE	sum of squared errors
T	temperature
TS	Theil-Sen
V	voltage
VW	Volkswagen

Chapter 1

Introduction

In this chapter, a brief introduction on electric vehicles and lithium-ion battery packs is made. Exploring the main insights and challenges related to battery pack aging provides manufacturers and researchers with a motivation for designing new methodologies for a reliable estimation of the state of health of a battery pack – the main contribution of this thesis.

1.1 Electric vehicles

Mitigating climate change is considered one of the key challenges of our century. With the growing concerns regarding the causes and effects of climate change, global efforts are being made by governments to accelerate clean energy transition and reach net zero emissions [1]. It is estimated that the transport sector accounts for 27% of the global emissions of greenhouse gases [2] and, more specifically, road travel accounts for three-quarters of transport CO2 emissions [3]. Research and development towards a greener automotive industry has therefore gained world-wide momentum in recent years. **Electric vehicles** (EVs) have been widely accepted as a clean and reliable alternative to fossil fuel vehicles, both in private and public transportation sectors, and are expected to quickly take over the market in the upcoming years [4]. It is therefore important to investigate the main technologies that enhance the performances of an EV.

The core component of an electric vehicle is its rechargeable **battery pack**. It is the heaviest, largest and most expensive component of an EV [5]. As such, its design is critical in determining the energy efficiency and driving range of an EV. A battery pack is typically made up of many battery cells, connected in parallel and series. **Lithium-ion battery** (LIB) cells are currently the dominating technology in battery pack design, thanks to favorable properties such as high energy density and efficiency, low memory effects, long cycle life, low self-discharge rate, high

charging and discharging rate capability and – last but not least – plummeting costs of their composing materials. [6–8]. Li-ion battery cells are discussed more in depth in sec. 1.2.

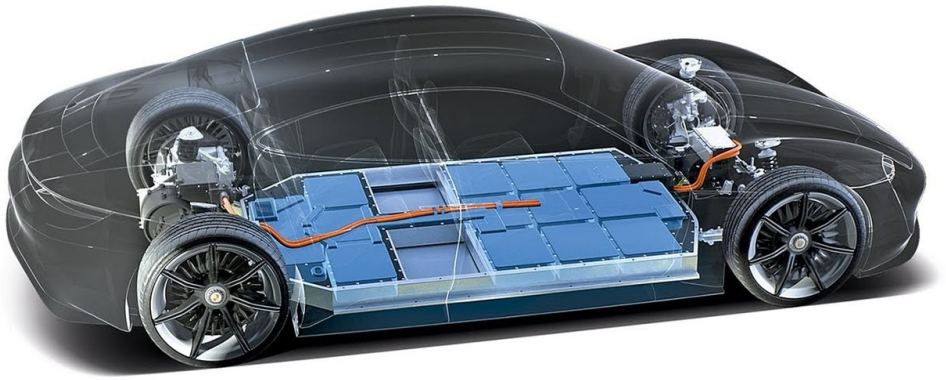


Figure 1.1: An electric car (Porsche Taycan) with its battery pack highlighted. Taken from [9]

A **battery management system** (BMS) is another important component in the design of an EV [10]. A BMS is an embedded software system which performs many functions related to the management of a battery pack, such as protecting the battery from operating outside its safe operating area (preventing overcharging and overdischarging, overcurrents, overheating), monitoring its state, calculating and reporting secondary data (such as the remaining driving range), balancing the state of charge among the battery cells and actively managing the battery pack's thermal management system (if present). All these operations are possible because of sensors installed inside the battery pack, which monitor parameters such as terminal voltage, current flow, internal temperature, state of charge and state of health (sec. 1.2.1) at the cell level. The BMS continuously monitors the sensors' measurements through a communication data bus, commonly known as Controller Area Network (CAN) bus. Moreover, these measurements can be communicated outside the vehicle through an interface called On-Board Diagnostics (OBD) port.

1.2 Li-ion battery cell

EV battery packs are hierarchically structured into three levels: cell, module and pack [7]. As visualized schematically in fig. 1.2, multiple battery cells are first connected together in series and parallel to form a battery module, and then a certain number of modules are assembled to form a battery pack. This design

has the aim of achieving a sufficiently high terminal voltage while adding up the capacity of each individual cell.

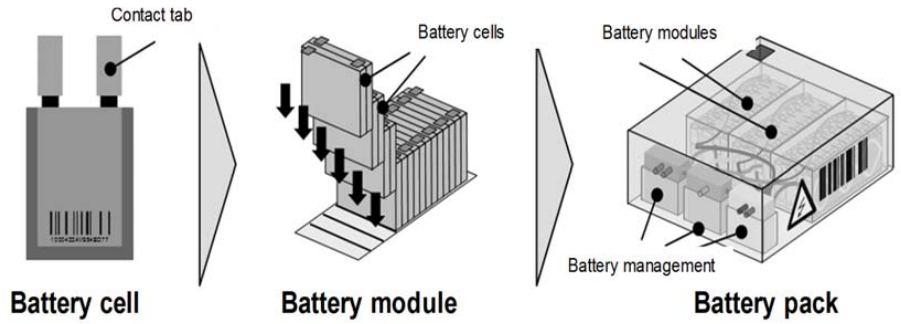


Figure 1.2: From battery cells to a battery pack. Taken from [11]

Battery cells come in different formats (pouch, cylindrical, prismatic) and chemistries (choice of the anode and cathode materials and electrolyte), which together determine the energy density and efficiency, terminal voltage, safety characteristics, costs, and – ultimately – performances. They are typically named according to their cathode materials: for example, lithium-ion battery cells are characterized by lithium-based cathode materials such as lithium cobalt oxide (LiCoO2), lithium iron phosphate (LiFePO4), lithium nickel manganese cobalt oxides ($LiNi_xMn_yCo_zO2$, or NMC-xyz for short) and so on [12]. Graphite is commonly used as anode material [13]. Cathode and anode are separated by a layer of electrolyte material, which is typically a polymer gel containing lithium salts[1]. The electrolyte has two functions: it prevents the electrical contact between the two electrodes and, at the same time, it allows the diffusion of lithium ions (Li^+) from cathode to anode and vice versa.

When a battery is being discharged, Li^+ ions are shuttled from the anode to the cathode, forcing electrons to flow around an outside circuit and thus allowing the conversion of chemical energy into electrical energy (fig. 1.3). On the contrary, charging a battery rips Li^+ ions out of cathode's oxide crystals and pulls them back to the graphite-based anode where they are stored [15].

[1]It is worth mentioning that recent advances in battery technology involve using a solid as the electrolyte material, with ceramics being the most promising [14]. Increased energy density, less need of thermal cooling and improved safety are among the many benefits of using a solid electrolyte.

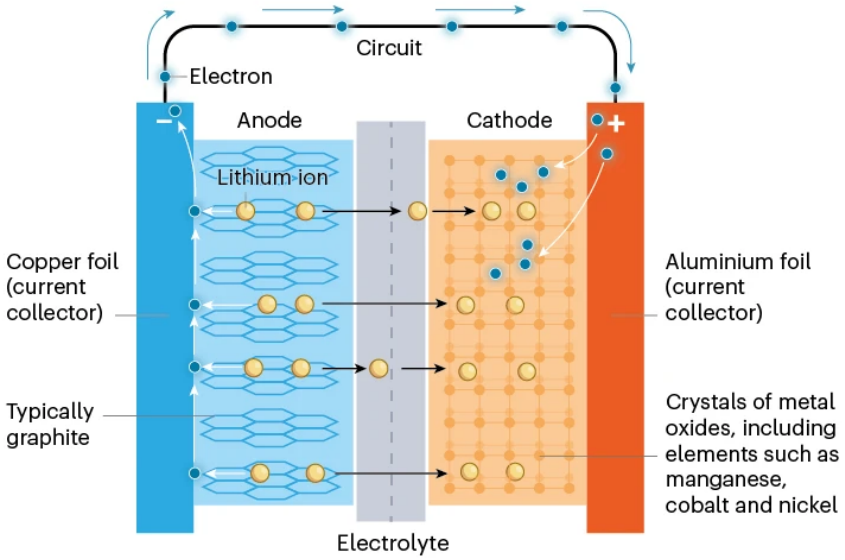


Figure 1.3: Discharging a Li-ion battery cell. Taken from [15]

1.2.1 Battery cell aging and SOH

Battery cells are characterized by many parameters describing their electrochemical state and dynamics. The **capacity** C of a battery is defined as the total amount of charge it can deliver and is commonly measured in ampere-hours [Ah]. Manufacturers usually declare a nominal (or rated) capacity $C_{nominal}$, i.e. the capacity of the battery when it is fresh out of the factory. For instance, a battery rated at 60 Ah can theoretically deliver a current of 5 A for 12 hours. The more electrode material is contained in the cell, the greater is its capacity. Nonetheless, the open-circuit voltage of a battery cell depends solely on its chemistry, regardless of its capacity [16]. The ratio between the amount of charge $Q_{available}$ that can be extracted from the battery at a given point in time and the actual capacity C_{actual} is called **state of charge**:

$$SOC = \frac{Q_{available}}{C_{actual}} \tag{1.1}$$

Actually, multiple external factors determine the effective battery capacity C_{actual} at any given moment. Therefore, capacity is typically defined up to a nominal set of operating conditions (or **nominal conditions**, for short), namely: external temperature equal to room temperature ($\approx$ 20 °C) and fixed discharge current magnitude. In fact, capacity tends to increase with high temperatures [17] and low discharge currents (a phenomenon called Peukert's law [18]).

When dealing with a current flowing through a battery, we can also express its magnitude in terms of **C-rate**. It is defined as the magnitude of the current

through the battery divided by the theoretical current draw under which the battery would deliver its capacity in one hour [19]. For instance, when referring to a battery with a capacity of 60 Ah, a current with C-rate 0.5 (or 0.5C current, for short) is a 30 A current.

Li-ion battery cells, like all batteries, are subject to degradation phenomena with time and usage, due to a variety of chemical and mechanical changes to the electrodes. These phenomena gradually lead to a decrease in capacity (a phenomenon called **capacity fading**) and an increase in internal resistance [20]. Other than normal usage in safe operating conditions, common causes of degradation include: shelving a full-charged battery for a long time, overcharging and overdischarging, storage or usage in very hot or very cold environments, high charge/discharge currents, discharging at high deltas of state of charge (depth of discharge, or DOD(%) for short). There exist a large number of degradation mechanisms[2], such as:

- thickening of the Solid Electrolyte Interface (SEI), a passivation layer formed on the surface of LIBs' anode material produced by electrolyte decomposition, in which Li^+ ions get irreversibly trapped (i.e. loss of lithium inventory)
- lithium plating, i.e. the deposition of lithium around the anode during charging
- loss of cathode and anode material due to its dissolution, cracking, exfoliation, detachment or volume change during usage
- corrosion and structural degradation of battery cell components, such as its current collectors.

Battery aging can be measured in a variety of ways. The most common way of quantifying the capacity loss of a battery is the **state of health** (SOH), defined as the ratio of the actual capacity of the battery to its nominal capacity:

$$SOH = \frac{C_{actual}}{C_{nominal}} \tag{1.2}$$

Another way of expressing the age of a battery is the number of **equivalent full cycles** (EFC). An EFC is defined as a virtual cycle of charge and discharge of a battery at a specified DOD [21]. The number of EFCs can be estimated as

$$EFC = \int \frac{|I|}{2 \cdot DOD \cdot C_{nominal}} \, dt \tag{1.3}$$

where I is the current flowing through the battery. It is easy to observe that the SOH of a battery decreases as the number of EFCs increases; however, there is no

[2]This list is by no means exhaustive: an in-depth discussion of degradation mechanisms for Li-ion battery cells can be found in [20].

general mathematical relationship between the SOH and the number of EFC, as the latter ignores operating conditions such as temperature and C-rate (as opposed to SOH), but also because SOH happens to not decrease uniformly over the lifespan of the battery [21, 22]. For instance, at $T = 25$ °C and a fixed 0.5C charging and discharging current a battery might undergo 3000 EFC at 100% DOD before reaching a SOH of 80%, while at $T = 45$ °C and 2C current the same battery might undergo 1200 EFC before reaching 80% SOH.

1.2.2 From cell SOH to pack SOH

The cells inside a battery pack are subject to uneven degradation over time, due to their asymmetrical placement inside the pack, uneven temperature distribution, different inter-cell contact resistances and possible subtle differences in their manufacturing quality [23]. As a result, each individual cell is characterized by a different capacity and internal resistance. This in turn causes current to flow unevenly among the cells, worsening the imbalance of cell degradation rates. To break this vicious cycle, the BMS tries to balance temperatures and currents inside a battery pack according to the estimated SOH of each cell [24, 25]. Due to cell capacity imbalance and the complexity of the BMS, estimating the SOH at the battery pack level in real time is not an easy task.

In the context of EVs, a battery pack is said to be at its **end of life** (EOL) when its state of health reaches 80% [26]. This threshold is commonly adopted since below 80% of its SOH a Li-ion battery pack incurs in a faster, typically more-than-linear degradation [27].

Monitoring SOH is a critical task, since the EOL of a battery pack typically marks the time when it has to be retired. SOH estimation can be performed in a laboratory setting, but in recent years researchers devoted their efforts to designing real-time and on-board procedures to allow for an easier and cheaper alternative for estimating SOH, as discussed in sec. 1.3 and chap. 2.

1.3 Proprietary SOH estimation procedures

Nowadays, a few private EV fleet management companies developed reliable manufacturer-independent, on-board diagnostic procedures for EV battery packs. Typically, they provide such battery tests to customers wishing to assess the degradation of a battery pack before buying or selling an EV.

Such SOH estimation procedures typically consist in connecting a monitoring device to the OBD port of the car and then driving the fully charged vehicle until the SOC drops below 10%. Meanwhile, the monitoring device continuously collects relevant operating data, which consists of measurements such as voltage, internal and external temperature, current, SOC, power consumption and so on,

sampled with a relatively high frequency and possibly at the cell, module and pack level simultaneously. Finally, this data is used as the input of a proprietary SOH prediction algorithm, which provides the customer with a reliable estimate of the SOH of their EV's battery pack.

Some of these companies retain the collected EV field data. This data is invaluable as it can be used for further analyses on the monitored battery, but can also be exploited for research purposes. Indeed, such data is extremely useful in designing data-driven methods to predict the SOH of a battery pack, as discussed throughout this thesis.

Chapter 2

State of the art

In recent years, research on battery SOH estimation – especially of Li-ion battery cells [28] – has been extensive and very diverse. In this chapter, a literature review of the existent SOH estimation methods is conducted.

2.1 SOH estimation methodologies

SOH estimation methods for Li-ion batteries mainly fall into three categories [28]: direct measurement-based methods, model-based methods, data-driven methods. This taxonomy can be further expanded by observing that some of these methods can be (or are meant to be) used in real-time with sensor data recorded while driving or charging an EV (online methods), whereas others can only be applied away from usual EV operations (offline methods) [29].

2.1.1 Direct measurement-based methods

Direct measurement-based (DMB) methods compute relevant quantities directly from experimental battery data and use them to estimate the SOH. Their main advantage is that they are relatively easy to implement and are generally computationally cheap.

Coulomb counting [30] is arguably the simplest offline DMB method. It estimates the SOH by integrating a discharge current signal over time, from full charge (SOC = 100%) to full discharge (SOC = 0%) of the battery. This value, which is the present capacity C_{actual} of the battery, is then divided by the nominal capacity of the battery $C_{nominal}$, thus following def. 1.2 exactly. This method has the advantage of being extremely cheap to apply, but its usage is typically confined to laboratory settings since EVs are typically never charged and discharged completely due to security hazards (namely overcharging and overdischarging [31]).

Electrochemical Impedance Spectroscopy (EIS) [32] is a DMB method for measuring a battery's internal impedance by imposing different galvanostatic or potentiostatic excitation frequencies. Internal impedance is frequently employed as an alternative SOH indicator of a battery, since it tends to increase as the battery ages[1] [34, 35]. Despite exhibiting remarkable performances in SOH estimation [32], this method is deemed infeasible for online applications as it is computationally costly and requires sophisticated equipment which cannot be easily embedded on an EV [28, 35].

Incremental capacity analysis (ICA) and differential voltage analysis (DVA) [36] are DMB methods based on the analysis of the evolution of a battery's incremental capacity and differential voltage curves. These curves are in fact correlated with battery aging (as shown in fig. 2.1), and can therefore be used to estimate SOH [37]. This method is computationally cheap, but it is only effective under low charge/discharge C-rates, a condition which cannot be ensured in real-world EV applications [38].

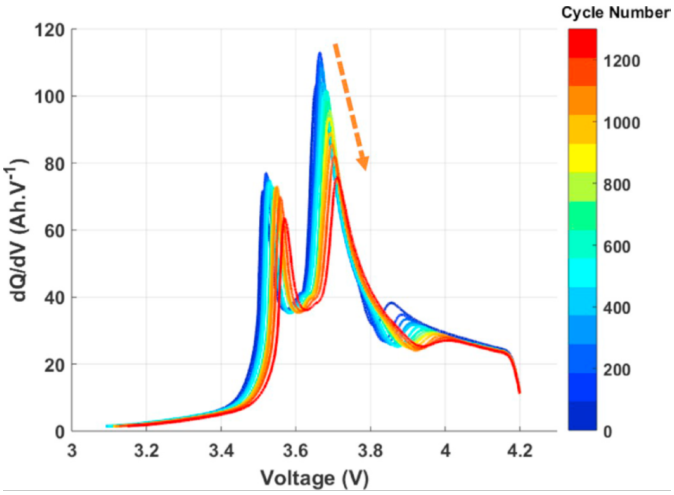


Figure 2.1: Incremental capacity curves for a high energy NMC/graphite cell under a charging current of 0.3C, after different charge/discharge cycles at ambient temperature. The peak intensity decreases and the peak location shifts to the right as more cycles are performed; these two quantities are therefore directly correlated with SOH. Taken from [39]

[1]When evaluating a battery's SOH in terms of internal resistance increase, the EOL of the battery is conventionally set to a 200% increase of the initial internal resistance [33]

2.1.2 Model-based methods

Model-based (MB) methods build a mathematical model of a battery and estimate its parameters by fitting the model's dynamics to experimental battery data. The fitted parameters can then be used to estimate the SOH directly (as in DMB methods) or indirectly (as in DD methods). A variety of models exist, which describe the battery at different levels of physical abstraction.

Equivalent circuit models (ECM) [28] are mathematical models of a battery based on electrical components such as resistors, capacitors and DC voltage sources, which describe the external characteristics of a battery [40]. A common ECM model is the n-th order RC model, which is schematized in fig. 2.2 The components of this

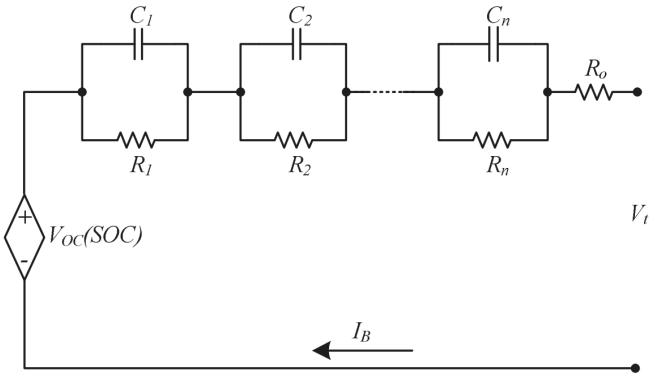


Figure 2.2: n-th order RC model of a battery. Taken from [41]

model are treated as look-up tables indexed by the battery's SOC, temperature and SOH. The parameters of an ECM model can be estimated through the available experimental data, by ensuring that the model's response to the experimental input current is compatible with the experimental output voltage. Increasing the order n makes the model more expensive to fit, but increase its ability to better adapt to experimental data. Note that an ECM model is a mathematical abstraction of a battery: the electrical components which compose it do not have any physical meaning, and the complex physical and chemical dynamics inside a battery are basically ignored. Moreover, the computational cost of fitting a n-th order RC model significantly increases with n, and a larger amount of experimental data in different SOC, SOH and temperature conditions would be needed (which is often not publicly available) [42].

Similarly to ECM, empirical models (EM) [29] are simple mathematical abstractions of a battery, which try to capture the non-linear relationship between SOH and degradation factors by means of a single function. Many possible degradation factors (such as temperature, age, SOC, ...) and many possible functional forms (linear, exponential, logarithmic, ...) can be chosen. However, just like ECM,

EM need an extensive amount of data in different experimental conditions to be fitted accurately; moreover, the choice of the functional form of the model and the degradation factors to consider are highly dependent on the particular battery chemistry and may lead to overfitting the particular experimental data available.

The electrochemical model (EChM) [28] tries to explicitly capture the complex physical and chemical phenomena which happen inside a battery and which cause capacity fading over time. The key assumption of this method is that the impedance spectrum of the battery is strongly related to the SOH. In practice, these physics-based electrochemical models generally achieve poor performances in the task of predicting the SOH of a battery, since a lot of physical and chemical properties must be identified and included in the model to accurately capture the dynamical behavior of a real battery [29]. A high expertise of the battery cell architecture and chemistry is needed to design such models.

2.1.3 Data-driven methods

Data-driven (DD) methods are a family of SOH approaches which leverage on large-scale datasets of experimental battery aging data, and whose performances are often dependent on the size and quality of such datasets [43]. DD methods are gaining increasing interest due to their extreme flexibility and variety. Many DD methods have an additional advantage of being domain-agnostic, i.e. they do not require domain expertise in the fields of electronics and chemistry [42], as they automatically learn the relationship between experimental data and SOH.

DD methods based on machine learning and deep learning are becoming the most prominent approaches to SOH estimation [29, 35, 42–44]. Different feature extraction techniques and different regression models are explored widely in literature [28]. Usually, these methods require a relatively long offline training phase, in which EV monitoring data measured by the BMS is preprocessed and used for training a machine learning model, but they are quite fast and accurate at prediction time. Due to these advantages, machine learning methods are particularly suited for real-time, on-board SOH estimation.

This family of methods share some similarities with direct measurement-based methods [43], as they both deal with experimental data recorded by sensors during battery's operating life. However, the key difference is that DM methods exploit recorded experimental data to compute specific hand-crafted features and use them to estimate the SOH directly, whereas DD methods automatically learn relevant features from raw data which are then used to train machine learning models, often without the need of a domain expert. Nonetheless, there can be a bit of overlap between the two families; for instance, in a previous thesis work [44] hand-crafted features were identified and extracted from raw EV monitoring data and used to train deep learning models for SOH prediction, thus taking advantage of both DM

and DD methods.

2.2 Literature review

In this thesis work, a novel real-time SOH estimation approach based on machine learning methodologies is designed. Due to the extreme interest shown by both academia and industry towards machine learning-based SOH estimation methods in recent years, the literature on the topic is incredibly vast. For our scopes, we specifically select those publications which share some similarities with our approach[2].

One common choice adopted by many researchers is to exploit experimental data acquired when charging [45–49] or discharging [47, 50] a battery cell in constant-current (CC) or constant-voltage (CV) modes. This choice is perhaps motivated by the particular stability of electrical signals such as voltage, SOC, and current during these operation modes, which allows for simpler estimation approaches. However, CC-CV discharge of a battery pack is feasible only in a laboratory setting, as these operation modes do not represent the randomized current load imposed to an EV battery pack during driving [51]; moreover, even when a battery pack is charged in CC-CV mode, its single cells are generally not charged in the same way, since the BMS is able to assign different currents to different cells, as discussed in sec. 1.2.2. Many works leverage on datasets which more accurately resemble real load conditions of EV battery packs, such as NASA Randomised Battery Usage Dataset [52–56] and Oxford Battery Degradation Dataset [57–60]. Raw experimental data acquired from sensors is typically not used directly; rather, the majority of the approaches found in literature extract relevant features which are assumed to be correlated with the SOH.

Roman et al. [61] proposed a manual feature extraction pipeline in which a total of 30 features are identified; recursive feature elimination based on random forests filters out the least relevant features for the SOH regression task and several regression models are trained on an augmented version of the resulting dataset. Among these models, random forest, deep neural network ensemble, Bayesian ridge regression and Gaussian process regression are explored.

Many other traditional regression models have been applied to SOH prediction pipelines. Among these: support vector regression [62], linear regression models (Ridge, Lasso, Elastic net) [63] and several variations of feed-forward neural networks (FFNN) [64–66].

[2]This literature review was conducted on several bibliographic databases, namely: MDPI, Science Direct (Elsevier), Springer, Wiley Online Library, IEEE Xplore.

Deep learning (DL) models are also widely explored. Chemali et al. [67] developed a SOH estimation method based on a convolutional neural network (CNN), which is given raw charging data (voltage, current and temperature over time) as input. The advantage of this methodology consists in getting rid of the manual feature engineering, as a CNN automatically learns how to extract relevant features from raw data.

More complex DL methods have been adapted to the task of predicting the SOH of a battery cell, namely Recurrent Neural Networks (RNN) [68], Long Short-Term Memory (LSTM) [69], Gated Recurrent Units (GRU) [70], independent recurrent neural networks (IndRNN) [71] and many more.

We highlight that all of these methods leverage on experimental data regarding single battery cells, mostly due to the unavailability of public datasets of EV battery packs' monitoring data. A further step needs to be taken in order to scale up from single battery cells to whole battery packs. Interestingly, Merkle et al. [72] introduced a digital battery twin for a 2014 Volkswagen e-Golf, setting up a data pipeline to predict the battery pack's SOC and SOH in real time. They acquired a training dataset directly from the OBD interface during a real drive cycle; however, this dataset has not been made publicly available. Song et al. [73] trained their SOH estimation methodology on a seemingly large public dataset of battery pack monitoring data collected by Shanghai Electric Vehicle Public Data Collecting, Monitoring and Research Center (SHEVDC). Despite their claim of being "the world's largest [public] platform for data collection and analysis of new energy vehicles", the Data Center requires researchers to apply for a membership in order to be granted access to the dataset[3].

To make up for the lack of publicly available EV monitoring data, a possible solution is generating such data artificially [6]. This can be achieved in a variety of ways; one possible solution is exploiting an EV model implemented on a simulation software such as Simulink [74–79] to simulate realistic drive cycles and gather synthetic electrical and mechanical signals from simulations. This is the approach followed in this thesis work: an existing EV model [79] is adapted (sec. 4.2) and then used (sec. 4.4) to generate a large dataset of voltage, current, temperature and SOC measurements. This dataset is then used to train a regression procedure for the SOH estimation (chap. 6).

[3]The application process seems unnecessarily long and rather obscure for a self-proclaimed "public" data center; for reference, it can be found on SHEVDC website: `http://en.cctp.org.cn/product/type/3849-8780-1.html`.

Chapter 3

Methodology

In this chapter, an overview of the concepts, algorithms and machine learning models encountered throughout this thesis work is provided.

3.1 Optimization problems

Everything in machine learning comes down to an optimization problem.

Optimization is the problem of finding one or more points that minimize or maximize a given objective function.

Definition 3.1.1 (Optimization problem)**.** Let $J : A \subseteq \mathbb{R}^n \to \mathbb{R}$. The problem of finding a global minimizer $\mathbf{x}^* \in B \subseteq A$ of J is an called optimization problem.

B is called the *search space*. When $A = B$, the optimization problem is *unconstrained*, whereas if $A \subset B$ it is *constrained*; in constrained optimization problems typically B is defined by a set of constraints, i.e. equalities and/or inequalities that points in B have to satisfy. Points in B are called *feasible solutions*, whereas all the possible $\mathbf{x}^*$ (if existent) are called *optimal solutions*.

There are a number of results which guarantee the existence and uniqueness of the optimal solution(s) of an optimization problem.

Definition 3.1.2 (Convex set)**.** A set $C \subseteq \mathbb{R}^n$ is convex if $\forall \mathbf{x}, \mathbf{y} \in C$ the segment joining $\mathbf{x}$ and $\mathbf{y}$ is contained in C:

$$\alpha\mathbf{x} + (1-\alpha)\mathbf{y} \in C \qquad \forall \alpha \in (0,1)$$

Definition 3.1.3 (Convex function)**.** Given a convex set $C \subset \mathbb{R}^n$, a function $f : C \to \mathbb{R}$ is convex if

$$f(\alpha\mathbf{x} + (1-\alpha)\mathbf{y}) \leq \alpha f(\mathbf{x}) + (1-\alpha) f(\mathbf{y}) \qquad \forall \mathbf{x}, \mathbf{y} \in C,\ \alpha \in (0,1)$$

If the inequality in the definition holds strictly, then the function is strictly convex.

Theorem 3.1.1. If f is convex on B, local minimizers of f in B are also global minimizers. Moreover, if f is also strictly convex on B, then there exist a unique global minimizer of f in B.

An optimization problem is (strictly) convex if J is a (strictly) convex function. In machine learning, the objective function J is usually a continuous non-linear function, and it is commonly called *cost function*.

Direct methods for solving non-linear optimization problems are often too impractical. Luckily, there exist a huge number of **iterative methods** for solving optimization problems, i.e. methods which construct a sequence $\mathbf{x}_0, \mathbf{x}_1, \dots$ which are asymptotically likely to converge to the optimal solution. Three of them are discussed hereafter: gradient descent (sec. 3.1.1), stochastic gradient descent (sec. 3.1.2), Nelder-Mead method (sec. 3.1.3).

3.1.1 Gradient descent

Gradient descent [80] is an iterative method in which the iterates $\mathbf{x}_k$ are obtained by moving along the local direction of steepest descent, i.e. $-\nabla J(\mathbf{x}_k)$:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \eta \nabla J(\mathbf{x}_k) \tag{3.1}$$

Basically, at each step the gradient descent algorithm locally builds a first-order approximation of J at $\mathbf{x}_k$, and then moves along the local direction of most rapid decrease of J (which is minus the gradient of J at $\mathbf{x}_k$) until a stopping criterion is met (e.g. threshold on $\|J(\mathbf{x}_{k+1}) - J(\mathbf{x}_k)\|$ reached and/or maximum number of iterations reached; both are user-specified hyperparameters).

In machine learning, the step-length $\eta \in \mathbb{R}$ is commonly known as *learning rate*. It is a user-defined hyperparameter which must be chosen carefully: a learning rate which is too high may make the method fail to converge, whereas a learning rate which is too low may slow down convergence. This is exemplified in fig. 3.1.

Typically, when the method is approaching the minimum of the function that is being optimized, we may want to decrease the learning rate to allow more fine-grained updates. There exists a huge variety of rules and schedules for the update of the learning rate [82]. Among them:

- Reduce η by a factor $\xi \in (0,1)$ when $\|J(\mathbf{x}_{k+1}) - J(\mathbf{x}_k)\|$ is below a specified threshold T for at least P iterations. ξ (learning rate decay factor), T (tolerance) and P (patience) are all user-specified hyperparameters (and they can be scheduled as well!). Typically implemented when training neural networks (sec. 3.6.3).

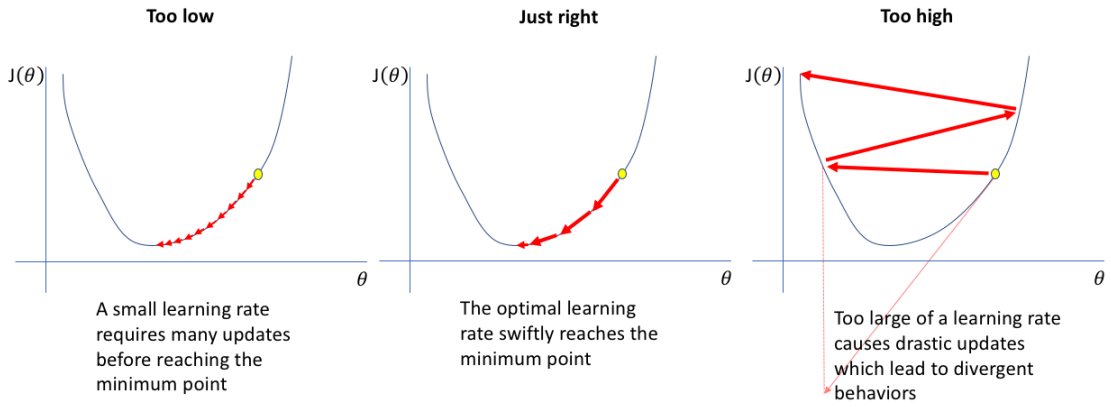


Figure 3.1: The importance of choosing an appropriate learning rate for the gradient descent algorithm. Taken from [81]

- Reduce η at each step by a factor $g(k)$ where g is a decreasing function of k (e.g. $g(k) = \frac{1}{k^p}$, with $p > 0$ hyperparameter). g is a user-specified hyperparameter. Used by default in scikit-learn when training ridge regression (sec. 3.6.1) via SGD.

Also the choice of the initial point $\mathbf{x}_0$ is of particular importance: choosing it to be near the effective minimum of the function may speed up the convergence of the method. However, quite obviously, this information is almost never available, and we must resort to random initialization.

Since non-linear functions are not convex in general, they may have many local minima, as shown in fig. 3.2.

A common problem that arises when using gradient descent on a non-linear objective function is that iterates may converge to a local minimum of J, especially when the initial point $\mathbf{x}_0$ is already closer to a local minimum than a global one. This issue can be avoided by resorting to other iterative methods which are more robust to the initialization of the starting point, e.g. stochastic gradient descent (sec. 3.1.2).

3.1.2 Stochastic gradient descent

In machine learning, the cost function J to be minimized is often built upon a dataset $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$ and has the form

$$J(\boldsymbol{\theta}; \mathcal{D}) = \frac{1}{N} \sum_{i=1}^{N} \ell(\boldsymbol{\theta}; \mathbf{x}_i, y_i) + R(\boldsymbol{\theta}) \tag{3.2}$$

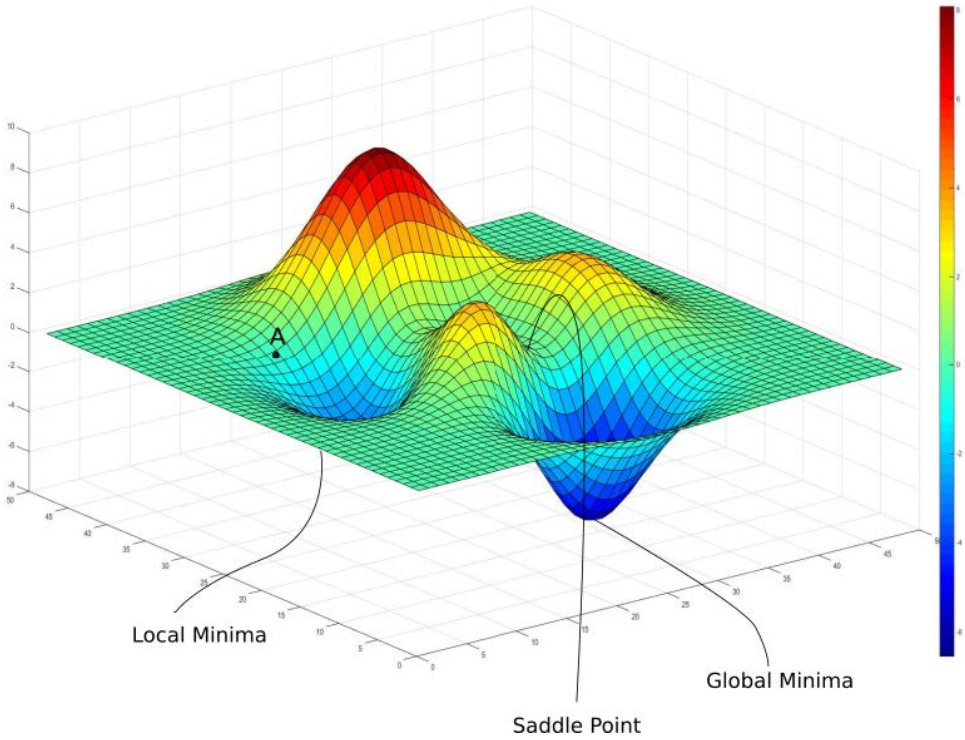


Figure 3.2: A non-convex bivariate function. Taken from [83]

where R is a regularization term and the function ℓ in each summand is commonly known as *loss function*, and is parametrized only by a single data point $(\mathbf{x}_i, y_i)$. For instance, the mean squared error (MSE) is a cost function which uses the squared difference between the predicted value $\hat{y} = f(\mathbf{x}; \boldsymbol{\theta})$ and the real value y as loss function: $\ell(\boldsymbol{\theta}; \mathbf{x}, y) = (f(\mathbf{x}; \boldsymbol{\theta}) - y_i)^2$.

Stochastic gradient descent (SGD) [84] is a variant of the gradient descent method in which at each iteration the cost function to be minimized is constructed only on a random subset $\mathcal{B}$ of $\mathcal{D}$ (called *minibatch*). This constitutes a stochastic approximation of the loss function. The size of each minibatch is an hyperparameter; a higher batch size ensures a better approximation of J (which therefore exhibits approximately the same minima of J), but determines a slower convergence due to the additional computational cost of computing the gradient. A *training epoch* (or simply *epoch*) refers to one sweep through the entire training set; therefore, a training epoch consists of $\lceil |\mathcal{D}|/|\mathcal{B}| \rceil$ iterations of the method[1]. At each epoch, the samples of each minibatch are randomly sampled from $\mathcal{D}$ without replacement.

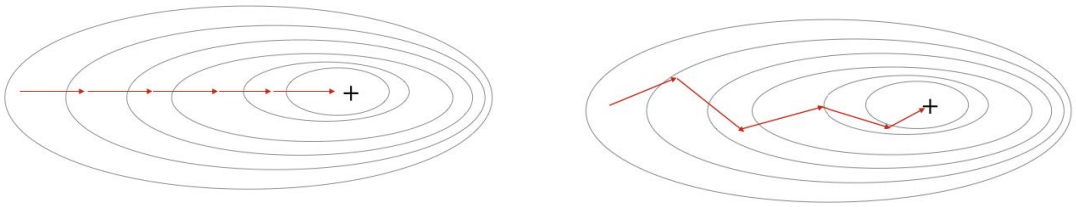
[1]There are $|\mathcal{D}|$ mod $|\mathcal{B}|$ samples left out from any minibatch; a typical choice is to ignore them during the current epoch

Iterations halt at the end of a user-specified number of epochs, or when other stopping criteria are met (e.g. the learning rate goes below a user-defined threshold).

SGD has several benefits over simple gradient descent:

- SGD is more robust to the initialization of $\mathbf{x}_0$ than gradient descent, since the varying cost function at each iteration ensures that the method is less likely to get stuck in local minima.
- SGD typically requires more iterations than gradient descent to converge to a minimum, but a single iteration of SGD is computationally cheaper than those of gradient descent. Overall, SGD is faster than gradient descent in terms of computational time, especially when $\mathcal{D}$ is large.
- SGD can handle very large training sets, because during each iteration only a minibatch of data must be stored in memory.

Gradient descent and SGD are visually compared in fig. 3.3



(a) **(b)**

Figure 3.3: Minimizing a bivariate function through (a) gradient descent (b) stochastic gradient descent. Taken from [85]

SGD with momentum

SGD with momentum (SGD-M) [86] is a variant of the standard SGD method. It changes the usual iterative update of gradient descent and SGD to

$$\mathbf{v}_{k+1} = -\eta \nabla J(\mathbf{x}_k) + \mu \mathbf{v}_k \tag{3.3}$$

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \mathbf{v}_{k+1} \tag{3.4}$$

with $\mathbf{v}_0 = 0$ and where factor $\mu \in (0,1)$ is an hyperparameter. The term $\mathbf{v}_k$ is called *momentum*. This "perturbation" to the direction along which to perform the update has the effect of preserving some information about the previously chosen

directions[2] and performing an update also in those directions, ultimately making the descent path less noisy than standard SGD. A visual interpretation of SGD-M is reported in fig. 3.4.

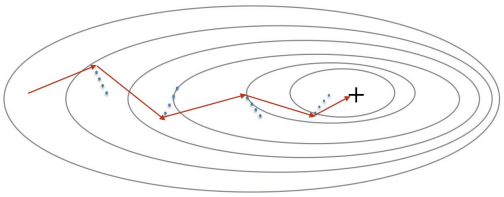


Figure 3.4: Minimizing a bivariate function through stochastic gradient descent with momentum. Taken from [85]

3.1.3 Nelder-Mead method

Nelder-Mead method [87] is a derivative-free iterative method for solving optimization problems. It is useful when we know how to compute the value of the objective function J but not how to evaluate its gradient (or when the gradient is too computationally expensive to compute).

Nelder-Mead method is based on the concept of *simplex*.

Definition 3.1.4 (Simplex)**.** A n-dimensional simplex $\mathcal{S}$ is the convex hull of $n+1$ points $\mathbf{x}_i \in \mathbb{R}^n$, i.e.

$$\mathcal{S} = \left\{ \mathbf{y} \in \mathbb{R}^n \;\middle|\; y = \sum_{i=1}^{n+1} \lambda_i \mathbf{x}_i,\ \lambda_i \geq 0,\ \sum_{i=1}^{n+1} \lambda_i = 1 \right\}$$

The points $\mathbf{x}_i$ $(i = 1, \ldots, n+1)$ are called vertices of $\mathcal{S}$.

$\mathcal{S}$ is said to be non-singular if $\mathbf{x}_2 - \mathbf{x}_1, \ldots, \mathbf{x}_{n+1} - \mathbf{x}_1$ are linearly independent. For instance, a line segment is a non-singular simplex in $\mathbb{R}^1$, a triangle in $\mathbb{R}^2$, a tetrahedron in $\mathbb{R}^3$ and so on.

[2]The term "momentum" has a meaning similar to that of momentum in physics: when an object is moving it is said to possess momentum, i.e. the product between its mass and (vectorial) velocity; if a force is applied to the object in a generic direction, the object gains momentum in that direction, losing the momentum it already possessed only due to friction. Similarly, in an update of the SGD with momentum we can view $-\nabla J(\boldsymbol{\theta}_k)$ as a force which gives the iterates "momentum" along its direction, and this momentum is preserved "by inertia", making the descent path more stable in comparison to that of SGD.

The basic idea of Nelder-Mead method is to start with a given simplex $\mathcal{S}_0$ and modify it at each iteration $(\mathcal{S}_1, \mathcal{S}_2, \dots)$ until a suitable simplex is found which provides a good approximation of the optimal solution of the problem.

A generic iteration k of the method is taken from [88] and reported hereafter. Let $\mathcal{S}_k$ be a given non-singular simplex and let $\mathbf{x}_1^{(k)}, \dots, \mathbf{x}_{n+1}^{(k)}$ be its vertices. Assume that the vertices are ordered in such a way that $f(\mathbf{x}_1^{(k)}) \leq \cdots \leq f(\mathbf{x}_{n+1}^{(k)})$ (so $\mathbf{x}_1$ is the "best" point of the simplex and $\mathbf{x}_{n+1}^{(k)}$ is the "worst"). We perform the following steps sequentially:

1. **Ordering phase**
   Evaluate f at the $n+1$ vertices of $\mathcal{S}_k$ and order the values $f(\mathbf{x}_i^{(k)})$ in increasing order $(f(\mathbf{x}_1^{(k)}) \leq \cdots \leq f(\mathbf{x}_{n+1}^{(k)}))$, and correspondingly order the vertices of $\mathcal{S}_k$.

2. **Reflection phase**
   Let $\overline{\mathbf{x}}^{(k)} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i^{(k)}$ be the barycenter of the n best points and compute a reflection $\mathbf{x}_R^{(k)}$ of $\mathbf{x}_{n+1}^{(k)}$ w.r.t. $\overline{\mathbf{x}}^{(k)}$:

   $$\mathbf{x}_R^{(k)} = \overline{\mathbf{x}}^{(k)} + \rho(\overline{\mathbf{x}}^{(k)} - \mathbf{x}_{n+1}^{(k)})$$

   If $f(\mathbf{x}_1^{(k)}) \leq f(\mathbf{x}_R^{(k)}) \leq f(\mathbf{x}_n^{(k)})$, then we accept $\mathbf{x}_R^{(k)}$ as a new vertex of $\mathcal{S}_{k+1}$ in place of $\mathbf{x}_{n+1}^{(k)}$ and go back to step 1, else go to step 3.

3. **Expansion phase**
   If $f(\mathbf{x}_R^{(k)}) < f(\mathbf{x}_1^{(k)})$ then compute an expansion $\mathbf{x}_E^{(k)}$ as

   $$\mathbf{x}_E^{(k)} = \overline{\mathbf{x}}^{(k)} + \chi(\mathbf{x}_R^{(k)} - \overline{\mathbf{x}}^{(k)})$$

   else go to step 4. If $f(\mathbf{x}_E^{(k)}) < f(\mathbf{x}_R^{(k)})$ then we accept $\mathbf{x}_E^{(k)}$ as a new vertex of $\mathcal{S}_{k+1}$ in place of $\mathbf{x}_{n+1}^{(k)}$ and go back to step 1. Otherwise accept $\mathbf{x}_R^{(k)}$ as a new vertex of $\mathcal{S}_{k+1}$ in place of $\mathbf{x}_{n+1}^{(k)}$ and go back to step 1.

4. **Contraction phase**
   Compute a contraction $\mathbf{x}_C^{(k)}$ between $\overline{\mathbf{x}}^{(k)}$ and the best among $\mathbf{x}_R^{(k)}$ and $\mathbf{x}_{n+1}^{(k)}$ as

   $$\mathbf{x}_C^{(k)} = \begin{cases} \overline{\mathbf{x}}^{(k)} - \gamma(\overline{\mathbf{x}}^{(k)} - \mathbf{x}_{n+1}^{(k)}) & \text{if } \mathbf{x}_{n+1}^{(k)} < \mathbf{x}_n^{(k)} \\ \overline{\mathbf{x}}^{(k)} - \gamma(\overline{\mathbf{x}}^{(k)} - \mathbf{x}_R^{(k)}) & \text{otherwise} \end{cases}$$

   If $f(\mathbf{x}_C^{(k)}) < f(\mathbf{x}_{n+1}^{(k)})$ then we accept $\mathbf{x}_C^{(k)}$ as a new vertex of $\mathcal{S}_{k+1}$ in place of $\mathbf{x}_{n+1}^{(k)}$ and go back to step 1. Else go to step 6.

5. **Shrinking phase**
   Compute
   $$\mathbf{x}_i^{(k)} = \mathbf{x}_1^{(k)} + \sigma(\mathbf{x}_i^{(k)} - \mathbf{x}_1^{(k)}) \qquad \forall i = 2, \ldots, n+1$$
   and go back to step 1.

A generic iteration of the Nelder-Mead method is visualized in fig. 3.5.

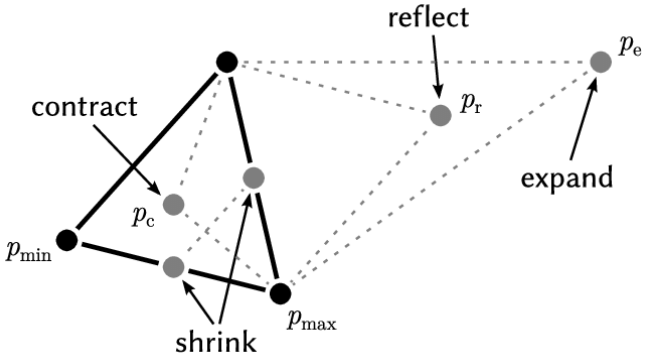


Figure 3.5: An iteration of the Nelder-Mead method applied to a function $J : \mathbb{R}^2 \to \mathbb{R}$. Taken from [89]

The method ends when a stopping criterion is met (e.g. maximum number of iterations reached or when $\left\| f(\mathbf{x}_{n+1}^{(k)}) - f(\mathbf{x}_1^{(k)}) \right\|$ is below a user-defined threshold). This method is based on the following 4 hyperparameters:

- ρ: reflection parameter ($\rho > 0$)
- χ: expansion parameter ($\chi > 1, \chi > \rho$)
- γ: contraction parameter ($0 < \gamma < 1$)
- σ: shrinking parameter ($0 < \sigma < 1$)

Typical choices are: $\rho = 1$, $\chi = 2$, $\gamma = 1/2$, $\sigma = 1/2$ [88].

The choice of the initial simplex $\mathcal{S}_0$ is also critical. In fact, an initial simplex that is too small can lead to a local search, and in this case the Nelder-Mead method can get more easily stuck.

Although there are no known convergence guarantees for Nelder-Mead method, it is nonetheless widely used because it is computationally cheap and does not require that J is differentiable (continuity is enough).

3.2 Time series

Monitoring a driving session of an EV means measuring and recording physical quantities such as vehicle speed, battery voltage, current, SOC and temperature over time. From a mathematical perspective, these discrete-time physical signals, along with the timestamps at which the measurements have been taken, can be treated as a multivariate time series.

Definition 3.2.1 (Time series)**.** A time series S is an ordered collection of T pairs of timestamps and tuples of C measurements

$$S = \{(t_1, \mathbf{s}_1), (t_2, \mathbf{s}_2), \ldots, (t_T, \mathbf{s}_T)\}$$

where $t_i \in \mathbb{R}^+$ and $\mathbf{s}_i \in \mathbb{R}^C$.

It follows that a time series can be thought of as a $C \times T$ matrix.

The number of timestamps, T, is also called *length* of the time series, and the number of measurements taken at each timestamp, C, is also known as the *number of channels* (or *dimensions*) of the time series; if $C = 1$, the time series is *univariate*, whereas if $C > 1$ it is *multivariate*. The ordered set of timestamps $t_1, \ldots, t_T$ is called *time base* of the time series.[3] If the time series has an unevenly spaced time base, it is called *unevenly spaced* time series.

An example of a 4-channel multivariate time series (extracted from a real set of EV monitoring data, chap. 5) is visualized in fig. 3.6.

Many time windows of monitoring data, along with the SOH value of the battery pack at the particular time the vehicle was driven, form a time series dataset.

Definition 3.2.2 (Time series dataset)**.** A time series dataset D is a collection of N instances (or *samples*)

$$D = \{(S_1, y_1), (S_2, y_2), \ldots, (S_N, y_N)\}$$

where S_i is a time series and y_i is a scalar value (called *ground-truth label*, *target variable*, *response variable* or *observation*).

The collection of time series composing the dataset, $(S_1, \ldots, S_N)$, can be thought of as a $N \times C \times T$ tensor.

[3] Def. 3.2.1 doesn't take into account those multivariate time series for which each channel has a different time base. However, we can still use the above definition by fixing a "shared" time base (the ordered union of the C different time bases) and opportunely adding *null* values where measurements are not available.

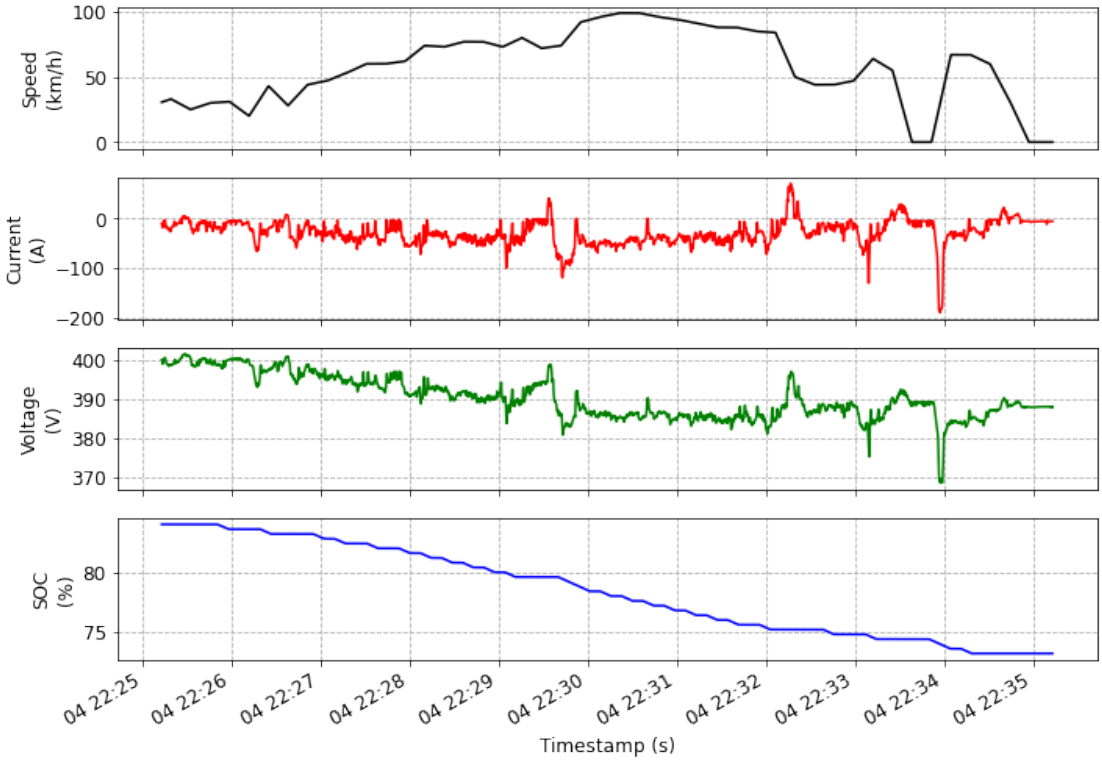


Figure 3.6: A multivariate time series with $C = 4$ channels.

3.2.1 Time series extrinsic regression

Time series datasets often show up in experimental settings. For this reason, in the past decade there has been an increasing interest in time series analysis research [90]. Common tasks in time series analysis are:

- Time series classification (TSC): the problem of assigning a label, taken from a finite discrete label set, to a time series
- Time series forecasting (TSF): the problem of predicting future values of a time series based on the past measurements; also known as time series regression (TSR)
- Time series extrinsic regression (TSER): the problem of assigning a scalar value to a time series

TSER is a generalization of both TSC and TSF. The difference between TSER and TSC is that the value to predict is discrete and categorical in TSC, and continuous in TSER; the difference between TSER and TSF is that in TSF the

scalar value to predict is some future measurement of the time series, whereas in TSER this assumption is relaxed.

We may frame the problem of predicting the SOH of the battery pack of an EV as a TSER problem: we seek to find a mapping from a time series of driving session monitoring data (with channels such as voltage, current, SOC, temperature, etc.) to the SOH of the battery pack (a scalar in the range [0,100]). In the context of machine learning, such a mapping is also called a *regression model*.

3.2.2 Time series feature extraction

In many experimental settings, the length of the recorded time series is too long in order to train a regression model directly on raw time series data: this is because each measurement is regarded as a feature, and many machine learning algorithms do not scale well with the number of features of a dataset [91]. Moreover, measurements taken at contiguous timestamps are typically correlated in some way, when dealing with continuous signals, meaning that some sort of feature selection is desirable to reduce this redundancy.

Therefore, it is often preferable to find a static feature-based representation of a time series [91]. In other words, we may want to find a way to transform a $N \times C \times T$ time series dataset into a static $N \times M$ features dataset (see fig. 3.7), and then train a regression model on this new dataset. These approaches to TSER problems are called *feature-based* regression algorithms [90].

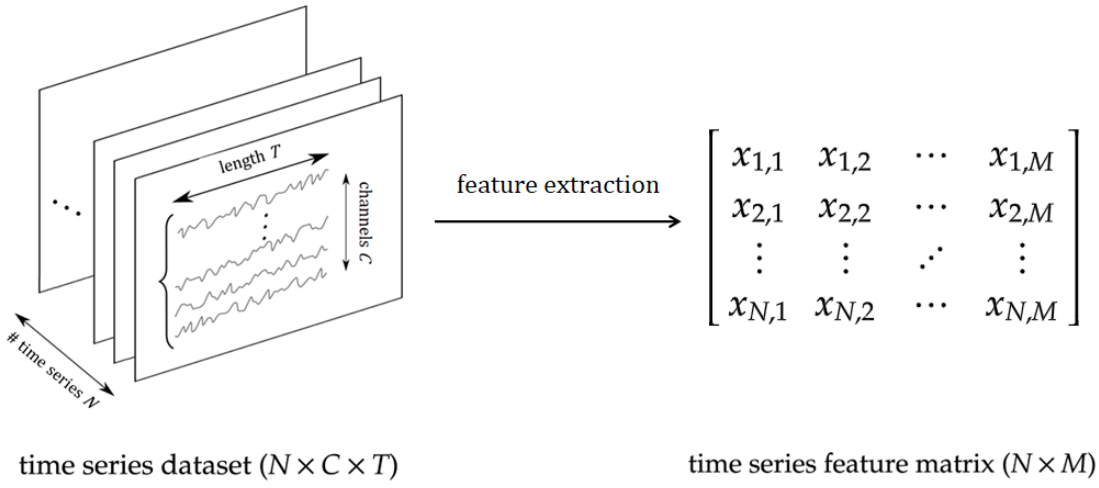


Figure 3.7: Time series feature extraction

Time series data differs from static data, which instead is not characterized by a temporal dimension. Since measurements in a time series are correlated in the temporal dimension, generally we cannot use feature extraction techniques

commonly used for static datasets, as they are typically not able to capture temporal relationships in data [91].

Time series feature extraction methods can be categorized into:

- Manual feature extraction: features are hand-crafted by a domain expert to capture relevant information of a specific problem. These domain-specific features are very relevant and discriminative for the problem at hand, but are often poorly generalizable to other regression problems.
- Automatic feature extraction: features are automatically extracted by an algorithm, in a supervised or unsupervised fashion (i.e. with or without the knowledge of the label associated to each time series in the dataset). Automatic feature extraction has the advantage of not requiring expertise of the data.

The majority of scientific papers on SOH estimation of Li-ion battery cells profusely exploit hand-engineered features [44, 61, 73, 92–101], but recent developments point in the direction of automatic feature learning and extraction based on deep convolutional neural networks [53, 67, 102–104]; in the latter approach, the feature extraction step is integrated in the convolutional part of the network (meaning that feature extraction and model training are performed in parallel).

In this thesis work, automatic approaches to feature extraction are explored. Specifically, a recent domain-agnostic unsupervised feature extraction method called MiniRocket is studied (sec. 3.3) and applied to our SOH estimation problem. Moreover, a novel computationally-inexpensive feature extraction method has been developed (sec. 3.5) and applied, achieving surprisingly high performance on a synthetic dataset of monitoring data. The two estimation strategies are thoroughly compared in sec. 6.4.

3.3 Feature extraction through MiniRocket

In 2020, Dempster et al. [105] proposed Rocket, a new time series classification procedure that achieves state-of-the-art accuracy in TSC with a low computational expense. Rocket consists of two parts: an unsupervised feature extraction step which transforms time series using a large number of random convolutional kernels, and a classification step with a classification model of choice (by default a ridge regression classifier). One year later, the same authors reformulated Rocket into a new classification method, called MiniRocket (for **MINI**mally **R**and**O**m **C**onvolutional **KE**rnel **T**ransform) [106]. It is up to 75 times faster than Rocket, while being almost fully deterministic and maintaining essentially the same performances on TSC tasks.

By replacing MiniRocket's classification model with a regression model, MiniRocket becomes a time series regression algorithm, which is able to achieve

state-of-the-art performances in TSER problems [90].

We will now briefly describe how MiniRocket's feature extraction step works when dealing with univariate time series datasets. A naive extension to multivariate time series is then described.

3.3.1 Convolution

Convolution is a widely used transformation applied to a matrix, which consists in computing a locally weighted sum of its elements with weights given by a matrix called *kernel*.

Definition 3.3.1 (Discrete 2D convolution)**.** Given a matrix $X \in \mathbb{R}^{n \times m}$ (called *input feature map*) and a matrix $W \in \mathbb{R}^{h \times k}$ (called *convolutional kernel*), the discrete 2D convolution (or simply convolution) between X and W is the matrix Z (called *output feature map*) whose elements are given by the following operation

$$Z(i,j) = (X * W)(i,j) = \sum_{r=1}^{h} \sum_{s=1}^{k} X(i+s-1, j+r-1) W(r,s)$$

Usually, a bias value $b \in \mathbb{R}$ is added to the convolution output in many applications.

Remark. When $m = k = 1$, the discrete convolution becomes 1-dimensional. This is the operation applied to univariate time series (which can be thought of as $1 \times T$ matrices, or equivalently as vectors of length T).

Convolution has a simple visual interpretation, visualized in fig. 3.8: the kernel matrix "slides" over the input feature map vertically and horizontally, locally identifying a submatrix on the input feature map of the same shape as the kernel; the elements of the output feature map are computed by taking the element-wise product of the kernel and said submatrix and summing them.

In many applications, richer variants of the convolution operation are used, which employ the techniques of *zero-padding* and *dilation* [108]:

- Zero-padding (or simply padding) consists in concatenating rows/columns full of zeros at the beginning or the end of an input feature map, prior to applying convolution. typically, the number of leading and trailing rows/columns added is the same (p). A zero-padded convolution is visualized in fig. 3.9a
- Dilation consists in "spreading" a kernel over the input feature map, such that with a dilation parameter of d the weights in a kernel are convolved with every d-th element of an input feature map along each direction. A dilated convolution is visualized in fig. 3.9b.

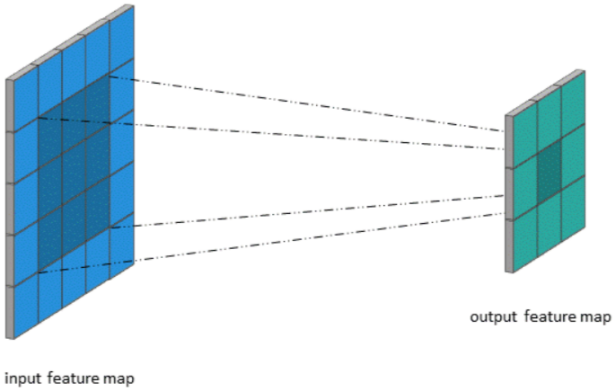
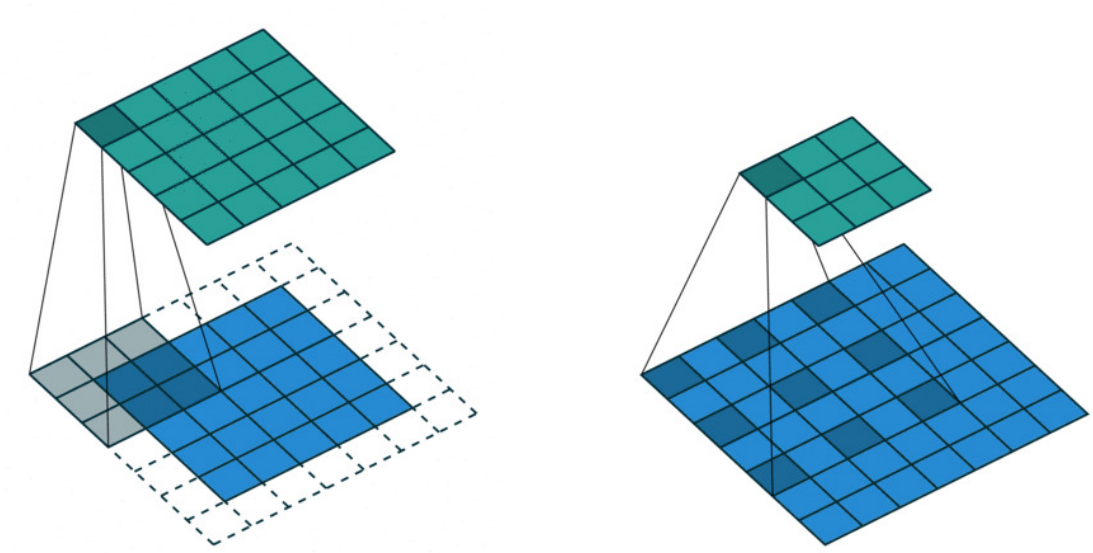


Figure 3.8: A visual representation of a discrete 2D convolution between a 5×5 input feature map and a 3×3 kernel. Taken from [107]



(a) Zero-padded convolution with padding parameter $p = 1$

(b) Dilated convolution with dilation parameter $d = 2$

Figure 3.9: Variants of the convolution operation between a 5×5 input feature map and a 3×3 kernel. Taken from [109]

3.3.2 Fitting MiniRocket's feature extractor

Length MiniRocket uses a fixed set of 84 kernels of length 9, with weights restricted to two values. These 84 kernels are selected a priori among the $2^9 = 512$ possible two-valued kernels of length 9. Using such a low number of kernels helps

in keeping the computational cost low enough while not hampering performances.

Weights The 84 chosen kernels have weights restricted to two values: $\alpha = -1$ and $\beta = 2$. This choice is made arbitrarily by the authors, in the sense that the scale of these values is unimportant, as bias values are drawn from the convolution output (as discussed in the next paragraph) and so they match its very scale. Thus, it is not necessary to normalise the input time series.

Moreover, the number of -1s is double the number of 2s in all 84 kernels, such that the sum of the weights of a kernel is 0. This ensures that the kernels are sensitive only to the *relative* magnitude of the input values, i.e. that the convolution output is invariant to the addition or subtraction of any constant value to the input: $X * W = (X \pm c) * W$.

The position of the three $\beta = 2$ weights inside each of the 84 kernels is defined by the sequence $[(1,2,3), (1,2,4), \ldots, (6,8,9), (7,8,9)]$.

Bias Bias values associated to each kernel/dilation combination are used in computing PPV features, as discussed in the associated paragraph. For a given kernel W and dilation parameter d (in short: W_d), bias values are drawn from the quantiles of the convolution output between W_d and a randomly selected time series X_i in the dataset. The orders of the quantiles assigned to each W_d are computed deterministically with the following low-discrepancy sequence

$$i \cdot \frac{1+\sqrt{5}}{2} \bmod 1, \ \forall i \in [1,2,\ldots,n] \tag{3.5}$$

where n is the total number of desired PPV features to be extracted.

The number of biases to use with each kernel/dilation combination W_d (or, equivalently, the number of features that are extracted from the convolution output with kernel/dilation W_d) depend on d and on the length l of the time series X, as discussed in the next paragraph.

Drawing at random a time series X_i for the purpose of sampling the bias values is the only stochastic element of MiniRocket; remaining parameters either depend solely on the time series length or are intended to be kept at their default values.

Dilation Possible dilation for each kernel values are in the range

$$D = \{\lfloor 2^0 \rfloor, \lfloor 2^{\max/m} \rfloor, \lfloor 2^{2\cdot\max/m} \rfloor, \ldots, \lfloor 2^{m\cdot\max/m} \rfloor\} \tag{3.6}$$

where m is the maximum possible number of dilations per kernel (set to 32 by default) and $\max = \log_2(l-1)/8$ is the largest possible dilation parameter for a kernel of length 9 applied to a time series of length l (i.e. such that the *effective* length of the dilated kernel of length 9 is exactly l).

The count of each unique integer dilation value in D determines the number of PPV features to be computed per dilation (scaled according to the total number of desired PPV features, n, so that the sum of the scaled counts equals n), ensuring that exponentially more features are computed for smaller dilations.

Padding Zero-padding is applied alternately every other kernel/dilation combination such that, overall, half the combinations use padding and half do not. The number of leading and trailing zeros added to the time series is set in such a way that the convolution operation begins with the middle (5th) weight of a kernel centered on the first measurement of the time series and ends with the middle weight centered on the last measurement.

Features The feature extracted by MiniRocket from each time series X for each kernel/dilation combination W_d and each bias b is the proportion of convolution output values which are greater than b:

$$\mathrm{PPV}(X * W_d, b) \coloneqq \frac{1}{n_{\text{out}}} \sum [X * W_d > b] \tag{3.7}$$

where n_{out} denotes the number of elements of the convolution output vector $X * W_d$ and $[\mathbf{x} \in a]$ denotes the indicator function. PPV stands for *proportion of positive values*.

By default, MiniRocket represents each time series with a total of 9,996 PPV features (i.e. the nearest multiple of 84 – the number of kernels – less than 10,000).

3.3.3 MiniRocket for multivariate time series

The authors [106] also extended MiniRocket to multivariate time series in a basic way[4].

Suppose that the time series in the dataset have C channels. For each kernel/dilation combination, a random subset of $k = \lfloor 2^{\mathcal{U}(0,M)} \rfloor$ out of C channels is taken, where $M = \log_2(\min(C,9) + 1))$ and $\mathcal{U}(0, M)$ means taking a random sample from the uniform distribution on $(0, M)$.[5]

[4]The explanation of multivariate MiniRocket given in this section is based only on the code implemented by the authors (which can be found at `https://github.com/angus924/minirocket/blob/main/code/minirocket_multivariate.py`), as the extension to multivariate time series is not mentioned anywhere in the original paper [106].

[5]k is distributed according to the floor of a log-uniform distribution on $[1, M]$.

A random variable X is distributed according to a log-uniform distribution with support on $[a, b]$ if $\ln(X) \sim \mathcal{U}(ln(a), ln(b))$, where $\mathcal{U}$ indicates the uniform distribution. [110]

When drawing the biases for each kernel/dilation combination, kernel W_d is convolved with each of the k channels of X that have been randomly selected; then, the k convolution outputs are summed, and this vector is used to draw the biases with the procedure already discussed in sec. 3.3.2.

This variant of MiniRocket's feature extractor for multivariate time series "suffers" from at least two drawbacks with respect to the original, univariate version:

- A new stochastic element is added: the random sampling of the channels of the time series to be used in the feature extraction, for each kernel/dilation combination.
- The different channels of a time series do not have the same mean and standard deviation in general. Since the sum of several convolution outputs is taken, the time series channels must be normalized for this method to make sense mathematically (whereas original MiniRocket doesn't require time series to be normalized, as discussed in sec. 3.3.2).

3.4 Feature selection with PCA

Dealing with high-dimensional datasets (i.e. datasets with a large number of predictors) is often undesirable. Many machine learning models do not scale well to high-dimensional datasets due to sparseness in data (a phenomenon known as *curse of dimensionality*) and generally high computational cost. Moreover, predictors have some degree of mutual dependence in many experimental settings. For these reasons, it is often desirable to perform a **dimensionality reduction** of the data [111].

Principal Components Analysis (PCA) [112] is a method for performing a particular change of basis on the data, in which the axes of the new coordinate system are the set of orthonormal eigenvectors of the scatter matrix $A = X^T X$, commonly known as "principal components" of the dataset X. PCA is typically exploited as an unsupervised dimensionality reduction technique, since it has the nice property of maximizing the variance of the data when mapped down to lower dimensional spaces.

Suppose now that $a, b \in \mathbb{N}$; it can be proved that $\lfloor X \rfloor$ is a discrete random variable with support on $\{a, a+1, \ldots, b-1\}$ with probability mass function

$$p(n) = \ln\left(\frac{n+1}{n}\right) / \ln\left(\frac{b}{a}\right)$$

E.g.: when sampling a random subset of k out of $C = 3$ channels from a multivariate time series, with k distributed according to the above distribution, the probability masses of k are: $p(1) = 0.5$, $p(2) = 0.292$, $p(3) = 0.208$. Therefore, smaller subsets are more likely to be extracted.

The PCA problem can be expressed in many different ways, which can be proven to be equivalent. One of the most interesting statements is the following one.

Given a dataset $X \in \mathbb{R}^{N \times M}$, with data points $\mathbf{x}_1, \ldots, \mathbf{x}_N$, and assuming that data is zero-centered ($\frac{1}{N}\sum_{i=1}^{N} \mathbf{x}_i = 0$), find a set of $D \leq M$ orthonormal directions $\mathbf{w}_1, \ldots, \mathbf{w}_D$ by solving consecutively the following variance maximization problems:

$$\begin{aligned} &\underset{\mathbf{w}_1 : \|\mathbf{w}_1\|=1}{\arg\max} && \frac{1}{N}\sum_{i=1}^{N}(\langle \mathbf{w}_1, \mathbf{x}_i \rangle)^2 \\ &\underset{\substack{\mathbf{w}_k : \|\mathbf{w}_k\|=1 \\ \mathbf{w}_k A \mathbf{w}_j = 0,\ j=1,\ldots,k-1}}{\arg\max} && \frac{1}{N}\sum_{i=1}^{N}(\langle \mathbf{w}_k, \mathbf{x}_i \rangle)^2, \qquad k = 2, \ldots, D \end{aligned} \tag{3.8}$$

This problem is solved by choosing $\mathbf{w}_1 = \mathbf{u}_1, \ldots, \mathbf{w}_D = \mathbf{u}_D$, where $\mathbf{u}_1, \ldots, \mathbf{u}_D$ are the unit-norm eigenvectors associated to the largest D eigenvalues of the scatter matrix A; the maximum values of the objective functions of each problem are, in order, $\lambda_1, \ldots, \lambda_D$.

This formulation is of particular interest because of the following interpretation: $\mathbf{w}_1$ is the direction of a line onto which the variance of the projected points, $\langle \mathbf{w}_1, \mathbf{x}_i \rangle$, is maximized; $\mathbf{w}_2$, is a second direction which maximizes the variance of $\langle \mathbf{w}_2, \mathbf{x}_i \rangle$ but which is also orthogonal to $\mathbf{w}_1$ (i.e., the projected points onto $\mathbf{w}_1$ and $\mathbf{w}_2$ are uncorrelated); and so on with the remaining maximization problems, which yield other directions of variance maximization which are orthogonal to all the previously found directions. This interpretation is particularly evident in fig. 3.10

The proportion of the variance represented (or "*explained*") by each eigenvector can be calculated by dividing the eigenvalue corresponding to that eigenvector by the sum of all eigenvalues. This means that by reducing dimensionality with D principal components, the proportion of variance explained by these is $\sum_{i=1}^{D} \lambda_i / \sum_{i=1}^{M} \lambda_i$.

Performing dimensionality reduction through PCA is especially useful when using MiniRocket as a feature extractor, since it produces a very large number of features (9,996) from each input time series.

3.5 Feature extraction through linear regression in the V-I-SOC space

MiniRocket is a domain-agnostic feature extractor, meaning that it could be effective in many different experimental contexts. However, embedding a SOH estimation method on an EV requires that the procedure has a low memory footprint and – most critically – a limited computational cost, due to the need of estimating the SOH in real-time. When performing a SOH prediction, MiniRocket computes almost 10,000 convolutions between the time window of monitoring data given as

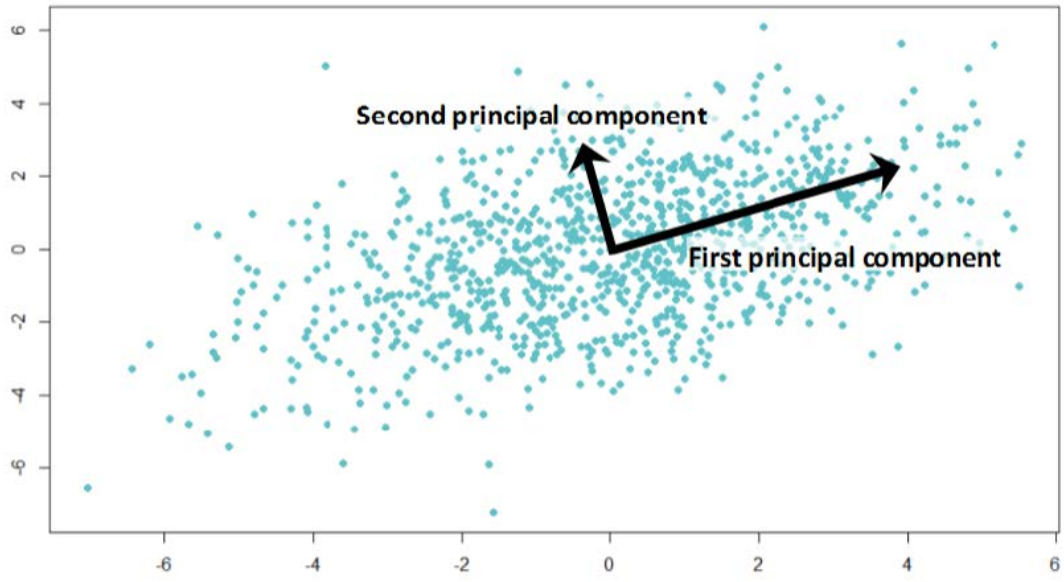


Figure 3.10: PCA on a simple 2D dataset. Taken from [113]

input and the MiniRocket kernels. As the length of the time windows increase, computing so many convolutions becomes infeasible for the computational devices embedded on a typical BMS of an EV.

In this section, a new feature extraction method is introduced. This method is:

- domain-specific: this method is specifically designed for the task of estimating an EV battery pack's SOH; it leverages only on the voltage (V), current (I) and SOC measurements recorded during a driving session and takes the shape of their data distribution into account;
- computationally cheap: for each time window of monitoring data, a simple multivariate linear regression model is fitted, an operation which requires only a limited amount of computation;
- low dimensional: from each time window of monitoring data only 3 features are extracted.

Some preliminary definitions are introduced in order to more easily explain how this method works.

EV field data typically consists of time series of V, I, SOC measurements (among others), taken during a driving session of the vehicle. From now on, we will refer to the tuple of measurements $(\mathrm{V}(t_i), \mathrm{I}(t_i), \mathrm{SOC}(t_i))$ at a timestamp t_i as an **operating point** of the EV.

The first key observation in the design of this novel procedure is that the operating points observed on a certain driving session seem to lie on a hyperplane in the V-I-SOC space. This observation is very accurate in relation to synthetic data (fig. 3.11a and 3.11b), but it is also true for real monitoring data with a certain degree of approximation (fig. 3.11c and 3.11d).

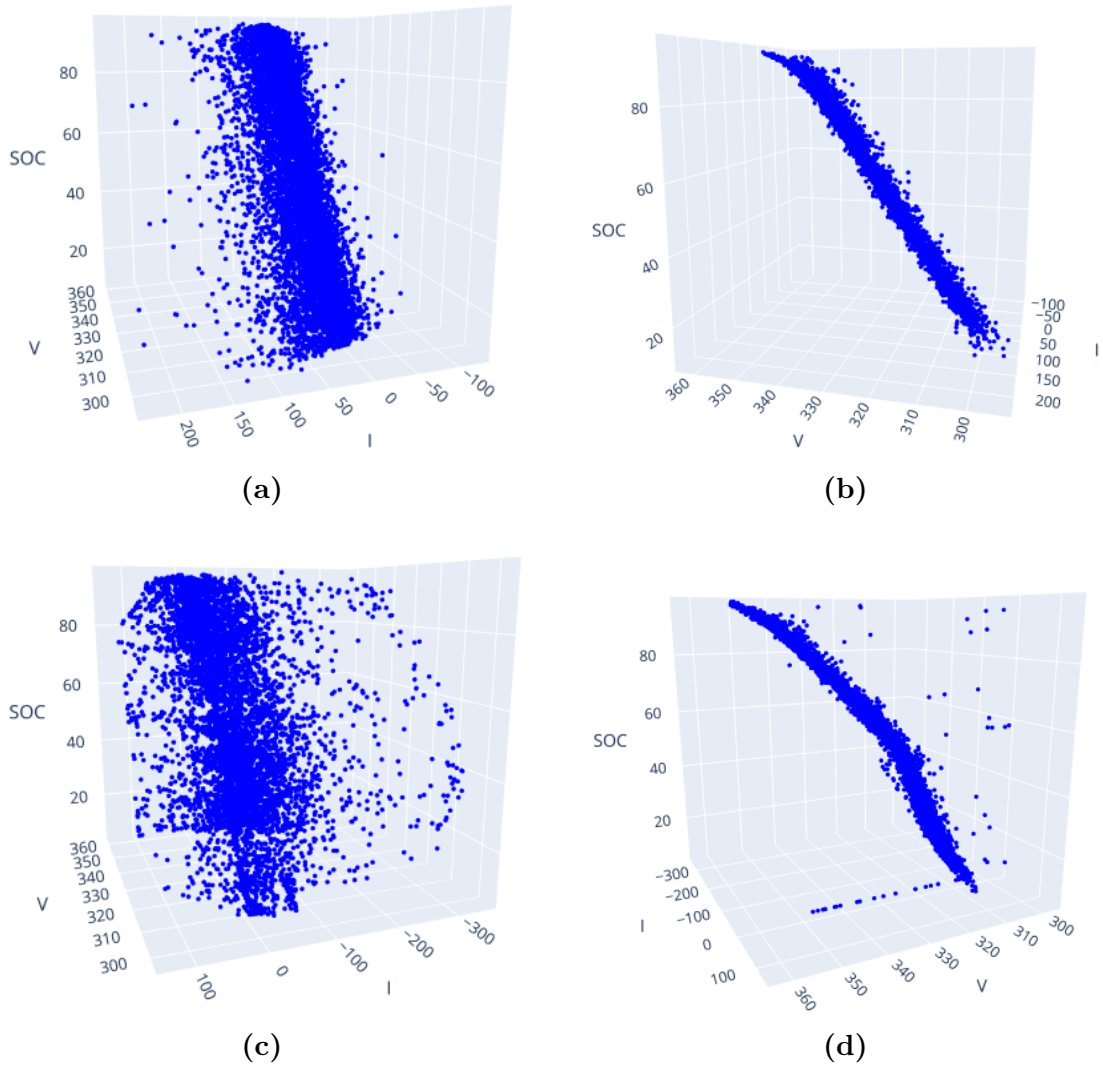


Figure 3.11: Operating points associated to the same SOH level, randomly extracted from: (a)-(b) synthetic dataset (sec. 4.4), (c)-(d) real dataset (chap. 5).

The particular hyperplane defined by the operating points seems to independent of the particular drive cycle, but interestingly it seems to be discriminative of the age of the battery during that driving session. In fact, the second key observation

is that different battery SOH values define different hyperplanes in the V-I-SOC space. This is highlighted in fig. 3.12.

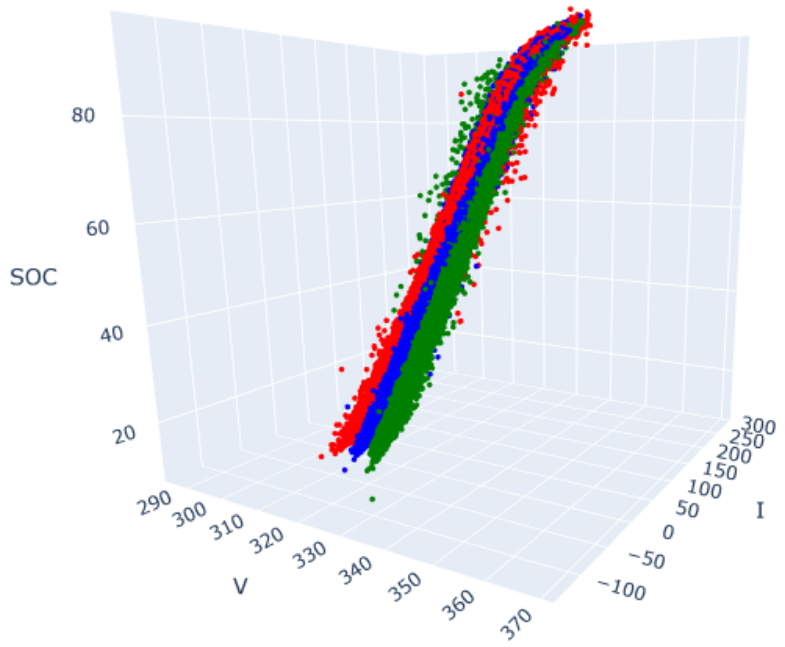


Figure 3.12: Operating points associated to different SOH levels (red: 100%, green: 90%, blue: 80%), randomly extracted from the synthetic dataset.

Therefore, we may argue that the problem of predicting the SOH of a battery pack is equivalent to the task of finding the hyperplane which approximates all the possible operating points of an EV at a specific point of its operating lifetime.

To make this task compatible with the requirements of a real-time SOH estimation method, we need to find said hyperplane from the knowledge of only a relatively short time window of operating points (i.e. a small sample of consecutive operating points) instead of the whole statistical population of operating points.

The equation of an affine hyperplane in a three-dimensional space is

$$z = ax + by + c \tag{3.9}$$

A 3D hyperplane is uniquely identified by its three parameters $a, b, c \in \mathbb{R}$.

Therefore, we can represent a time window of operating points with just three features, namely the a, b, c parameters defining the hyperplane which better "describes" those operating points.

Estimating the parameters of the hyperplane which passes near all the points in a given set can be stated as a linear regression problem [114].

Definition 3.5.1 (Multiple linear regression model)**.** Let $\mathbf{x} = (x_1, \ldots, x_M) \in \mathbb{R}^{\mathbb{M}}$. A multiple linear regression model (or simply linear model) is a function of the form

$$f(\mathbf{x}; \boldsymbol{\beta}) = \beta_0 + \sum_{i=1}^{M} \beta_i x_i$$

In many experimental contexts, a linear model is a reasonable approximation of the relationship between an output variable y and an input vector $\mathbf{x}$.

The problem of estimating the parameters $\boldsymbol{\beta}$ of a linear model from a set of training data $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$ is called *linear regression problem.* There exists many estimation methods for solving the linear regression problem. Two such methods have been explored for designing our novel feature extractor: ordinary least squares estimation (sec. 3.5.1) and Theil-Sen estimation (sec. 3.5.2).

3.5.1 Ordinary least squares estimation

Ordinary least squares (OLS) estimation is the most common solver for a linear regression problem. It consists in finding the parameters $\boldsymbol{\beta}$ minimizing the residual sum of squares

$$\mathrm{RSS}(\beta) = \sum_{i=1}^{N} (y_i - f(\mathbf{x}_i; \boldsymbol{\beta}))^2 \tag{3.10}$$

Let X the $N \times M + 1$ matrix having the dataset samples $\mathbf{x}_1, \ldots, \mathbf{x}_N$ as rows, with a constant $x_{i,0} = 1$ prepended to each row to account for the bias term, and let $\mathbf{y} = (y_1, \ldots, y_N)^T$. The analytical expression of the RSS can be rewritten as

$$\mathrm{RSS}(\boldsymbol{\beta}) = \|\mathbf{y} - X\boldsymbol{\beta}\|^2 = (\mathbf{y} - X\boldsymbol{\beta})^T(\mathbf{y} - X\boldsymbol{\beta}) \tag{3.11}$$

If the Gram matrix $X^T X$ is positive definite, then the OLS regression problem has a unique solution:

$$\hat{\boldsymbol{\beta}}^{OLS} = (X^T X)^{-1} X^T \mathbf{y} \tag{3.12}$$

The OLS method is visualized in fig. 3.13.

OLS generally yields a good estimate for the parameters $\boldsymbol{\beta}$ under the assumption that $\mathbb{E}(y|\mathbf{x})$ is a linear function with good approximation. However, OLS estimates are highly sensitive to outliers, since it seeks to minimize all residuals equally and penalizes large residuals more (due to squaring) [115]. In many experimental contexts the available data is contaminated by outliers, which for instance arise from measurement errors due to sensors (as is often the case in EV monitoring data), approximation errors (as is the case when resampling EV monitoring data to a common time base for all the signals through linear interpolation) or even unavoidable sampling error (which in the EV context is due to the fact that sensors

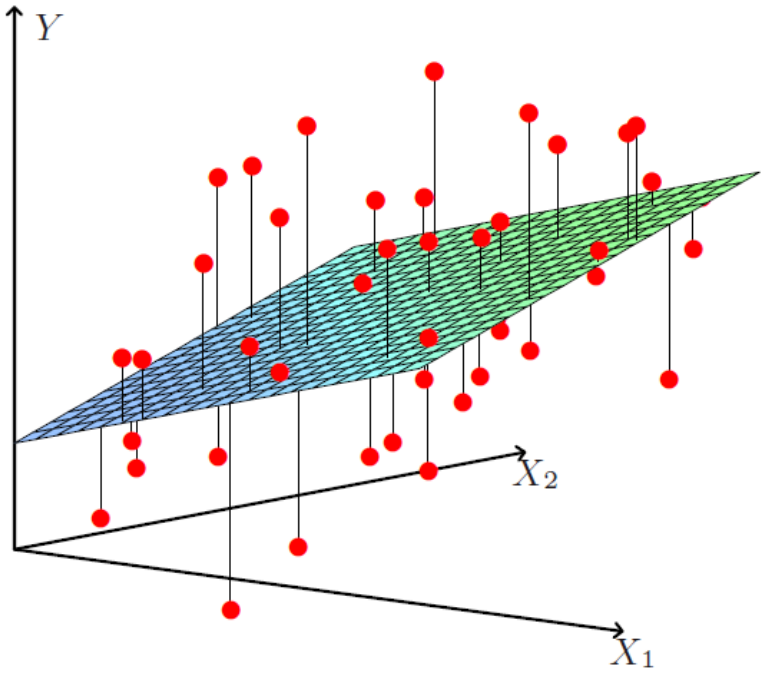


Figure 3.13: Fitting of a linear model on a 2D dataset by ordinary least squares estimation. We seek the linear function of **x** that minimizes the sum of squared residuals from y. Taken from [114]

monitor continuous signals by collecting them at a specific sampling rate, therefore often neglecting their sudden high-frequency variations).

Robust regression methods [115] are designed to limit the effect that the presence of outliers has on regression estimates.

3.5.2 Theil-Sen estimation

Theil-Sen (TS) estimation method is a robust alternative to the OLS method that reduces outliers' contribution to the error loss, thereby limiting their impact on regression estimates.

TS estimation was first introduced by Henri Theil [116] and Pranab K. Sen [117] as a method for fitting simple linear models (i.e. models with just one explanatory variable, $f(x; m, b) = mx + b$); recently, it has been generalized to multiple linear regression by Dang et al. [118].

Theil-Sen estimation for simple linear models

As originally defined by Theil [116], the TS estimate for the slope parameter m of the simple linear model on a set of two-dimensional points $(x_1, y_1), \ldots, (x_N, y_N)$ is the median of the slopes $m_{i,j} = (y_j - y_i)/(x_j - x_i)$ determined by all pairs of sample points. Sen [117] extended this definition to handle the case in which two data points have the same x coordinate, by taking the median of the slopes determined only by pairs of points having distinct x coordinates. The intercept b can be estimated in a number of ways; one of them is setting b as the median of the values $b_i = y_i - \hat{m}^{TS} x_i$.

It is clear that computing $\hat{m}^{TS}$ and $\hat{b}^{TS}$ with this procedure becomes very expensive as the dataset size N increases: for N points, $\binom{N}{2} = \frac{N(N-1)}{2}$ slopes must be computed – one for each pair of points.

One possible solution is sampling at random a subpopulation $S \ll \binom{N}{2}$ of pairs of points and performing Theil-Sen estimation on this reduced training set, effectively computing only S slopes. This has the effect of reducing the computational cost, but the estimation may become very sensitive to the particular random choice of the subpopulation, especially if S is very small. This variant of the Theil-Sen method is implemented in the Python library scikit-learn, where S is set with the hyperparameter `max_subpopulation` [119]. In this thesis work, S has been set to 10,000 for all experiments.

The Theil–Sen estimator is an unbiased estimator for the true slope m [117, 120], has high asymptotic efficiency [117, 121] and is more robust to outliers than OLS estimator [118]. In its original formulation (simple linear regression and $S = \binom{N}{2}$), it has a breakdown point of 29.3%, meaning that it can tolerate arbitrary corruption of up to 29.3% of the input data points without degradation of its accuracy.

A visual comparison between OLS and Theil-Sen estimation applied to a simple 2D dataset with outliers is reported in fig. 3.14.

On a final note, there is no closed form solution for TS estimates, as opposed to OLS; the algorithmic nature of TS method makes it well-suited for a computer implementation.

Theil-Sen estimation for multiple linear models

Recently, Dang et al. [118] extended Theil-Sen estimation to multiple linear regression.

Consider a multiple linear regression model $f(\mathbf{x}; \boldsymbol{\beta}) = \boldsymbol{\beta}\mathbf{x}$ (where, as usual, $\boldsymbol{\beta}, \mathbf{x} \in \mathbb{R}^{M+1}$ and $x_{i,0} = 1$). The hyperplane defined by the regression parameters $\boldsymbol{\beta}$ is determined by exactly $M + 1$ points. Similarly to the simple regression case, we can consider a (random) subpopulation of $S \leq \binom{N}{M+1}$ subsets of $M + 1$ points $(\mathbf{k}_1, \ldots, \mathbf{k}_S)$ and compute the unique $\boldsymbol{\beta}^{(i)}$ associated to each $\mathbf{k}_i$ by solving the

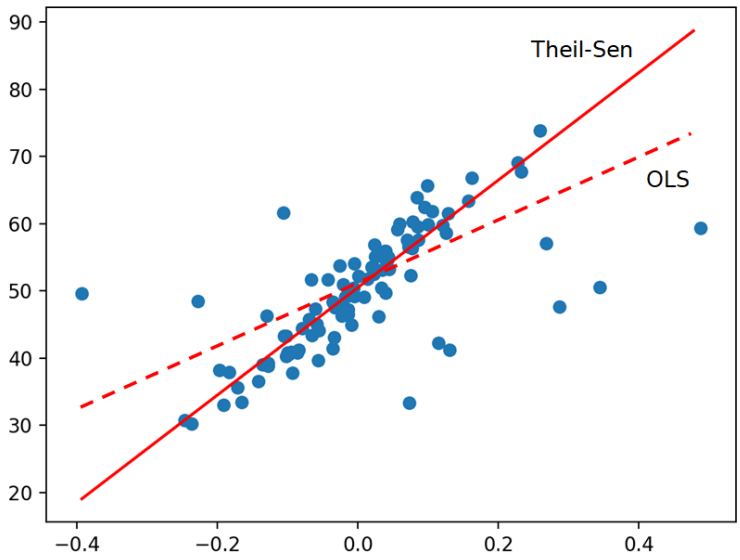


Figure 3.14: Fitting a simple linear regression model via Theil-Sen estimation vs ordinary least squares. Adapted from [122]

associated OLS problem[6]:

$$\boldsymbol{\beta}^{(i)} = (X^{(i)T} X^{(i)})^{-1} X^{(i)T} \mathbf{y} \tag{3.13}$$

where $X^{(i)}$ is the matrix whose rows are the vectors $\mathbf{x}_j \in \mathbf{k}_i$. Notice that with this procedure the "slope vector" $(\beta_1^{(i)}, \ldots, \beta_{M+1}^{(i)})$ and the intercept $\beta_0^{(i)}$ are computed simultaneously.

After having computed $\boldsymbol{\beta}^{(1)}, \ldots, \boldsymbol{\beta}^{(S)}$, the Theil-Sen estimate for $\boldsymbol{\beta}$ is their sample spatial median [123]

$$\hat{\boldsymbol{\beta}}^{TS} = \textsc{SpatialMedian}(\boldsymbol{\beta}^{(1)}, \ldots, \boldsymbol{\beta}^{(S)}) \tag{3.14}$$

For reference, the sample spatial median is discussed in appendix B.1

Multiple Theil-Sen estimation (MTS) is still robust to outliers, although a bit less than its simple counterpart, with a breakdown point of $1 - (1/2)^{1/(M+1)}$. E.g.: for $M + 1 = 3$, as in our experimental setting (see chap. 6), the breakdown point is 20.6%.

[6]Note that the OLS problem with exactly n points in a n-dimensional space is trivially equivalent to finding the (unique) hyperplane passing through those n points.

3.6 Regression models

Upon transforming a time series dataset to its static feature-based representation, we must choose and train a regression model to predict the SOH. A huge amount of regression models exist, both linear and non-linear; in this thesis work, we experiment with three well-known regression models: ridge regression (sec. 3.6.1), random forest (sec. 3.6.2), feed-forward neural network (sec. 3.6.3).

3.6.1 Ridge regression

Consider once more the linear regression model (def. 3.5.1). In some cases, the Gram matrix X^TX is singular or nearly-singular: this happens in particular when there are multicollinear (i.e. highly correlated) independent variables in the data. Because the OLS estimates depend upon $(X^TX)^{-1}$, when X^TX is nearly-singular the OLS estimation problem becomes ill-posed, and this may lead to serious numerical problems, namely estimates exhibiting high variance[7]. To mitigate this problem, we may impose a size constraint on the coefficients by introducing a L2 regularization term in the residual sum of squares:

$$\mathrm{RSS}_{\mathrm{L2}}(\boldsymbol{\beta};\lambda) = \sum_{i=1}^{N}(y_i - f(\mathbf{x}_i;\boldsymbol{\beta}))^2 + \lambda\sum_{i=1}^{M}\beta_i^2 \tag{3.15}$$

or, in matrix form:

$$\mathrm{RSS}_{\mathrm{L2}}(\boldsymbol{\beta};\lambda) = (\mathbf{y} - X\boldsymbol{\beta})^T(\mathbf{y} - X\boldsymbol{\beta}) + \lambda\boldsymbol{\beta}^T I_0\boldsymbol{\beta} \tag{3.16}$$

where I_0 is the $(M+1)\times(M+1)$ identity matrix with 0 as the first diagonal entry (i.e. $(I_0)_{1,1} = 0$) and $\lambda \in \mathbb{R}$ is a user-defined penalty factor (the larger, the stronger the penalization of large coefficients, as shown in fig. 3.15).

Note that the intercept β_0 is left out of the penalty term, as penalizing the intercept would make the procedure dependent on the scale of $\mathbf{y}$.

The linear regression problem solved by minimizing the L2-regularized RSS above is called **ridge regression** problem [114, 125]. Ridge regression has the following closed form solution

$$\hat{\boldsymbol{\beta}}^{\mathrm{ridge}} = (X^TX + \lambda I_0)^{-1}X^T\mathbf{y}$$

[7]For instance, let us suppose that a dataset has two independent variables "weight" and "height" and one dependent variable "age". It is reasonable to assume that weight and height have a strong linear correlation. When fitting a linear model to the dataset via OLS, we may end up with a very large positive coefficient for the weight and a similarly very large negative coefficient for height.

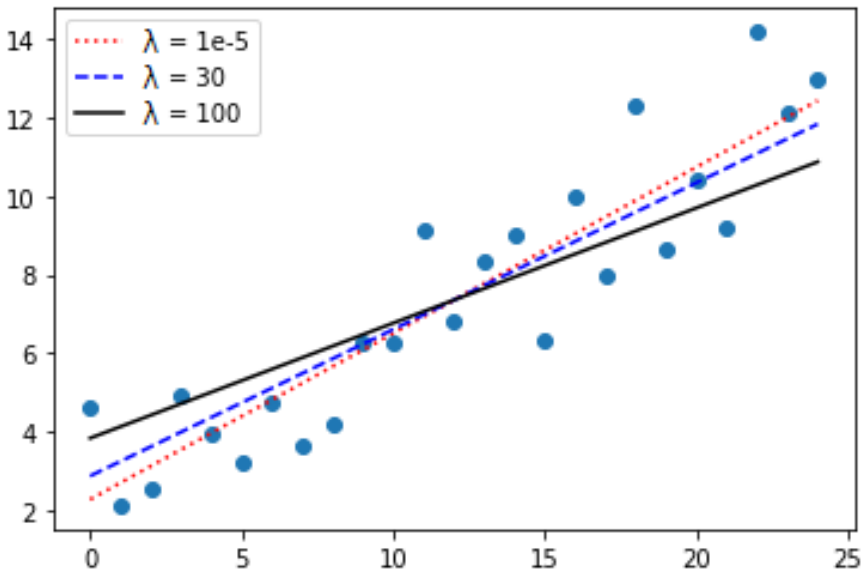


Figure 3.15: The effect of the choice of the penalty factor λ on the solution of the ridge regression problem. Adapted from [124]

As we can observe, a positive constant is added to the diagonal of X^TX before inversion: this makes the resulting matrix non-singular even when X^TX is, thus making the linear regression problem well-conditioned. Notice also that due to the addition of λI_0 ridge estimates are not equivariant under scaling of the inputs (as opposed to OLS), and so one normally standardizes the dataset before solving the ridge regression problem.

In many experimental contexts, X is a very large matrix. In such cases, the matrix X^TX and the inverse of $X^TX + \lambda I_0$ are very expensive to compute. For this reason, when N and/or M are particularly large, the ridge regression problem is solved iteratively via SGD (sec. 3.1.2).

3.6.2 Random forest

Random forest [114, 126] is an ensemble learning method for classification and regression tasks, which performs prediction through majority voting (classification) or averaging (regression) the predictions of an ensemble of decision trees. In this section, we will discuss only random forest regression.

A regression tree is a simple regression model that predicts the response variable y associated to a sample $\mathbf{x}$ by traveling from its *root node* to one of its *leaves*. At each node on the root-to-leaf path, the successor node is chosen on the basis of a splitting of the input space. Usually, the splitting is a binary "test" on one

of the features of $\mathbf{x}$. In this way, a decision tree tesselates the input space with non-overlapping hyper-rectangles ("boxes") in which a certain number (≥ 1) of training samples are located. A prediction is made by averaging the ground-truth labels of the training samples located in the input space box associated to the leaf reached. An example is shown in fig. 3.16.

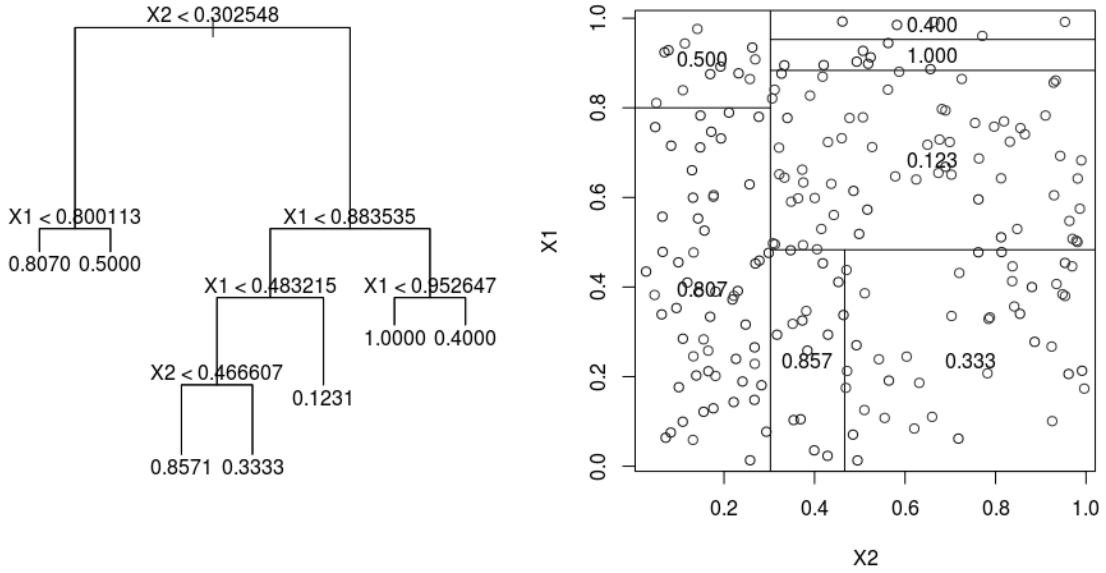


Figure 3.16: A regression tree trained on a 2D dataset (left), and the induced tesselation of the input space (right). Taken from [127]

Although many different regression tree learning algorithms exist, most of them follow the same approach, known as recursive binary splitting. This approach is:

- top-down: the tree is built recursively starting from the root and going down to the leaves
- greedy: at each recursive step the local best split is performed, rather than a locally suboptimal one which might lead to the globally optimal tree

Different learning algorithms use different metrics for defining the "best" predictor on which to split the training samples, and the associated test to perform on it. These metrics generally measure the "homogeneity" of the target variable within a set of samples. One of the most commonly used is variance reduction [128]. The variance reduction of a node is defined as the total reduction of the variance of the target variable y due to the split at this node; the predictor on which to split is the one which would cause the largest variance reduction.

This training process would continue to expand nodes by minimizing the chosen metric, until all leaves are homogeneous (i.e. all the training samples in the input space box associated to each leaf have the same response variable). Clearly, this may lead to overfitting the training data. To prevent it, we may decide to build the tree until a stopping criterion is reached: for example, we could fix the maximum depth of the tree as a hyperparameter of the learning algorithm (`max_depth` in scikit-learn [129]).

Regression trees are simple, highly interpretable and can be displayed graphically. However, they typically can't compete with other well-known regression approaches in terms of prediction performances. To reduce variance, prevent overfitting and hence increase performances, we may train a random forest, an ensemble learning technique that consists in growing multiple regression trees and then combining them to produce a prediction based on averaging the predictions given by each tree. It is based on two statistical techniques:

- Bagging (bootstrap aggregating) [130]: train an ensemble of B models on B different training sets, each obtained by randomly sampling N instances with replacement from the original training set of N samples; used to reduce the variance of a learning algorithm
- Feature bagging (random subspace method) [131]: at each split, the features on which to split are selected within a random subset of only $M' < M$ of the total set of M predictors (a typical choice is $M' = \sqrt{M}$); used to reduce the correlation between regression trees in an ensemble

Combining many trees usually results in dramatic improvements in performances, at the expense of some loss of interpretability. An example of a random forest is reported in fig. 3.17.

3.6.3 Feed-forward neural network

A **feed-forward neural network** (or simply neural network) is a non-linear model which can be used for regression tasks. The central idea is to extract linear combinations of the input variables as derived features, and then model the target as a nonlinear function of these features.

The "building block" of a neural network is the neuron, shown in fig. 3.18

Inside a neuron, the linear combination of an input vector $\mathbf{x}$ with weights $w_1, \ldots, w_M$ and bias b is computed, which is then used as the argument of a so-called activation function. The activation function is a non-linear function used to make the neuron a non-linear model.

Neural networks are modeled as collections of neurons that are connected in a directed acyclic graph (DAG). In other words, the outputs of some neurons can

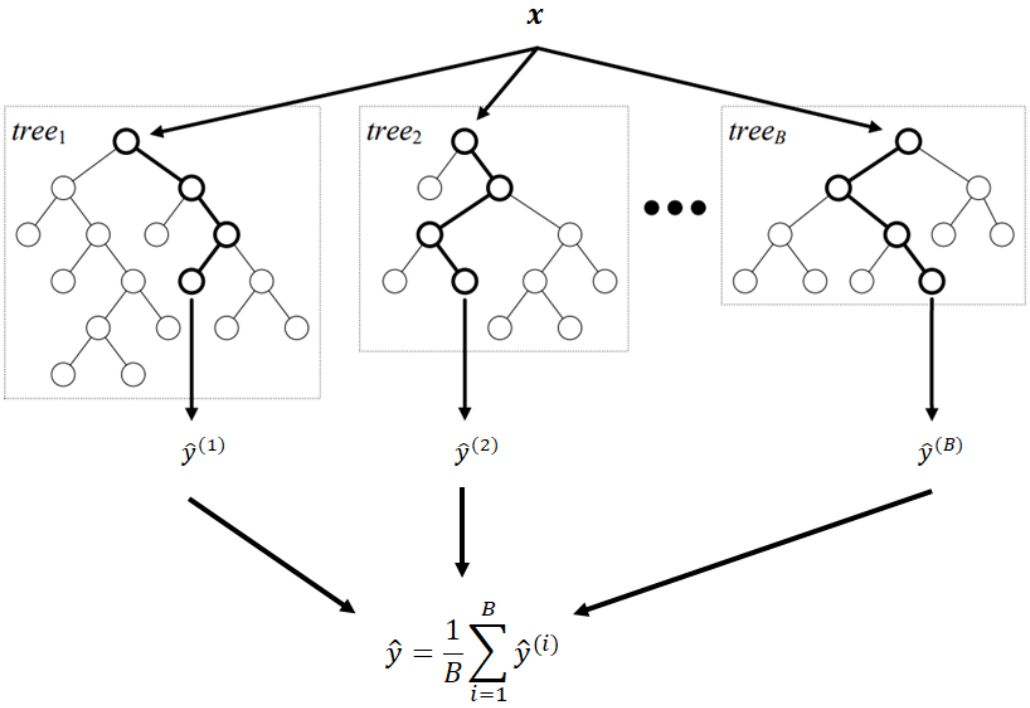


Figure 3.17: A random forest regression model with B regression trees. Adapted from [132]

become inputs to other neurons. The neurons of a neural network are organized into distinct layers of neurons, called fully-connected (FC) layers: each neuron of layer i is connected to all the neurons of layer $i+1$. The FC layers between the input layer and the output layer are called hidden layers. The neurons of an output layer typically have the identity function as activation function. An example of a neural network architecture is depicted in fig. 3.19.

Each hidden layer extracts multiple hidden features from the input. Use of multiple hidden layers allows construction of hierarchical features at different levels of resolution [114].

The number of layers L in a neural network, the number of neurons $N_1, \ldots, N_L$ in each layer and the activation function ϕ for each neuron are hyperparameters to be optimized via e.g. grid search cross-validation (sec. 3.8), random search or simply by trial and error. The complexity of a neural network is measured by the number of its free parameters (weights and biases), which in turn depends by L and $N_1, \ldots, N_L$.

Due to neurons being non-linear models (thanks to the non-linear activation function), the neural network too is a non-linear model. More complex neural networks are more expensive to train and to perform predictions with, but they

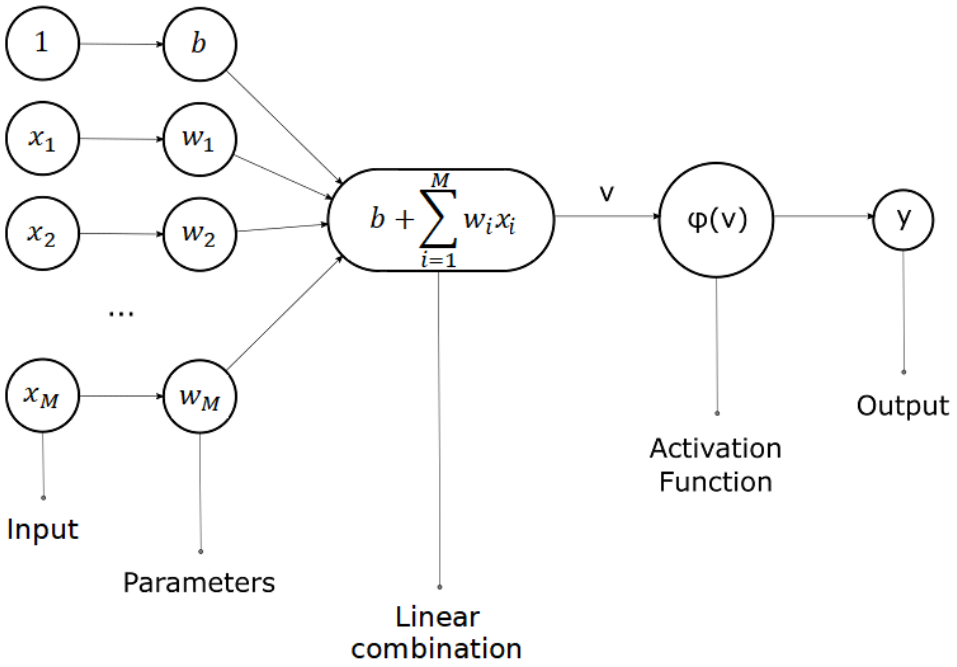


Figure 3.18: A neuron in a neural network.

have a higher representational power (i.e. they can learn functions which are more non-linear). On the other hand, neural networks which are too complex may overfit the training set, but this issue can be solved with proper regularization of the weights (as discussed in the next section).

Training

Training a neural network means estimating its weights and biases (collectively called "learnable parameters", $\boldsymbol{\theta}$) in such a way that the predicted values for the target variable associated to the training samples $\mathbf{x}_1, \ldots, \mathbf{x}_N$ are close to the ground-truth targets $y_1, \ldots, y_N$. More specifically, we seek to find $\mathbf{w}^{(1)} \in \mathbb{R}^{N_1}, \ldots, \mathbf{w}^{(L)} \in \mathbb{R}^{N_L}$ and $\mathbf{b} \in \mathbb{R}^L$ such that the L2-regularized mean squared error (MSE) cost function

$$\mathrm{MSE}_{\mathrm{L2}}(\boldsymbol{\theta}; X, \lambda) = \frac{1}{N} \sum_{i=1}^{N} (y_i - f(\mathbf{x}_i; \boldsymbol{\theta}))^2 + \frac{1}{2} \lambda \sum_{w \in \boldsymbol{\theta} \setminus \mathbf{b}} w^2 \tag{3.17}$$

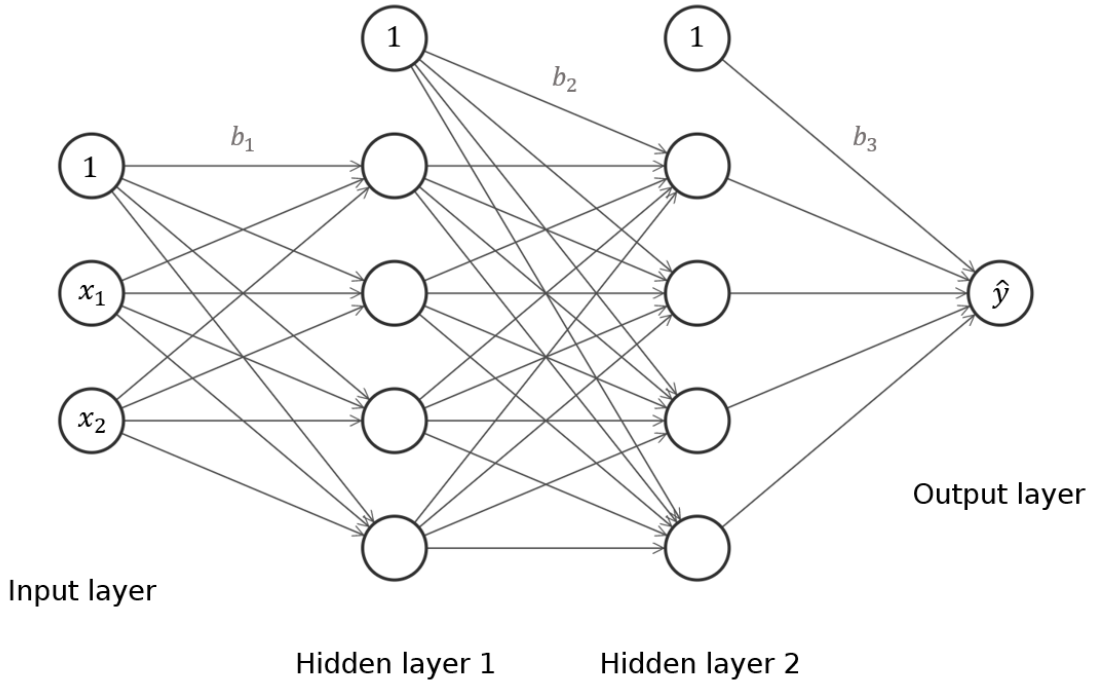


Figure 3.19: A feed-forward neural network with a 2D input and 2 hidden layers with 4 neurons each.

is minimized. The penalty factor λ (also called *weight decay* in the context of neural networks) measures how strongly large weights[8] are to be penalized. Choosing λ carefully is crucial to prevent overfitting the training set, as evident in fig. 3.20.

As f is a highly non-linear function in general, a closed form solution for the neural network learning problem is too impractical; rather, the problem is usually solved iteratively via SGD (sec. 3.1.2).

SGD requires the gradient of the cost function $\mathcal{L} = \text{MSE}_{L2}$ with respect to each learnable parameter ($\nabla_{\boldsymbol{\theta}}\mathcal{L}$) to perform an update. Due to the particular topology of a feed-forward neural network, there exists a very efficient way of computing $\nabla_{\boldsymbol{\theta}}\mathcal{L}$ analytically, called **backpropagation** [134]. Backpropagation computes each $\partial\mathcal{L}/\partial\theta$ through recursive application of the chain rule; its name suggests the idea that the prediction error is "propagated back" to the parameters "causing" it, and each $\partial\mathcal{L}/\partial\theta$ quantifies how much θ contributes to the error, thus how steeply we can update the parameter θ in order to minimize the error.

The depth of the neural network as well as the activation function must be

[8]Regularization is not applied to biases, for the same reason discussed in sec. 3.6.1 for ridge regression.

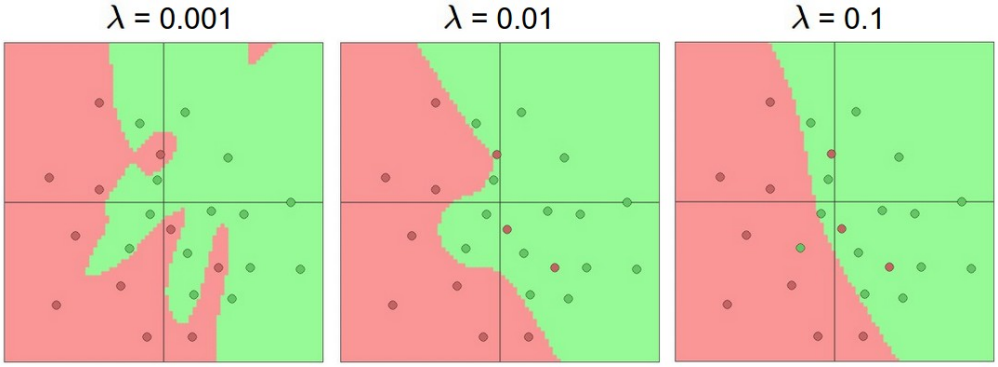


Figure 3.20: Decision boundaries of a neural network trained on a 2D dataset for a classification task, for varying values of λ. Taken from [133]

chosen carefully, to avoid the vanishing gradient problem [135]. In particular, the network should be not too deep, otherwise the gradient of the cost function with respect to early layers' parameters would likely get small (because of a long chain of products between many quantities smaller than 1) and they would be updated very slowly (the neural network would not "learn" anymore). Moreover, the activation function should not operate in regions where it has a derivative approaching zero, as this would effectively kill the gradients of the parameters preceding that activation function. A very popular choice for the activation function is the Rectified Linear Unit (ReLU) [136], $\phi(x) = \max(0, x)$, represented in fig. 3.21. ReLU has the advantage of being scale-invariant ($\max(0, ax) = a(\max(0, x))$, cheap to compute (only a comparison between 0 and x is needed) and non-saturating for $x > 0$, which mitigates the vanishing gradient problem. On top of that, ReLU seems to work very well in practice [137].

When training a neural network, it is common sense to standardize the data, as the scale of the training data has a relevant effect on the scale of the learned parameters and thus on the quality of the final solution. Having input variables scaled to zero mean and unit standard deviation implies that each feature will be deemed equally relevant in the learning process, and it can be shown that this improves the rate of convergence of training a neural network [138].

Lastly, also the particular initialization of the parameters affects the rate of convergence of training. In particular, the cost function $\mathrm{MSE}_{\mathrm{L2}}$ is nonconvex and typically exhibits many local minima; as a result the estimates of the parameters are very sensible to their initialization. The recommended heuristic for initializing weights when using ReLU activation functions is Kaiming-He initialization [137], which consists in drawing the initial value of each weight w randomly according to a normal distribution

$$w \sim \mathcal{N}\left(0, \frac{2}{n}\right)$$

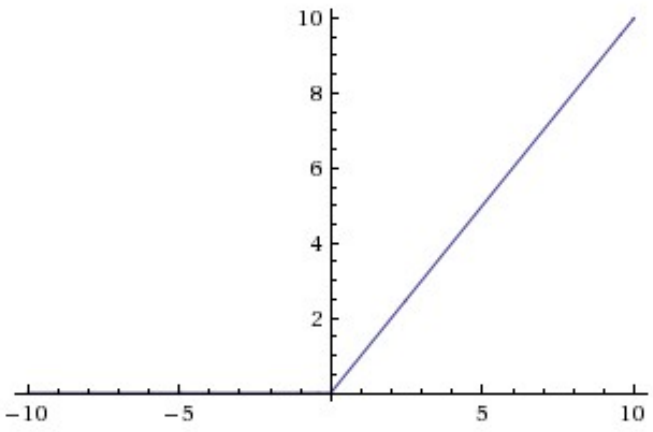


Figure 3.21: ReLU function. Taken from [133]

where n is the number of inputs to a specific neuron (i.e. $n = |\mathbf{w_{l-1}}|$ where l is the layer in which the neuron is located); biases, instead, are initialized to 0. It can be proved that Kaiming-He initialization ensures that all neurons in the network initially have approximately the same output distribution with variance 1 and empirically improves the rate of convergence of the training process.

3.7 Performance metrics for regression

To evaluate the performances of a regression model f, many different performance metrics can be used [139]. These metrics generally measure how good (or bad) f is at estimating the ground-truth target y of a sample, i.e. how close (far) are the predictions $f(\mathbf{x}_i; \boldsymbol{\theta}) = \hat{y}_i$ from y_i.

- Mean squared error (MSE)

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2$$

  It gives information on the prediction error that is committed on average. However, due to squaring, this metric does not have the same unit of measurement as the quantity that is being predicted; therefore, other metrics such as the RMSE or the MAE are usually preferred.

- Sum of squared errors (SSE)

$$\text{SSE} = \sum_{i=1}^{N}(\hat{y}_i - y_i)^2$$

Similar to the MSE but with a sum instead of an average operation.

- Root mean squared error (RMSE)

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2}$$

It is similar to the MSE, but has the same unit of measurement as the target variable; however, it has no straightforward interpretation.

- Mean absolute error (MAE)

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|\hat{y}_i - y_i|$$

It measures the average absolute difference between the predicted and real target values.

- Coefficient of determination (R^2)

$$R^2 = 1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \overline{y})^2}$$

where $\overline{y} = \frac{1}{N}\sum_{i=1}^{N} y_i$. It provides a measure of how well the predictions $\hat{y}_i$ approximate the true values y_i. An R^2 of 1 indicates perfect prediction, whereas a model that always predicts the average value of y disregarding the input features ($f(\mathbf{x}_i; \boldsymbol{\theta}) = \overline{y}$) would get an R^2 of 0. The R^2 can also be negative, because the model can do arbitrarily worse than said constant model, in terms of MSE. The R^2 is not defined when $\overline{y} = y_1 = \cdots = y_N$; in this case, we set $R^2 = 1$ for perfect prediction and $R^2 = 0$ for imperfect prediction.

3.8 *K*-fold cross-validation

To tune the hyperparameters which characterize a regression procedure[9], **(stratified)** K**-fold cross validation** can be used. It consists in randomly splitting the training set into K subsets, called *folds*. One fold is held out as a validation set, while the remaining $K - 1$ folds are used for training the model: each prediction

[9]By "procedure" we refer in general to the whole training pipeline of a regression model, including data cleaning, data preprocessing and feature extraction steps. In fact, also these earlier steps can be characterized by user-defined hyperparameters, which are to be optimized.

pipeline is fitted and trained on the $K-1$ training folds, and tested on the remaining validation fold, on which one or more performance metrics are evaluated (e.g. MSE or MAE). This procedure is repeated K times, in order to use each fold as a validation set; therefore a total of K regression models are trained. The resulting performance metrics are combined (e.g. averaged) over the K rounds to give an estimate of the model's predictive performance on unseen data.

Typically, a grid search over different hyperparameters is performed through cross-validation. The hyperparameter combination achieving the highest cross-validation score is considered the optimal one, and the regression procedure with the optimal hyperparameters is refit on the whole training set, thus obtaining the final regression model. The cross-validation procedure is visually summarized in fig. 3.22.

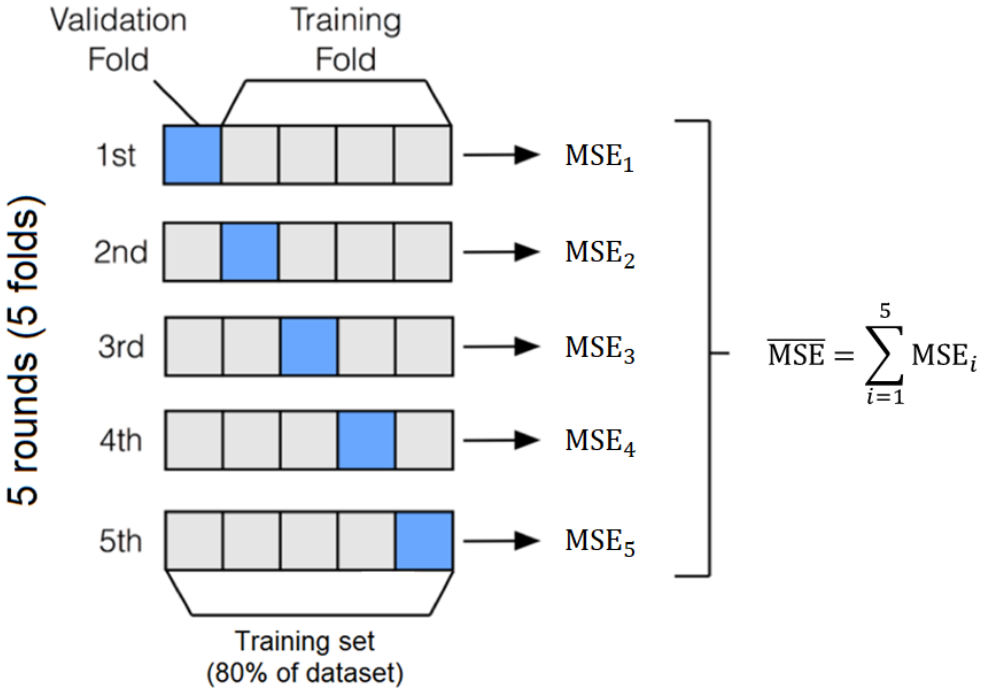


Figure 3.22: 5-fold cross-validation procedure adopted in the experiments of this thesis (see sec. 6.3).

Chapter 4

EV model and synthetic dataset

In order to train a machine learning regression model for estimating the SOH of an EV battery pack, it is critical to have a relatively large dataset of EV monitoring data to use for training. However, as of today, the availability of large-scale, freely accessible datasets of real EV monitoring data is very limited [140].

To overcome this issue, we may generate the required dataset synthetically.

The first step is to develop a mathematical model of an electric vehicle (sec. 4.1). The model must fulfil the following requirements:

- it should capture and mimic with high accuracy the mechanical dynamics of a real EV in driving conditions (speed, motor torque, braking force, etc.), as well as the electro-thermal dynamics of the vehicle's battery pack (current, voltage, SOC, temperature, etc.).
- it should be flexible enough to mimic a wide variety of real EVs just by changing its main mechanical/electrical parameters (so as to match those of the desired EV model).
- the user can simulate the model upon specifying a driving cycle, external temperature, initial battery temperature, initial battery SOC, battery age (SOH).
- the computational cost of performing a simulation should be reasonably low.

The model can be used after its definition (sec. 4.1) and parametrization (sec. 4.2) to simulate an EV in different driving conditions and then collect the physical signals generated by multiple simulations to build a synthetic dataset of EV monitoring data (sec. 4.4).

4.1 EV Simulink model

A simple electric vehicle model fulfilling all the requirements has been built through MATLAB®and Simulink®programming environments. Similarly to a real vehicle, it is composed of many mutually dependent subsystems (such as the electric motor, battery pack, wheels, brakes, and so on), connected to each other through signals that are calculated and updated at each time step of the simulation. A visual representation of the EV model is shown in fig. A.1 in appendix A.

This design is heavily based on an existing model found on MATLAB File Exchange [79], but many components have been modified and improved to better suit our experimental needs, most notably the battery pack and the braking subsystem. A brief technical description of each component of the model is given in this section. Variable names which are written with the `typewriter` font are user-defined parameters of the vehicle.

4.1.1 Driver

This subsystem implements a discrete-time proportional-integral (PI) controller to mimic a human driver for the vehicle. At each time step the controller tracks the input reference speed (e.g. driving cycle speed signal) and the current simulated vehicle speed, and tries to match them by acting on the brake and accelerator pedals. Brake pedal position (BPP) and accelerator pedal position (APP) take values in the range [0,1], where 0 stands for "not pressed" and 1 for "fully pressed".

4.1.2 Motor

The electric traction motor is used for propelling the EV. It is powered by electricity (supplied by the battery pack) and generates the power to rotate the wheels of the vehicle. Many types of electric motors exist; the main distinction is between DC and AC motors, depending on the type of electric flow they require to operate. Nowadays, most EVs feature an AC motor [141], either asynchronous (high power output) or synchronous (high torque output). Such EVs feature a DC/AC inverter to convert the DC supplied by the battery pack to the AC required to operate the AC motor (fig. 4.1).

Luckily, there exist a very simple characterization of an electric motor which is agnostic to its type and which is therefore used in our EV model [143]. Every electric motor can be characterized by two modes:

1. Constant-torque mode: the motor can output a constant rated rotor torque, $T_{r(rated)}$, and the rotor power P_r increases linearly with rotor/vehicle speed; achieved at low speeds.

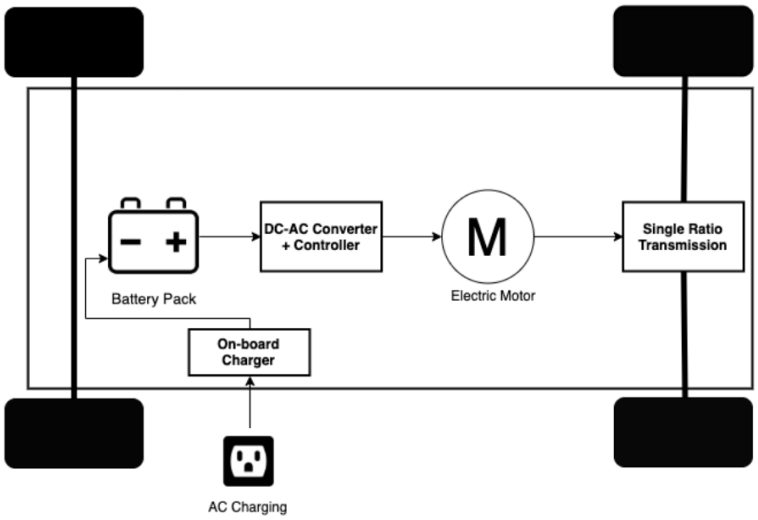


Figure 4.1: Typical electrical powertrain of an EV equipped with an AC motor. Adapted from [142]

2. Constant-power mode: the motor can output a constant rated rotor power, $P_{r(rated)}$, and the rotor torque T_r decreases inversely with rotor/vehicle speed; achieved at high speeds.

This characterization is schematized in fig. 4.2.

Moreover, there is a mathematical relation between power and torque

$$P_r = T_r \omega_r = T_r \frac{v n_g}{r} \tag{4.1}$$

where ω_r is the rotor speed [rad/s], v is the vehicle speed [m/s], n_g is the vehicle total drive ratio, r is the radius of the wheels [m]. This means that we can characterize an electric motor just by specifying its torque-speed envelope.

On a final note, electric motors cannot convert 100% of the electric power supplied by the battery (P_{batt}) to mechanical power, due to electrical and mechanical losses (P_{losses}) intrinsic to how the engine is designed. Losses are accounted for by introducing an efficiency term η_{motor}, so that the following equation holds:

$$P_r = P_{batt} - P_{losses} = \eta_{motor} P_{batt} \tag{4.2}$$

We implement an electric motor in our EV model through the Mapped Motor block in Simulink [144]. It is a mathematical model of an electric motor operated in torque-control mode. It can be parametrized by the user specifying a torque-speed envelope and other efficiency-related parameters:

- `motorSpdRPM`: a vector of N rotor speed values (in RPM), at which the value of maximum rotor torque is known

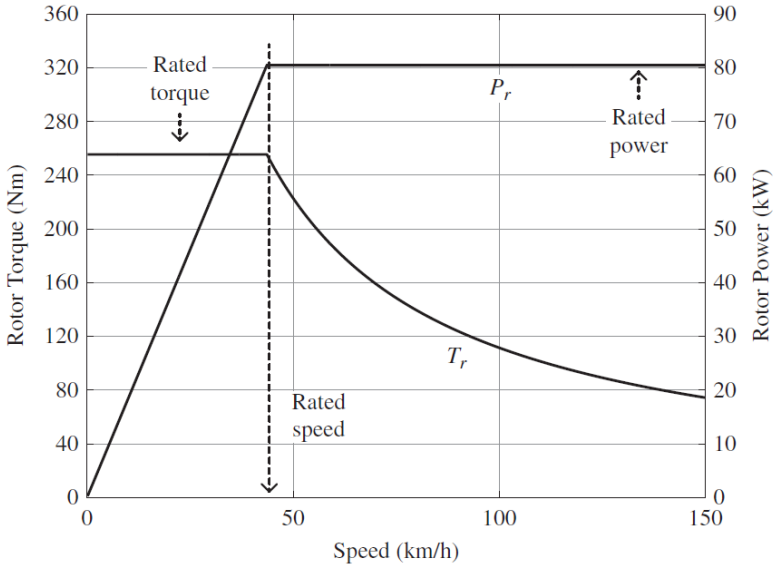


Figure 4.2: Torque-speed and power-speed characterization curves for the electric motor of a 2015 Nissan Leaf. Taken from [143]

.

- `motorMaxTrq`: a vector of N maximum rotor torque values [N·m] which can be generated by the motor at a certain speed.
- `motorEff`: motor efficiency η_{motor} at a rotor speed of `motorEffSpd` and a rotor torque of `motorEffTrq`
- `motorEffSpd`: rotor speed at which efficiency is measured [RPM]
- `motorEffTrq`: rotor torque at which efficiency is measured [N·m]
- `motorPIron`: iron losses at the speed and torque at which efficiency is defined [W]
- `motorPBase`: fixed losses independent from torque and speed [W]

When the driver presses the accelerator pedal (i.e. APP > 0), the model controls the mapped motor block with a rotor torque request of APP $\cdot\, T_{r(max)}(v)$. The block reads the current battery pack voltage V_{batt}, current rotor speed ω_r and the torque reference demand and computes the new rotor torque and the current $I_{batt} = P_{batt}/V_{batt}$ to be drawn from the battery to supply the needed power P_r.

4.1.3 Braking system

The braking system of an EV is based on two different contributes: friction braking and regenerative braking [143]. The BPP value is interpreted as the fraction of the maximum total braking torque (`brqTrqMax`) that can be exerted on each wheel by the combination of friction and regenerative braking. The resulting braking torque "requested" by the driver is then distributed between the two braking subsystems, in such a way that the regenerative braking is used with priority and whenever possible, and friction braking only makes up for the remaining torque which cannot be supplied by regenerative braking. In formula:

$$T_{braking} = \text{BPP} \cdot \texttt{brqTrqMax} = T_{regen} + T_{disc}$$

The details about this distribution rule, known as brake blending [145], are discussed in the following paragraphs.

Regenerative braking Regenerative braking is a braking technology which achieves the aim of slowing down a vehicle while recharging its battery pack at the same time. When the regenerative braking system is activated, the traction motor can develop a negative torque, up to the rated value in the forward direction. This torque reverses the usual flow of power such that the kinetic energy of the vehicle is converted to negative mechanical power on the driveshaft, and then converted to electrical power by the system. The generated electrical power is used to recharge the battery pack. This mechanism is schematized in fig. 4.3.

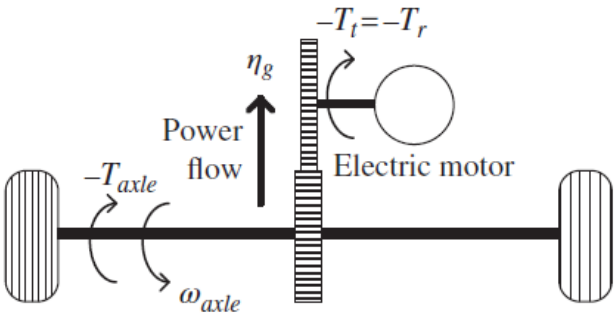


Figure 4.3: Regenerative braking scheme. Taken from [143]

.

Recent studies [146, 147] suggest that regenerative braking is not energy-efficient nor sufficiently effective to apply at low speeds. For this reason, our model implements a user-defined cutoff speed (`RegenBrkSpd_bpt`) above which the regenerative braking can be used. This is a common choice among different EV manufacturers. Fig. 4.4 shows how braking torque is supplied by the two braking contributes at different speeds.

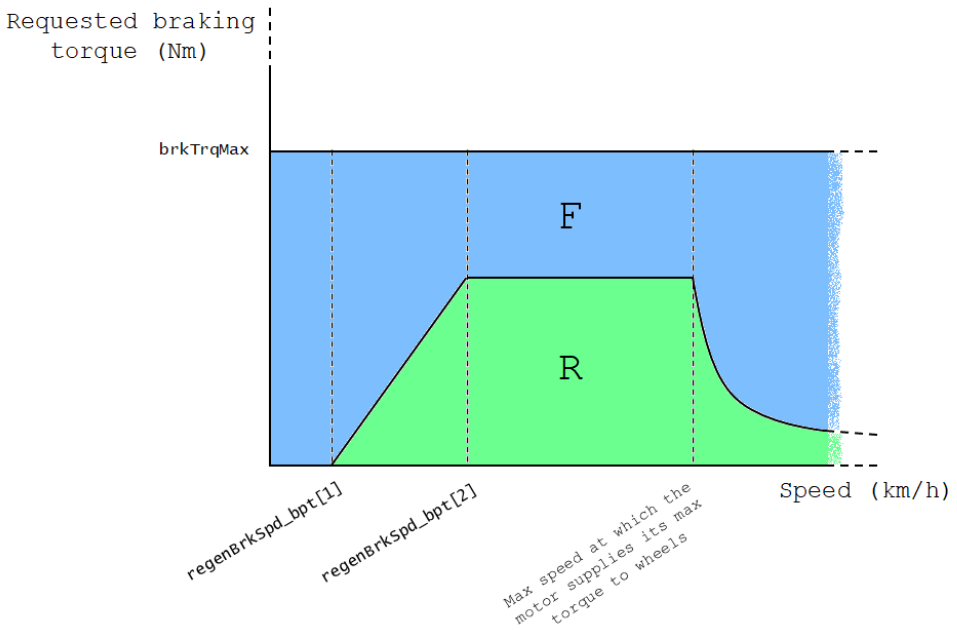


Figure 4.4: How the model distributes the requested braking torque between the regenerative (`R`) and friction (`F`) braking subsystems. Regenerative braking is activated when speed is above `RegenBrkSpd_bpt[1]`. The maximum regenerative braking torque which can be supplied increases linearly with speed until a speed of `RegenBrkSpd_bpt[2]` is reached. Above this speed, the regenerative brake is used to its full potential.

The formula which models the maximum regenerative braking torque on each wheel given the speed v of the vehicle is:

$$T_{regen} = \begin{cases} 0 & \text{if } v \leq v_1 \\ \dfrac{v - v_1}{v_2 - v_1} \cdot \dfrac{n_g T_{motor}(v)}{\eta_{regen}} & \text{if } v_1 < v \leq v_2 \\ \dfrac{n_g T_{motor}(v)}{\eta_{regen}} & \text{if } v > v_2 \end{cases} \tag{4.3}$$

where

- v_1 (`RegBrkSpd_bpt[1]`): cutoff speed above which the regenerative braking can be used [m/s]
- v_2 (`RegBrkSpd_bpt[2]`): speed beyond which the regenerative brake can be used to its full potential [m/s]

- n_g (`GR0*GRfinal`): overall gear ratio of the vehicle (see sec. 4.1.4)
- $T_{motor}(v)$ (previously written as $T_r(v)$ in sec. 4.1.2): maximum rotor torque at speed v [N·m]
- η_{regen} (`regenEff`): efficiency factor for the regenerative braking

The negative torque $-T_{regen}/n_g$ is then used to control the motor (sec. 4.1.2), which will generate a negative current which recharges the battery pack.

Friction braking Friction braking is the conventional braking mechanism of all wheeled vehicles. A friction brake performs its function by pressing a brake pad against a rotating wheel. As a result, a friction force opposing the direction of the wheel is developed. In the process, the kinetic energy of the wheel is dissipated as heat.

Disc brakes (fig. 4.5) are the most commonly used form of friction brake for motor vehicles [148]. They are implemented in our EV model on all four wheels.

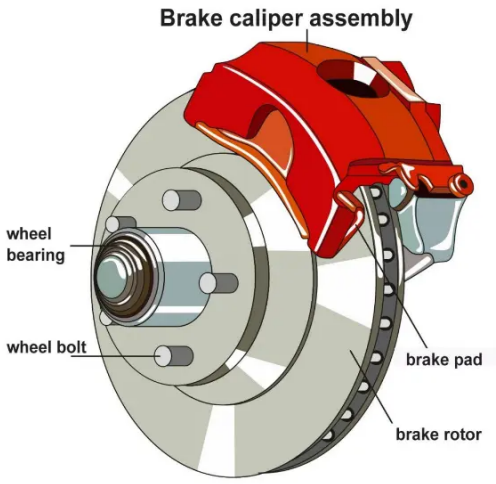


Figure 4.5: Disc brake scheme. Taken from [149]

The formula which models how much friction is generated on the wheel (measured as negative brake torque) is:

$$T_{disc} = \frac{\mu_{kinetic} P \pi d_{bore}^2 R_m N_{pads}}{4} \tag{4.4}$$

where

- $\mu_{kinetic}$ (`muKinetic`): coefficient of kinetic friction between disc pad and brake rotor

- P: total applied brake pressure [N/m^2]
- d_{bore} (`discActBore`): brake actuator bore diameter [m]
- R_m (`brakePadRadius`): mean radius of brake pad force application on the brake rotor [m]
- N_{pads} (`numPads`): number of brake pads in the disc brake assembly

When the vehicle is still or the wheels are locked up (wheel skidding), the maximum friction braking torque on the wheel is still given by eq. 4.4, but μ_{static} (`muStatic`) is used in place of $\mu_{kinetic}$.

As anticipated, friction braking is used only when regenerative braking torque doesn't fulfil the braking torque request entirely: $T_{disc} = T_{braking} - T_{regen}$. The value T_{disc} computed in this way is then used to compute the total pressure P to be applied by the disc brakes to the wheels (sec. 4.1.5), thanks to the inverse of eq. 4.4.

4.1.4 Drivetrain

Drivetrain (also known as transmission, or driveline) is the set of rotating shafts and gears that distributes the mechanical power generated by the electric motor (sec. 4.1.2) to the wheels (sec. 4.1.5).

While conventional internal combustion vehicles have a multi-gear gearbox with numerous ratios, nearly every electric car has a single-rate gearbox and no clutch, thus saving in cost, build complexity, weight and efficiency losses [150, 151]. This is because electric motors have a much larger RPM range than the typical internal combustion engine (they often peak at about 20,000 RPM), they stay efficient across a very broad RPM range and produce a decent amount of torque at low RPM. EV designers pick a fixed gear ratio that provides a good compromise between acceleration and top speed [150].

Gears are used across the drivetrain to reduce the angular velocity and increase torque before they are supplied to the wheels, while preserving power [152]. A typical EV drivetrain is schematized in fig. 4.6. It consists of the gear between the motor and the driveshaft, the driveshaft, the differential (which houses the gear between the driveshaft and the drive axles), the two drive axles and the two drive wheels. Depending on the EV design, the drive wheels can be the rear or front ones (see sec. 4.1.5); accordingly, the differential will be in the rear or front.

The two gears have gear ratio n_0 and n_{final}, and efficiency of η_0 and η_{final} (`GRfinal`) respectively; the driveshaft and drive axles have efficiency of η_{ds} and η_a respectively. It is convenient to express the overall gear ratio as

$$n_g = n_0 \cdot n_{final} \tag{4.5}$$

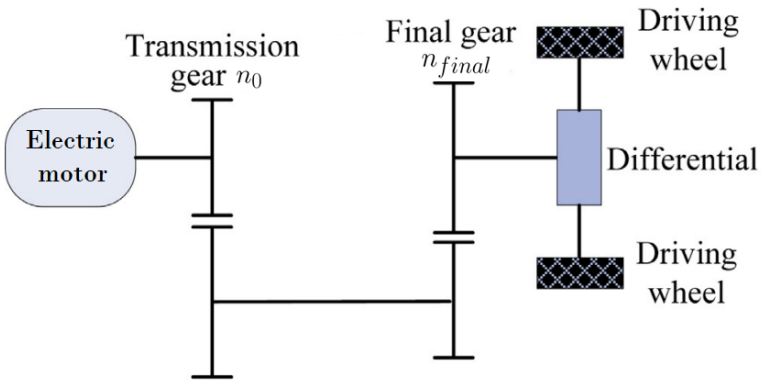


Figure 4.6: Schematic diagram of an EV drivetrain. Adapted from [153]

and the overall drivetrain efficiency as

$$\eta_g = \eta_0 \cdot \eta_{ds} \cdot \eta_a \cdot \eta_{final} \tag{4.6}$$

In the Simulink model, we simulate the components of the drivetrain with the following Simulink blocks:

- First gear: Gearbox [154]. Parameters: gear ratio n_0 (`GR0`), efficiency η_0 (`gearEff`).
- Driveshaft: Torsional Compliance [155]. Parameters: torsional stiffness k_{ds} [N·m/rad] (`driveshaft_k`), damping b_{ds} [N·m·s/rad] (`driveshaft_b`), damping cutoff frequency ω_{ds} [rad/s] (`driveshaft_wc`).
- Differential: Open Differential [156]. Parameters: gear ratio n_{final} (`GRfinal`), efficiency η_{final} (`diffEff`).
- Drive axles: Torsional Compliance [155]. Parameters: torsional stiffness k_a [N·m/rad] (`axle_k`), damping b_a [N·m·s/rad] (`axle_b`), damping cutoff frequency ω_a [rad/s] (`axle_wc`).

Shaft efficiency can be computed from torsional stiffness, torsional damping and damping cutoff frequency[1]. The formula for computing it can be inferred from Simulink documentation [155].

[1]Note however that most EV manufacturers do not share such technical, low-level information about the vehicle components. For this reason, the user may want to stick with the default parameters taken from the model in [79], which seem to work well in our experiments.

If the vehicle is driven on a straight line, the torque at the differential is equally split between the left and right drive wheel. In this case, the torque that the drivetrain transmits from the motor to each drive wheel is computed as:

$$T_{wheel} = \frac{n_g \cdot \eta_g \cdot T_{motor}}{2} \tag{4.7}$$

where

- n_g: overall gear ratio
- η_g: total drivetrain efficiency
- T_{motor}: rotor torque [N·m]

4.1.5 Wheels

Wheels are modeled using the "Longitudinal wheel with disc brake" Simulink block [157]. The total longitudinal traction (or braking) force F_x that the two drive wheels exert on the vehicle is given by Pacejka's Magic Formula [158]:

$$F_x = F_z D \sin(C \arctan(B\kappa - E(B\kappa - \arctan(B\kappa)))) \tag{4.8}$$

where

- F_z: vertical load on the tyre [N] (see sec. 4.1.6)
- κ: longitudinal wheel slip[2]
- B, C, D, E: stiffness, shape, peak and curvature factor

The four dimensionless coefficients are based on empirical data. We will assume that our EV model drives on dry tarmac, for which $B = 10$, $C = 1.9$, $D = 1$, $E = 0.97$ [157]. The block internally computes κ knowing the wheel radius under load r_w (`wheelRadius`), the vehicle speed v and the torque that the drivetrain transfers to the wheel T_{wheel} (sec. 4.1.4).

[2]Longitudinal wheel slip is defined as

$$\kappa = -\frac{v - r_w\omega_w}{v}$$

where r_w and ω_w are respectively the wheel radius and wheel angular velocity, and v is the longitudinal vehicle speed. When $\kappa = 0$, the wheel is rolling without slipping; $\kappa > 0$ means the wheel is slipping; $\kappa < 0$ means the wheel is skidding.

This block also models the disc brake that operates on the wheel (sec. 4.1.3). When the disc brake is operated, T_{wheel} is negative, therefore the computed F_x is too (i.e. F_x becomes a braking force).

Actually, tires are made of viscoelastic material, so they are subject to hysteresis energy loss, a phenomenon which results in rolling resistance [159]. However, rolling resistance is not implemented in our model, as it would increase the complexity of the model and slow down the simulation. Nonetheless, one can take rolling resistance into account by using Pacejka's magic formula for rolling resistance (eq. 4.E70 of [159]).

4.1.6 Vehicle body

The aerodynamic model of the vehicle body is implemented with the Vehicle Body 1DOF Longitudinal Simulink block [160]. It implements a one degree-of-freedom (1DOF) rigid vehicle body with constant mass undergoing longitudinal motion.

Assuming that the vehicle is driving on a flat road (angle of road grade $\gamma = 0$) with no wind, and that the two drive wheels are the front ones, this block internally solves the system of equations 4.9 associated to the vehicle's free body diagram (fig. 4.7):

$$\begin{cases} F_x = m\ddot{x} \\ F_x = F_{xF} + F_{xR} - F_{d,x} \\ F_{d,x} = \dfrac{1}{2T_{air}R_{air}} C_d A_f P_{air} \dot{x}^2 \\ F_{zF} = \dfrac{bmg - h(F_{xF} + F_{xR})}{2(a+b)} \\ F_{zR} = \dfrac{amg + h(F_{xF} + F_{xR})}{2(a+b)} \end{cases} \tag{4.9}$$

where

- F_x: effective longitudinal force [N]
- F_{xF}, F_{xR}: total longitudinal force exerted by the two front (F) and rear (R) wheels [N]
- $F_{d,x}$: longitudinal aerodynamic drag force [N]
- F_{zF}, F_{zR}: normal load force on each front (F) and rear (R) wheel [N]
- a (`a`): horizontal distance from vehicle's center of gravity (CG) to rear axle [m]
- b (`b`): horizontal distance from vehicle's CG to front axle [m]

- h (`h`): vehicle's CG height above axles [m]
- m (`vehicleMass`): vehicle loaded mass [kg]
- $\ddot{x}$: vehicle longitudinal acceleration [m/s^2]
- A_f (`frontalArea`): vehicle's frontal area [m^2]
- C_d (`dragCoeff`): frontal air drag coefficient
- $T_{air}, P_{air}, R_{air}$: air temperature [K], pressure [Pa] and specific gas constant [J·kg^{-1}·K^{-1}]

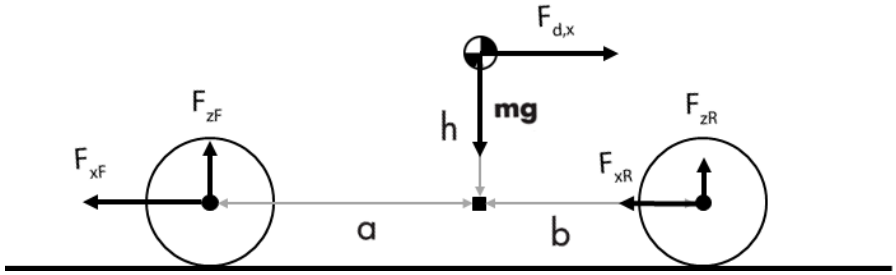


Figure 4.7: Free body diagram of the vehicle model. Adapted from [160]

Given F_{xF} and F_{xR} (sec. 4.1.5), the block outputs the vehicle speed v and the normal load forces F_{zF} and F_{zR}.

4.1.7 Battery pack

The EV's battery pack is the most critical component of the model, since our aim is to predict its SOH. As discussed in sec. 1.2, an EV battery pack is actually a set of battery cells connected together in series and parallel; however, modeling an EV battery pack at the cell level would greatly increase the complexity of the simulation, and would be very hard to parametrize in a realistic way without extensive knowledge of the specific topology of the pack, its positioning inside the car and the specific features of the BMS in use. Therefore, we model our battery pack as if it consisted of only one high-voltage cell. To do so, our model uses the Generic Battery Model (GBM) block from the Simscape electrical Simulink library [161]. This block implements a generic dynamic equivalent circuit model (ECM) of a rechargeable battery, summarized in fig. 4.8.

The GBM block interacts with the rest of the EV model as described in sec. 4.1.2: the motor reads the terminal voltage of the battery pack V_{batt} and requests a current I_{batt} based on the power demand P_{batt}. Then, a controlled current source imposes a current I_{batt} to the battery pack, causing the terminal voltage to change

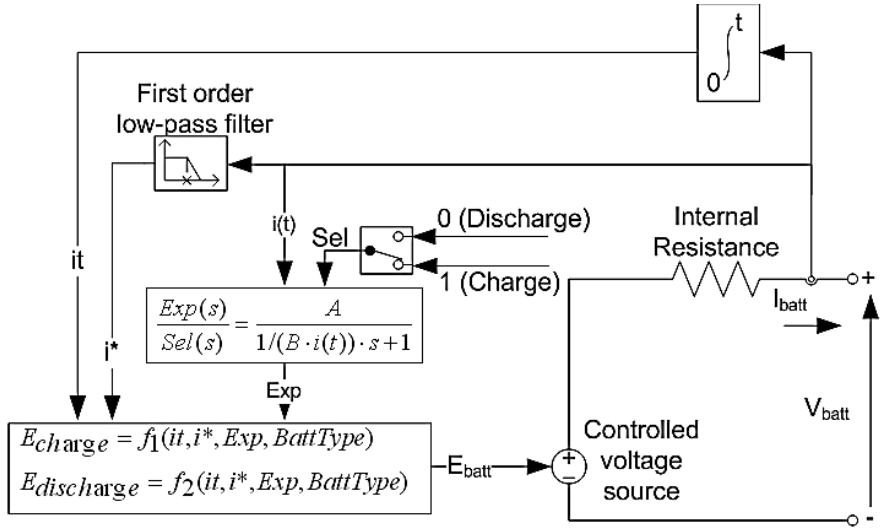


Figure 4.8: Internal ECM model of the Generic Battery Model Simulink block. Taken from [161]

accordingly to the charge/discharge characteristics specified for the battery pack (see "Discharge characteristics" paragraph). Moreover, the battery pack thermally interacts with the environment by reading the external temperature T_a, which can be set by the user. The equations which govern the electrical and thermal dynamics of the battery can be found in [161].

The GBM block parameters must be set according to those characterizing the battery pack of the specific EV we want to model. The parameters required to initialize the block are briefly discussed in the following paragraphs.

Discharge characteristics

In fig. 4.9 a typical discharge curve of a battery cell is plotted. Three distinct sections can be observed:

1. *Exponential zone*: exponential voltage drop from a fully charged state (V_{full},0) to a point (V_{exp}, Q_{exp});

2. *Nominal zone*: characterized by an almost constant voltage; from (V_{exp}, Q_{exp}) to (V_{nom}, Q_{nom});

3. *Total discharge zone*: an extremely rapid voltage drop from (V_{nom}, Q_{nom}) to (V_{empty}, Q_{full}).

The discharge characteristics depend on a set of nominal conditions of operation defined by a nominal ambient temperature T_{nom} (`Tnom1`) and a nominal discharge

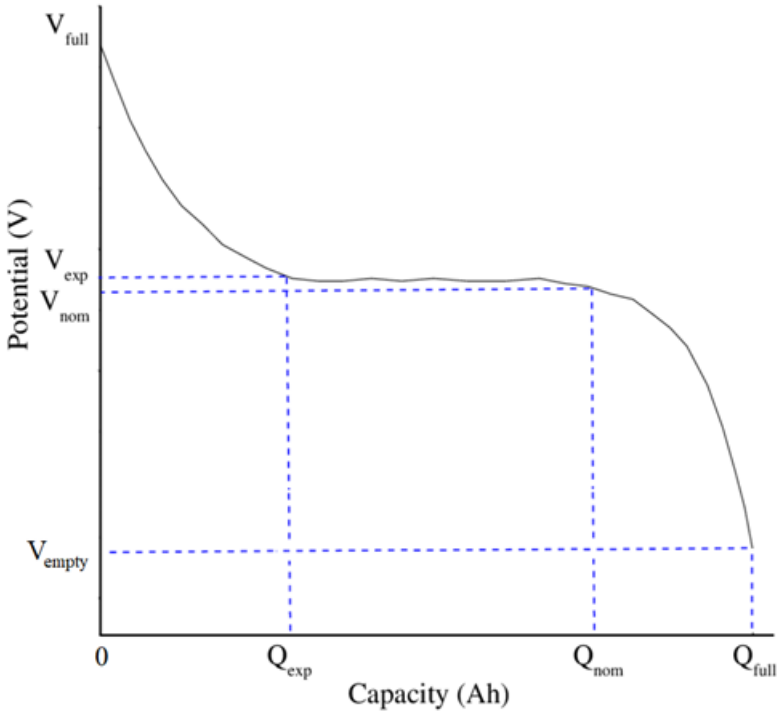


Figure 4.9: Typical discharge curve of a battery (at a certain C-rate, temperature, SOH), function of the discharged capacity Q. NB: the curve is not in scale, as the nominal zone is typically much wider compared to the exponential and total discharge zones.

current I_{nom} (`nomDischCurr`). To internally model the discharge curve in such operating conditions, the block requires the following parameters to be set:

- Q_{rated} (`ratedCap`): rated capacity, i.e. the minimum effective capacity of the battery [Ah]
- R_{int} (`intRes`): internal resistance when the battery is fresh [Ω]
- V_{full} (`fullychargV`): voltage at which the battery is fully charged [V]
- V_{exp} (`expV`): voltage at the end of the exponential zone [V]
- Q_{exp} (`expVCap`): discharged capacity at V_{exp} [Ah]
- V_{nom} (`nomV`): voltage at the end of the nominal zone [V]
- Q_{nom} (`nomVCap`): discharged capacity at V_{nom} [Ah]
- V_{empty} (`cutoffV`): voltage at which the battery is fully discharged [V]

- Q_{full} (`maxCap`): maximum theoretical capacity [Ah]

The user must also provide the SOC (%) of the battery at the beginning of the simulation (`initSOC`). Finally, self-discharge is modeled by adding a large resistor of 2000 Ω in parallel with the battery terminals.

Thermal model

The following parameters describe the discharge characteristics at a second operating condition (different from the nominal one), characterized by an external temperature $T_{nom,2}$ (`Tnom2`):

- $Q_{full,2}$ (`maxCap_Tnom2`): maximum theoretical capacity [Ah]
- $V_{full,2}$ (`initDischV_Tnom2`): voltage at which the battery is fully charged [V]
- $V_{90\%}$ (`V90maxCap_Tnom2`): voltage when 90% of $Q_{full,2}$ has been discharged [V]
- $V_{exp,2}$ (`expV_Tnom2`): voltage at the end of the exponential zone [V]
- $Q_{exp,2}$ (`expVCap_Tnom2`): discharged capacity at $V_{exp,2}$ [Ah]

Moreover, battery-to-ambient thermal interaction is described by two parameters: thermal resistance (`thRes`) and thermal time constant (`thTimeConst`). The user must also provide the internal temperature [°C] of the battery at the beginning of the simulation (`initBattTemp`), and the external temperature T_a [°C], which can either be a constant or a time series of temperature measurements.

Aging model

Other than current and internal temperature, discharge characteristics are also affected by battery pack's SOH. The GBM block features a simple aging model of the battery, which is parametrized with the following parameters:

- Q_{EOL}: maximum theoretical capacity at end of life [Ah]; usually for EV applications, battery pack's EOL is set at 80% Q_{full} (i.e. $0.8 \times$ `maxCap`) [162].
- R_{EOL}: internal resistance at end of life [Ω]; set to $2 \times$ `intRes` by default [163].
- $I_c, I_{c,max}$ (`Ic, Icmax`): nominal and maximum charge current [A]
- $I_d, I_{d,max}$ (`Id, Idmax`): nominal and maximum discharge current [A]
- `cycLife_100DOD_Ic_Id_Ta1`: EFC at 100% DOD, at I_c and I_d, to reach EOL
- `cycLife_25DOD_Ic_Id_Ta1`: EFC at 25% DOD, at I_c and I_d, to reach EOL

- `cycLife_100DOD_Ic_Idmax_Ta1`: EFC at 100% DOD, at I_c and $I_{d,max}$, to reach EOL

- `cycLife_100DOD_Icmax_Id_Ta1`: EFC at 100% DOD, at $I_{c,max}$ and I_d, to reach EOL

- $T_{amb,2}$ (`Ta2`): a second ambient temperature at which the aging model is parametrized [°C]

- `cycLife_100DOD_Ic_Id_Ta2`: EFC at 100% DOD, at I_c and I_d and temperature $T_{amb,2}$, to reach EOL

The user must also specify the battery age (as number of EFC) which characterizes the battery pack throughout the simulation (`initBattAge`).

The mathematical relationship between the battery age expressed in number of EFC and the percentage of fresh maximum capacity (which is our reference definition of SOH, eq. 1.2) is not linear in general [164]. However, a simple simulation of a battery characterization test may be performed to unambiguously associate an EFC level to the correspondent SOH value (sec. 4.3).

4.1.8 Known limitations

Our proposed EV model suffers from several limitations, mainly due to its low-complexity design. Known limitations are:

- Energy losses due to auxiliary devices (such as air-conditioner, power steering, car lights and so on), which are usually powered by a 12V auxiliary battery [165], are not modeled.

- Rolling resistance is not modeled, as explained in sec. 4.1.5.

- It is often difficult to retrieve technical specifications of a specific EV model from the Internet, as this information is not always publicly shared by EV manufacturers. Required specifications which could not be found have been estimated indirectly by exploiting real EV field data or otherwise made up by choosing plausible values.

- Many EVs can operate in different driving modes, with the aim of adapting the energy consumption to a desired driving profile, ultimately optimizing the vehicle's driving range [166]. Due to the huge variety of driving modes implemented by different EV manufacturers, the model is designed with a default driving mode.

- A fair validation of the Simulink EV model by means of monitoring data acquired from real EVs is impossible, because an EV's BMS monitors only a fraction of the many different variables and factors which characterize driving. For instance: road type, slope, road curves, wind direction and speed are usually not recorded. For this reason, our model is designed to mimic the dynamics of a car moving in the forward direction on a flat dry tarmac road.
- The ECM model which characterizes Simulink generic battery block (sec. 4.1.7) has several limitations and assumptions in its design, as stated in its documentation [161]. For instance:
  - internal resistance is assumed to be constant during charging and discharging and does not vary with the amplitude of the current;
  - the capacity of the battery does not change with the amplitude of the current (there is no Peukert effect);
  - the battery has no memory effect.
- Simulink generic battery block implements a very low complexity thermal model, which cannot precisely capture heat transfer phenomena at the cell and module level and between the heat-producing components of the car (e.g. the motor) and the battery itself. Moreover, the thermal behaviour of the block seems to be independent of the SOH of the battery (as opposed to real batteries, as discussed in sec. 1.2).
- The user cannot specify battery's SOH directly, but rather the battery age expressed in number of EFC; this limitation is overcome by mapping EFC levels to SOH values (see sec. 4.3).

4.2 EV model parametrization

In order to generate a synthetic dataset of monitoring data, the EV Simulink model must be parametrized according to the technical specifications of a specific EV model. In this thesis, we focus on two EV models by Volkswagen: the e-up! (18.7 kWh) and the e-Golf (35.8 kWh). This choice is due to the availability of real driving session monitoring data for these two models (sec. 5), acquired from a private EV fleet management company (sec. 1.3).

The required technical specifications for the VW e-up! and VW e-Golf are reported in table 4.1. As mentioned in sec. 4.1.8, retrieving the actual technical specifications for an EV model is not an easy task in general. For this reason, whenever any specification was not directly found on the Internet or in the literature it has been either:

- estimated with a parameter estimation procedure (see sec. 4.2.1); marked with "**est**" in table 4.1
- computed indirectly from pertaining sources, assumed from sources pertaining to similar models, or otherwise left as Simulink's default value; marked with "*****" in table 4.1

Table 4.1: Technical specifications for the VW e-up! (18.7 kWh) and the VW e-Golf (35.8 kWh). Parameters marked with 'est' are estimated through parameter estimation (sec. 4.2.1), and those marked with '*' are computed indirectly from pertaining sources or left as default.

	Parameter name	Workspace variable	e-up!	e-Golf	Unit	Source (e-up!)	Source (e-Golf)
Motor	Rotor speed breakpoints	`motorSpdRPM`	(see source)	(see source)	RPM	[167]	[168]
	Max. motor torque at rotor speed breakpoints	`motorMaxTrq`	(see source)	(see source)	N·m	[167]	[168]
	Motor efficiency	`motorEff`	0.95	0.95	–	[169, 170],*	[169, 170],*
	Rotor speed at which motor eff. is measured	`motorEffSpd`	6500	6500	RPM	[167, 170],*	[168, 170],*
	Rotor torque at which motor eff. is measured	`motorEffTrq`	89	146	N·m	[167, 170],*	[168, 170],*
	Iron losses	`motorPIron`	0	0	W	*	*
	Fixed losses	`motorPBase`	0	0	W	*	*
Braking system	Max. total braking torque	`brqTrqMax`	2000	2000	N·m	[171],*	[171],*
	Regen. braking cutoff speeds	`RegenBrkSpd_bpt`	[18, 32.4]	[18, 32.4]	km/h	[79],*	[79],*
	Regen. efficiency factor	`regenEff`	0.75	0.75	–	[79],*	[79],*
	Kinetic friction coeff.	`muKinetic`	0.35	0.35	–	[79],*	[79],*
	Static friction coeff.	`muKinetic`	0.9	0.9	–	[79],*	[79],*
	Brake actuator bore diameter	`discActBore`	0.05	0.05	m	[79],*	[79],*
	Brake pad radius	`brakePadRadius`	0.15	0.15	m	[79],*	[79],*
	N. of brake pads	`numPads`	2	2	–	[79],*	[79],*
Drivetrain	First gear ratio	`GR0`	1.576	2.7	–	[172]	[173]
	First gear efficiency	`gearEff`	0.95	0.95	–	[174],*	[174],*
	Driveshaft tors. stiffness	`driveshaft_k`	2500	2500	N·m/rad	[79],*	[79],*
	Driveshaft tors. damping	`driveshaft_b`	10	10	N·m·s/rad	[79],*	[79],*
	Driveshaft damping cutoff frequency	`driveshaft_wc`	500	500	rad/s	[79],*	[79],*
	Final gear ratio	`GRfinal`	5.176	3.61	–	[172]	[173]
	Differential efficiency	`diffEff`	0.95	0.95	–	[174],*	[174],*
	Axle tors. stiffness	`axle_k`	5000	5000	N·m/rad	[79],*	[79],*
	Axle tors. damping	`axle_b`	10	10	N·m·s/rad	[79],*	[79],*
	Axle damping cutoff frequency	`axle_wc`	400	400	rad/s	[79],*	[79],*
Body and wheels	Wheel radius	`wheelRadius`	0.293	0.316	m	[172, 175]	[173, 175]
	CG to front axle	`a`	1.04	1.21	m	[176–178]	[173, 178, 179]
	CG to rear axle	`b`	1.38	1.42	m	[176–178]	[173, 178, 179]
	CG height above axles	`h`	0.35	0.35	m	[79],*	[79],*
	Vehicle mass	`vehicleMass`	1375	1690	kg	[180]	[173]
	Frontal area	`frontalArea`	2.07	2.21	m^2	[176],*	[181],*
	Air drag coefficient	`dragCoeff`	0.32	0.27	–	[180]	[173]

continued on next page

Table 4.1 (continued)

continued from previous page

	Parameter name	Workspace variable	e-up!	e-Golf	Unit	Source (e-up!)	Source (e-Golf)
Battery pack - discharge	Nom. amb. temperature	`Tnom1`	20	20	°C	[161]	[161]
	Nom. disch. current	`nomDischCurr`	24.8	54.2	A	est	est
	Rated capacity	`ratedCap`	50.3	116.1	Ah	[180], est	[173], est
	Internal resistance	`intRes`	0.08	0.075	Ω	[182], est	[182], est
	Voltage at 100% SOC	`fullychargV`	429.9	386.6	V	[183], est	[182], est
	Voltage at the end of the exp. zone (V_{exp})	`expV`	407.6	351.9	V	est	est
	Capacity disch. at V_{exp}	`expVCap`	5.0	10.6	Ah	est	est
	Voltage at the end of the nom. zone (V_{nom})	`nomV`	345.6	304.3	V	[180], est	[173], est
	Capacity disch. at V_{nom}	`expVCap`	44.0	95.8	Ah	est	est
	Voltage at 0% SOC	`cutoffV`	297.2	288.7	V	[183], est	[182], est
	Max. theor. capacity	`maxCap`	49.9	107.7	Ah	est	est
Battery pack - thermal	2nd nom. amb. temperature ($T_{nom,2}$)	`Tnom2`	-30	-30	°C	[161]	[161]
	Max. theor. capacity at $T_{nom,2}$	`maxCap_Tnom2`	38.3	87.9	Ah	est	est
	Voltage at 100% SOC at $T_{nom,2}$	`initDischV_Tnom2`	397.2	350.2	V	est	est
	Voltage when 90% disch. cap., at $T_{nom,2}$	`V90maxCap_Tnom2`	304.5	289.8	V	est	est
	Voltage at the end of the exp. zone, at $T_{nom,2}$ ($V_{exp,2}$)	`expV_Tnom2`	392.2	326.9	V	est	est
	Cap. discharged at $V_{exp,2}$, at $T_{nom,2}$	`expVCap_Tnom2`	2.0	9.9	Ah	est	est
	Thermal resistance	`thRes`	0.065	0.013	°C/W	est	est
	Thermal time constant	`thTimeConst`	30000	27000	s	est	est
Battery pack - aging	Nom. charge current	`Ic`	7.5	7.5	A	*	*
	Max. charge current	`Icmax`	9.58	9.58	A	*	*
	Nom. discharge current	`Id`	17.83	17.83	A	*	*
	Max. discharge current	`Idmax`	75.93	75.93	A	*	*
	EFC @ 100% DOD, I_c, I_d to EOL	`cycLife_100DOD_Ic_Id_Ta1`	3000	3000	–	[184],*	[184],*
	EFC @ 25% DOD, I_c, I_d to EOL	`cycLife_25DOD_Ic_Id_Ta1`	20000	20000	–	*	*
	EFC @ 100% DOD, I_c, $I_{d,max}$ to EOL	`cycLife_100DOD_Ic_Idmax_Ta1`	2000	2000	–	[184],*	[184],*
	EFC @ 100% DOD, $I_{c,max}$, I_d to EOL	`cycLife_25DOD_Icmax_Id_Ta1`	2500	2500	–	[184],*	[184],*
	2nd amb. temperature ($T_{amb,2}$)	`Ta2`	45	45	°C	[161]	[161]
	EFC @ 100% DOD, I_c, I_d, $T_{amb,2}$ to EOL	`cycLife_100DOD_Ic_Id_Ta2`	2500	2500	–	*	*

4.2.1 Parameter estimation

Most parameters specific to the battery pack have been estimated with Simulink Parameter Estimator app [185]. Parameter estimation can be formulated as a constrained or unconstrained optimization problem (sec. 3.1).

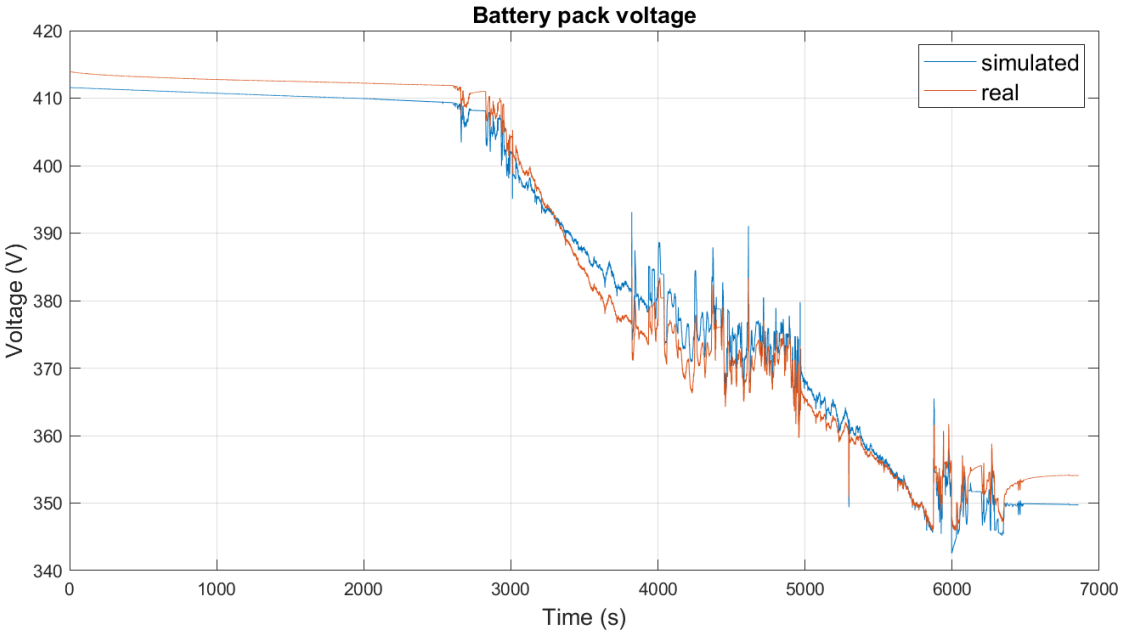
The parameters estimated with this procedure are those marked with "est" in table 4.1. We set up a parameter estimation experiment by picking a real driving session from each of the two acquired real datasets to use as experimental input and output data. The battery pack current signals from the two driving sessions have been extracted and set as the input signals for the simulation. We have chosen two battery test driving sessions, as they produce signals spanning from a full charge to an almost full discharge of the battery pack: these wide operating conditions ensure that the found parameters are well-fitted for almost any section of the discharge curve of the battery pack. The initial SOC, initial battery temperature and battery SOH of the simulated battery pack are set accordingly to the selected driving session. The battery pack voltages have been chosen as the measured output signals; parameters have been tuned so that the simulated voltage signal matches the experimental one as much as possible (i.e. the SSE over the measured signal time base[3] is minimized). Nelder-Mead method (sec. 3.1.3) has been chosen to solve the optimization problem.

The value for each parameter to be estimated has been initialized randomly in a plausible value interval for the battery pack under consideration. The chosen value intervals are compatible with the electrical parameters reported in tab. 4.1, and are reported in tab. 4.2 for each parameter to be estimated. Nelder-Mead optimization method solves the parameter estimation problem in an unconstrained way.

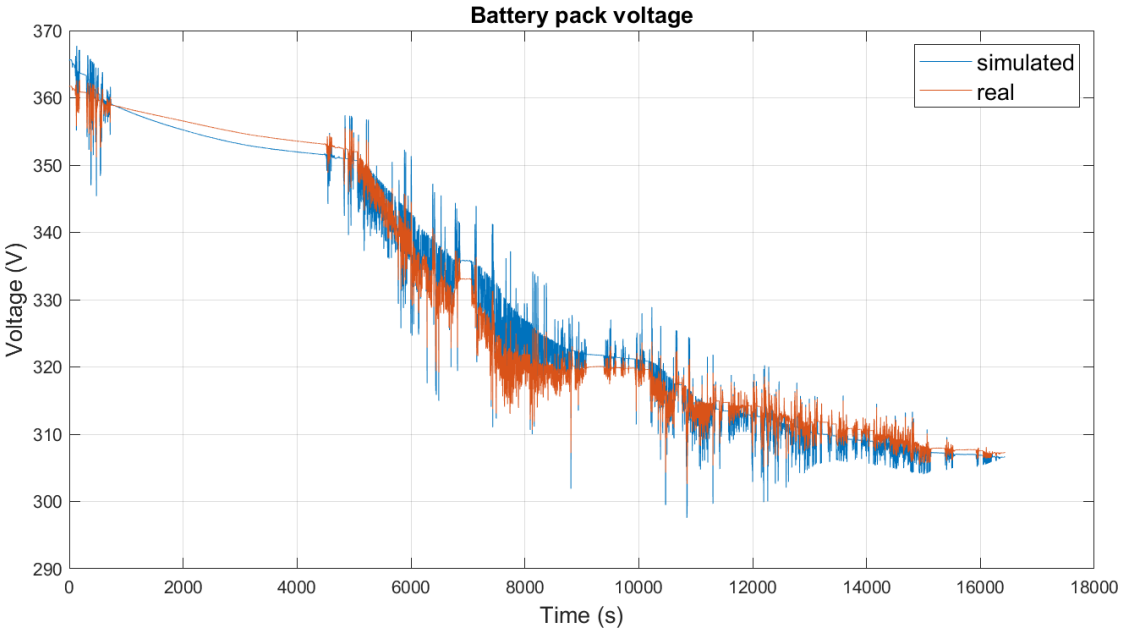
Fig. 4.10 and 4.11 show a comparison of the simulated voltage and SOC signals versus the experimental (real) ones. In both cases, the simulated signals tend to mimic the experimental ones with a very low error (as high as 10 V for the voltage signals).

As for the thermal model, the thermal resistance and thermal time constant have been estimated by trial and error. Namely, they were initialized to different

[3]The measured signal time base consists of all the timestamps for which the measured signal is specified, whereas the simulated signal time base consists of all the timestamps for which the model is simulated. The two time bases are different in general. By default, Simulink Parameter Estimator computes the cost function only in the timestamps of the measured signal time base. Our Simulink model uses a variable-step discrete solver, which adjusts the simulation step size during the simulation by reducing it to increase accuracy when model states are changing rapidly and increasing it to avoid taking unnecessary steps when model states are changing slowly; therefore, it can always simulate the model in the specific timestamps of the measured signal time base, if informed to do so.

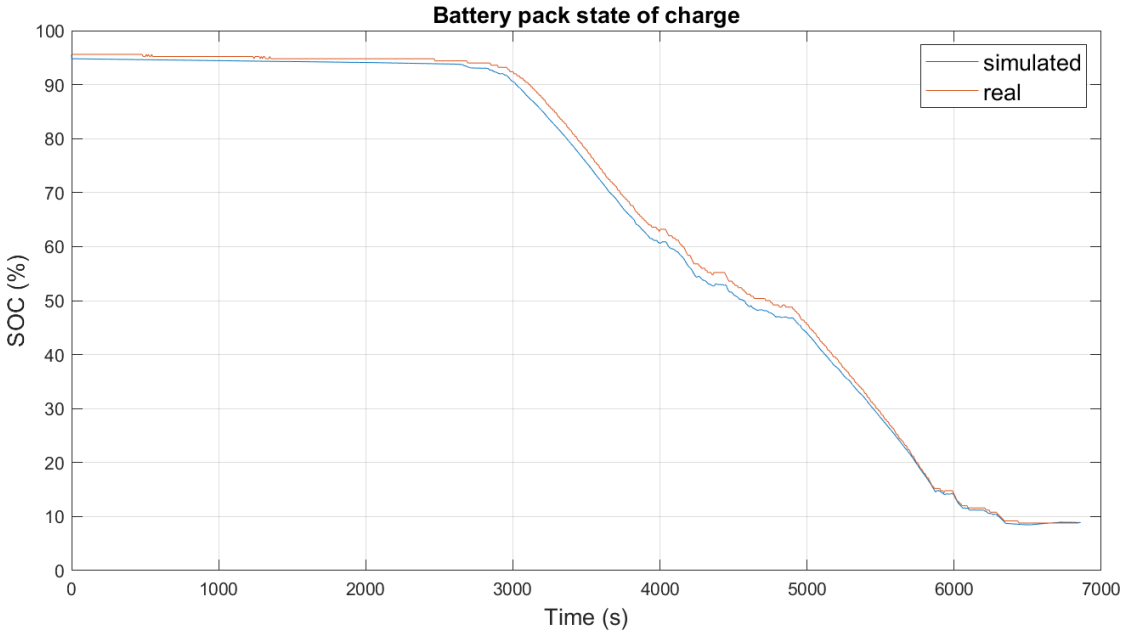


(a) e-up!

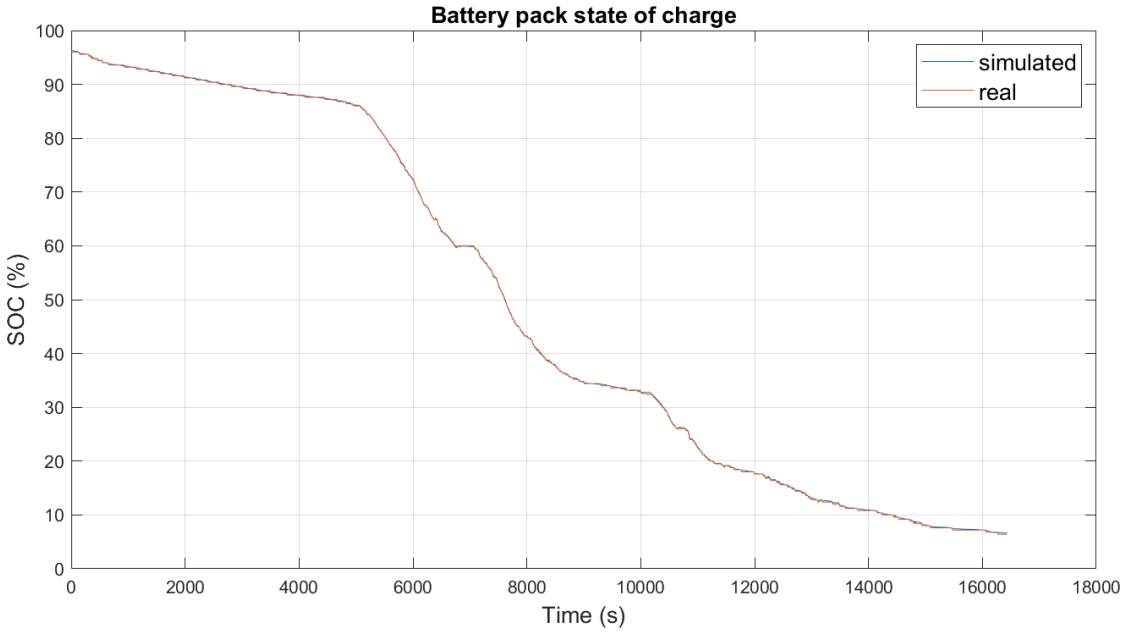


(b) e-Golf

Figure 4.10: Simulated and real voltage signal after parameter estimation



(a) e-up!



(b) e-Golf

Figure 4.11: Simulated and real SOC signal after parameter estimation

Parameter	Search interval (e-up!)	Search interval (e-Golf)	Unit
nomV	[330, 390]	[300, 360]	V
ratedCap	[45, 55]	[105, 115]	Ah
maxCap	[50, 60]	[108, 125]	Ah
cutoffV	[270, 310]	[250, 280]	V
fullychargV	[405, 435]	[345, 375]	V
nomDischCurr	[25, 30]	[50, 60]	A
intRes	[0.06, 0.1]	[0.06, 0.1]	Ω
nomVCap	[35, 48]	[75, 105]	Ah
expV	[390, 420]	[325, 360]	V
expVCap	[0, 10]	[0, 20]	Ah
maxCap_Tnom2	[30, 45]	[66, 100]	Ah
initDischV_Tnom2	[390, 415]	[325, 355]	V
V90maxCap_Tnom2	[290, 330]	[270, 300]	Ah
expV_Tnom2	[375, 405]	[305, 345]	V
expVCap_Tnom2	[0, 10]	[0, 20]	Ah

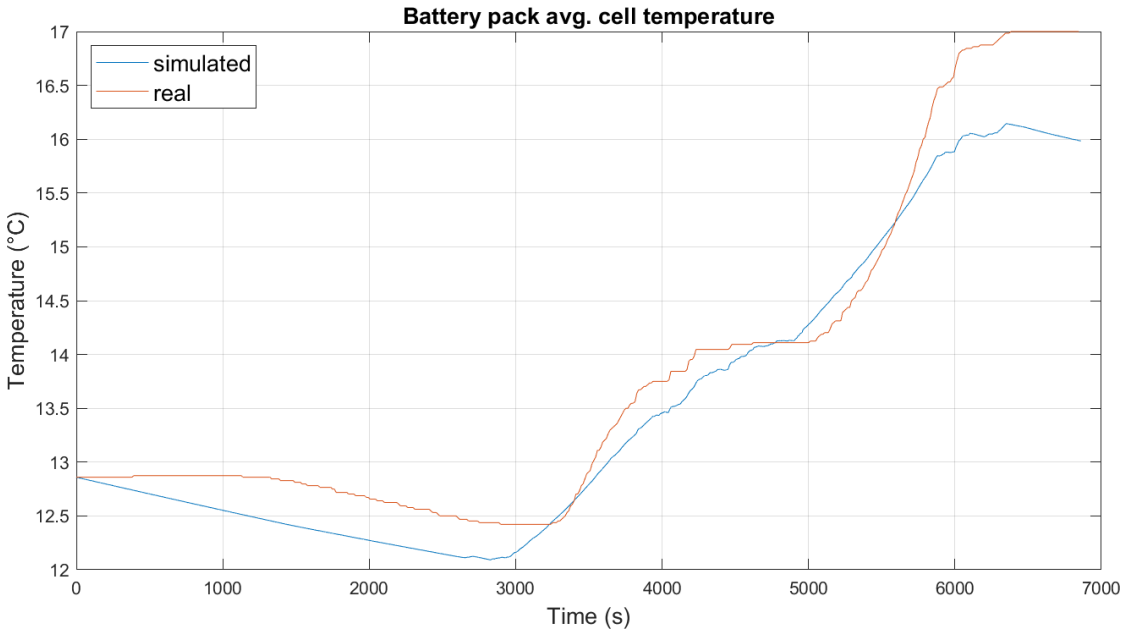
Table 4.2: Initialization intervals for each parameter to be tuned.

values for many simulations, and the values for which the deviation between the simulated and real battery temperature was low enough were kept. Fig. 4.12 shows a comparison of the simulated temperature signal versus the experimental (real) one. The deviation between the simulated and real signal is very high at times, but the general trend of the real signal is mimicked successfully. We claim that this behaviour is due to the very low-complexity thermal model that the Simulink battery block implements (as discussed in sec. 4.1.8).

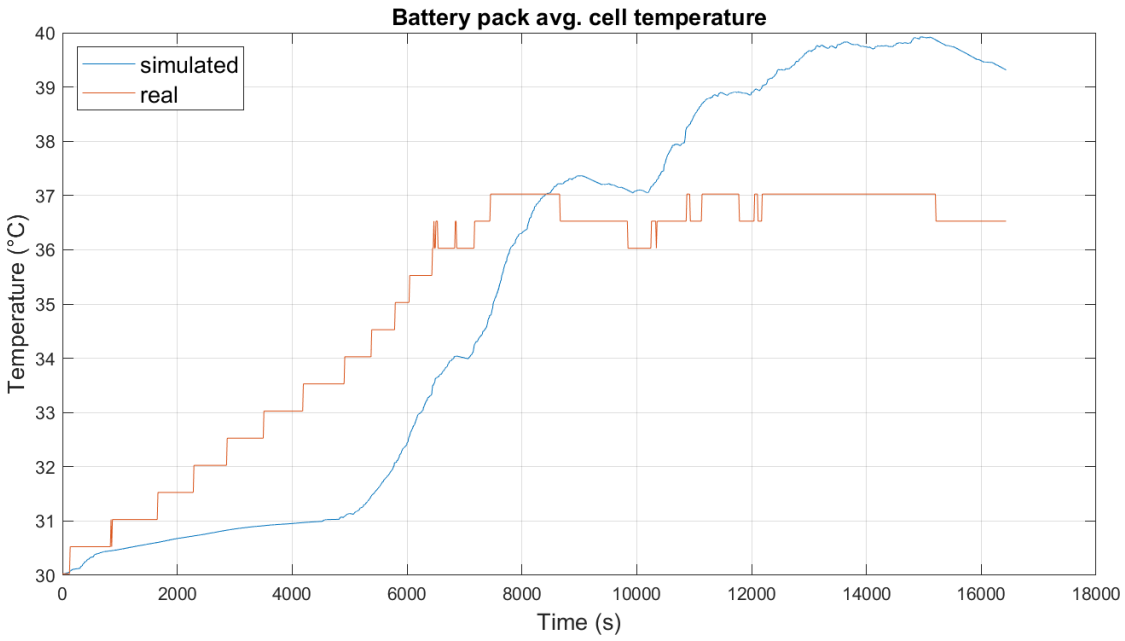
4.3 EFC to SOH conversion

As discussed in sec. 1.2.1, there is no general mathematical relationship between the SOH and the number of EFC. The age of the generic battery model in Simulink can only be set as number of EFC; however, we are interested in expressing the age of the battery pack in terms of SOH.

To overcome this issue, we can set up a simple Simulink simulation which consists in imposing a constant 1C discharge current to the battery block (with an initial



(a) e-up!



(b) e-Golf

Figure 4.12: Simulated vs real temperature signal after (manual) parameter estimation

SOC of 100%) and waiting for its complete discharge (SOC = 0%). Then, the actual capacity of the battery is computed by simply multiplying the magnitude of the imposed discharge current and the ending time of the simulation (in hours). This capacity estimation technique is known as Coulomb counting (sec. 2.1.1). Finally, the SOH is obtained by dividing the result of the multiplication by the maximum theoretical capacity of the battery under nominal operating conditions (`maxCap`).

Since the battery pack aging model is parametrized in the same way for the e-up! and the e-Golf Simulink models, the EFC-to-SOH mapping is unique for the two models. The experiment has been performed 31 times, for EFC values ranging from 0 to 3000 with a step of 100. The resulting mapping is reported graphically in fig. 4.13.

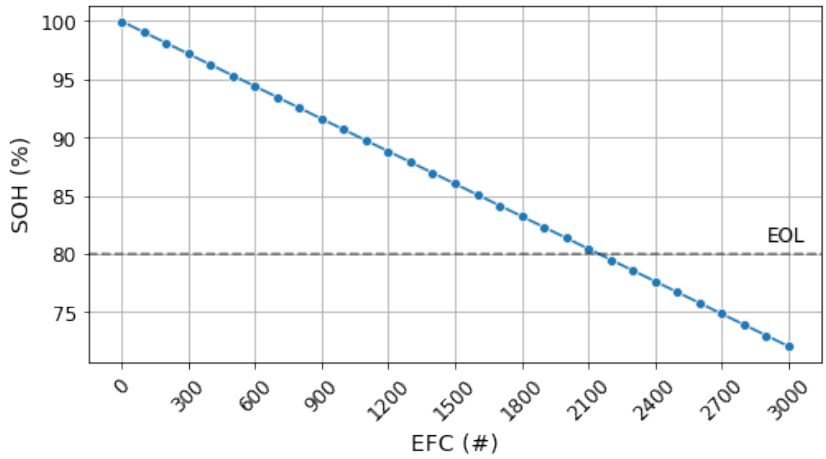


Figure 4.13: EFC to SOH conversion (linearly interpolated)

It appears that Simulink's generic battery model is characterized by a linear relationship between EFC and SOH (as opposed to a generic real battery).

4.4 Synthetic dataset generation

The parametrized EV model can now be used to generate a synthetic dataset of EV monitoring data. To simulate a single driving session, the user must specify a driving cycle, outside temperature [°C] (fixed or as a time series of measurements), initial average battery pack temperature [°C], initial battery pack SOC and battery pack SOH. The monitored signals are: speed [km/h], terminal voltage V [V], current I [A], SOC(%), average battery internal temperature T [°C]. Additionally, the user can specify a specific `sampling_rate` for the monitored signals (default is 0.2 s). Resampling through linear interpolation is performed at the end of each simulation

on all the acquired signals according to the specified sampling rate, in order to achieve a common time base for all the signals. The choice of the sampling rate is a trade-off between computation time and signal fidelity: a fast sampling rate may capture high frequency changes in the signals more accurately, at the cost of a higher simulation time.

Since it is infeasible to acquire data for every possible combination of SOH, driving cycle, initial internal and external temperature, we design the following synthetic dataset generation procedure, with the aim of acquiring a minimal but sufficiently representative set of monitoring data in a very wide range of driving conditions:

Algorithm 1 Simulink EV model synthetic dataset generation

```
for efc ∈ EFC_LEVELS do
    for dc ∈ DC_LIST do
        initSOC ← 𝒰(91,96)
        T_outside ← 𝒰(−5,30)
        initBattTemp ← 𝒰(T_outside, T_outside + 5)
        simulate dc in loop until SOC < 10
        get simulated signals (speed, V, I, SOC, T)
        resample signals every sampling_rate s
        store dataset on disk
    end for
end for
concatenate datasets
```

where "$\leftarrow \mathcal{U}(a,b)$" is a random extraction from the uniform distribution on $[a,b]$, `EFC_LEVELS` is a list of EFC values at which to set the battery block for the simulation and `DC_LIST` is a list of drive cycles that we wish to simulate.

To be representative, the synthetic dataset should contain driving session data associated to SOH values ranging from 100% to 80% (end of life); therefore we have set `EFC_LEVELS` = [0, 100, 200, ..., 2000] (since EFC = 2000 roughly corresponds to SOH = 80%). Moreover, to span different driving conditions (city, urban and highway driving[4]), 12 standard drive cycles [186] have been included in `DC_LIST`; they are listed in tab. 4.3. Additionally, we have also included three driving cycles (2x highway, 1x urban) extracted from the e-up! real dataset (chap. 5), for a total of 15 drive cycles.

The dataset generation procedure has been run once for each EV model. Each

[4]We can categorize drive cycles in three categories: city (max speed <50 km/h), urban (max speed <90 km/h), highway (max speed >90 km/h).

Drive cycle name	Type
Artemis Urban	city
Artemis Rural Road	urban
Artemis Motorway 130 kmph	highway
ECE R15	city
EUDC	highway
FTP72	urban
IM240	urban
J1015	city
LA92	urban
NYCC	city
UDDS	urban
US06	highway

Table 4.3: Standard drive cycles included in `DC_LIST`.

of the two resulting datasets consists of a combination of 315 driving sessions, each associated to a specific (SOH, drive cycle) combination. There is no invalid or missing data. Before using these two synthetically generated datasets, they are further cleaned and preprocessed as explained in sec. 6.1.

Chapter 5

Real dataset

5.1 Data acquisition

To further expand our collection of EV monitoring data and in order to parametrize our Simulink EV model correctly (sec. 4.2.1), two datasets of real EV field measurements were acquired from a private EV fleet management company (sec. 1.3). The datasets span several months of data acquisition from a VW e-up! and a VW e-Golf. Many battery SOH tests were performed throughout the monitored time periods, as reported in the next section.

(a) VW e-up!

(b) VW e-Golf

Figure 5.1: The two Volkswagen EV models monitored by the private EV fleet management company.

5.2 Data exploration

The two vehicles were monitored over different time periods[1]:

- e-up!: 20/10/2021 - 18/12/2021
- e-Golf: 04/06/2019 - 12/06/2019, 27/06/2019 - 22/10/2020, 16/11/2020 - 07/12/2020, 15/02/2021 - 10/03/2021, 06/04/2021 - 10/05/2021, 12/07/2021 - 07/09/2021

Battery tests were performed over the monitored time periods for both models. The e-up! underwent a single battery test on 06/12/2021, which returned an SOH of 88%. The e-Golf underwent 16 battery tests, whose dates and returned SOH values are reported in tab. 5.1 for convenience.

Test date	SOH(%)	Test date	SOH(%)
22/07/2020	95	08/07/2021	93
10/09/2020	95	12/07/2021	95
23/11/2020	95	16/08/2021	93
29/11/2020	93	23/08/2021	95
17/02/2021	94*	15/09/2021	> 100*
08/03/2021	95	02/11/2021	89*
26/04/2021	95	05/11/2021	93*
03/05/2021	95	22/11/2021	91*

Table 5.1: Battery tests performed on the monitored VW e-Golf. Asterisk marks (*) indicate that the battery test doesn't fulfill the standard conditions for a reliable SOH estimation (i.e. driving the car from 100% SOC to $< 10\%$ SOC, as discussed in sec. 1.3), thus the estimated SOH value might not reflect the actual one precisely. The estimated SOH values have a maximum error of ± 1.5, as claimed by the private EV fleet management company from whom the data was acquired.

Training a machine learning algorithm for the real-time on-board SOH estimation requires having extensive monitoring data collected at different SOH levels. Unfortunately, the estimated SOH values during the monitored time periods exhibit very little variability. Therefore, e-up! and e-Golf real datasets will be used only for evaluating the performances of the proposed SOH estimation procedure (sec. 6.4). To make the two datasets suitable for testing, only data of driving sessions associated to known battery test SOH values may be selected. We will make the

[1] dates are in dd/mm/yyyy format

assumption that data spanning from one week before and one week after a battery test (where available[2]) is associated with the SOH value returned by that battery test. Whenever two contiguous battery tests were performed less than two weeks away from each other, less than a week of monitoring data is selected after or before those battery tests, in order to avoid any overlapping.

The provided monitoring data is characterized by many signals, whose measurements were collected by the BMS of the EV at different sampling rates and read by a monitoring device through the OBD port. The signals monitored by the BMS vary among different EV models; however, being designed from the same manufacturer, the VW e-up! and VW e-Golf have almost the same set of monitored signals. A complete list of all the common signals between the e-up! and the e-Golf is reported in tab. 5.2.

Signal name	Description	Avg. sampling rate (s)
CUMULATIVE_CC	Cumulative charged charge [C]	33
CUMULATIVE_CE	Cumulative charged energy [J]	33
CUMULATIVE_DC	Cumulative discharged charge [C]	33
CUMULATIVE_DE	Cumulative discharged energy [J]	33
CURRENT	Current flowing through the pack [A]	0.15, 0.1
ENERGY_REMAINING_EXPECTED	Remaining energy (expected) [J]	23
IGNITION	Whether the vehicle is on (1) or off (0)	7
MILEAGE	Mileage [km]	17
SERIAL_BATTERY	Serial number of the pack (string)	100
SOC_DISPLAY	Displayed SOC(%)	7
SOC_REAL	Actual SOC(%)	7
SPEED	Speed [km/h]	13, 19
T_CELL_AVG	Average cell temperature [°C]	12
T_CELL_MAX	Maximum cell temperature [°C]	12
T_CELL_MIN	Minimum cell temperature [°C]	12
T_MODULE_n	Temperature of the n-th module [°C]	13, 4
T_OUTSIDE	External temperature [°C]	100, 14
VOLTAGE	Terminal voltage of the pack [V]	0.15, 0.1
VOLTAGE_12V	Terminal voltage of the ignition battery [V]	21
VOLTAGE_CELL_n	Terminal voltage of the n-th cell [V]	1
VOLTAGE_CELL_MAX	Maximum cell terminal voltage [V]	1
VOLTAGE_CELL_MAX_NUMBER	Cell with the highest terminal voltage	1
VOLTAGE_CELL_MIN	Minimum cell terminal voltage [V]	1
VOLTAGE_CELL_MIN_MUMBER	Cell with the lowest terminal voltage	1
VOLTAGE_CONN	Voltage at the connector [V]	0.15, 0.1

Table 5.2: Signals monitored by the BMS and read by a monitoring device through the OBD port. When sampling rates are different between the two models, the e-up! sampling time is reported first. In both datasets there is no invalid data; however "SPEED" measurements are missing in some e-Golf driving sessions.

[2]As evident from tab. 5.1, there are some battery tests which were performed outside of the monitored time periods, therefore no monitoring data is available close in time to them

The e-up! data was provided directly on a proprietary online platform, in a format which makes it easy to explore and manage the data. Data associated with each driving session can be selected and downloaded in csv format directly from the platform. On the other hand, the e-Golf data was provided in raw csv files, from which it was difficult to tell different driving sessions apart. Moreover, "SPEED" measurements are missing in some of these files. Therefore, in this case, driving sessions have been manually identified and extracted with the following procedure:

- where the "SPEED" measurements are available, a driving session corresponds to the time interval from the first sampled non-zero speed value to the last one, with sub-intervals of at most five minutes in which speed may be zero.
- where the "SPEED" measurements are not available, a driving session corresponds to the time interval from the first increase in "MILEAGE" value to the last one, with sub-intervals of at most five minutes in which mileage may not increase.

The csv files containing driving session data have rows formatted as

```
Timestamp, Timestamp (Unix Microseconds), Signal Type, Value
```

For our aims, only "SPEED", "VOLTAGE", "CURRENT", "SOC_REAL" and "TEMPERATURE" signals are selected. Specifically: "SPEED" is needed to add real drive cycles to `DC_LIST` for the synthetic dataset generation (sec. 4.4); "VOLTAGE", "CURRENT", "SOC_REAL" and "TEMPERATURE" are needed to parametrize Simulink's GBM block (sec. 4.2.1) and – with the exception of "TEMPERATURE" – to test the performances of the proposed SOH estimation procedure (chap. 6). The resulting dataset of extracted driving sessions, each associated to a known SOH level, will undergo further cleaning and preprocessing steps in sec. 6.1.

At a later time, a further set of battery test field data for six different e-Golf vehicles became available, with associated SOH values: 93%, 94%, 99%, 94%, 96%, 98%. They were acquired as raw csv files and processed as already explained. Finally, they were concatenated to the existing e-Golf driving sessions dataset.

Chapter 6

Experiments and results

In this chapter, several novel SOH estimation procedures are introduced, each consisting in a combination of one feature extraction method followed by one regression model. Synthetic and real datasets, described in chapters 4 and 5 respectively, are used to train and validate regression models for the SOH estimation, and finally to test their performances.

6.1 Data preprocessing

Several preprocessing steps have been performed on the two datasets, before using them for training and testing purposes. The two preprocessing pipelines are described in the following sections.

6.1.1 Synthetic dataset

The two synthetic datasets generated in sec. 4.4 have been used to train the proposed SOH estimation procedures and to evaluate their performances. The following preprocessing steps have been performed on each driving session of the two datasets:

1. Filter out signals other than voltage, current and SOC.

2. Set the first voltage and SOC measurements by backward linear interpolation:

$$\mathrm{V}(0) = \frac{\mathrm{V}(t_2) - \mathrm{V}(t_1)}{t_2 - t_1}(-t_1) + \mathrm{V}(t_1) \tag{6.1}$$

$$\mathrm{SOC}(0) = \frac{\mathrm{SOC}(t_2) - \mathrm{SOC}(t_1)}{t_2 - t_1}(-t_1) + \mathrm{SOC}(t_1) \tag{6.2}$$

where t_1 and t_2 are respectively the second and third timestamp at which signals were sampled[1]. This is done in order to address a bug in Simulink's GBM block (sec. 4.1.7) which gives incorrect voltage and SOC measurements at the beginning of the simulation.

3. Format data into a multi-index Pandas dataframe, attaching the SOH value associated to the current driving session and a short identifier for the name of the drive cycle.

The preprocessed driving sessions are then concatenated into a single dataset. At this point, each dataset consists of 315 driving sessions from half an hour to several hours long.

Each driving session is then split up into several non-overlapping 5-minute-long time windows which inherit the SOH value associated to that session. The time window extraction algorithm is reported in appendix B.2. A total of 10192 time windows are extracted from the e-up! dataset, and a total of 24224 from the e-Golf dataset. The data associated to each time window can be viewed as a multivariate time series with $C = 3$ channels (V, I, SOC) and length $T = 1500$ (i.e. 300 s $\times$ 5 measurements/s), therefore the resulting datasets of extracted time windows are time series datasets (def. 3.2.2) with $N_{\text{e-up!}} = 10192$ and $N_{\text{e-Golf}} = 24224$ samples respectively.

Since synthetic data will be used both for training regression models and testing them, each of the two time windows datasets is randomly split into a training set (80% of the time windows) and a test set (20%). Time windows are sampled in a stratified fashion, so that the distribution of different SOH levels in each subset is the same as in the original dataset.

Finally, standardization is applied to each signal, both in training and test set. Namely, the measurements $s_1, \dots, s_T$ of each signal are transformed to:

$$z_i = \frac{s_i - \bar{\mathbf{s}}^{(\text{train})}}{\hat{\sigma}_{\mathbf{s}}^{(\text{train})}} \tag{6.3}$$

where $\bar{\mathbf{s}}^{(\text{train})}$ and $\hat{\sigma}_{\mathbf{s}}^{(\text{train})}$ are respectively the sample mean and sample standard deviation of the signal measurements in the training set. Computing the mean and standard deviation with respect to only the training set avoids data leakage from test data, which must be kept unseen until the end of the training phase.

We can convert each time series dataset to a static one by extracting relevant features from each time window. Feature extraction may be performed in a variety of ways, outlined in sec. 6.2.

[1] we set `sampling_rate` = 0.2 s, so $t_1 = 0.2$ s and $t_2 = 0.4$ s.

6.1.2 Real dataset

Real field data (sec. 5.2) has been used to evaluate the performances of the SOH estimation procedures, previously trained on synthetic data. The following preprocessing steps have been performed on each driving session of the two real datasets:

1. Filter out signals other than voltage, current and SOC.
2. Resample signals through linear interpolation with the same sampling rate specified for the synthetic dataset (sec. 4.4). This is done in order to achieve a common time base for all signals.
3. Format data into a multi-index Pandas dataframe, attaching the SOH value associated to the current driving session and a short identifier for the name of the drive cycle. Each signal will have its own column.

The preprocessed driving sessions are concatenated into a single dataset. The synthetic and real datasets are now formatted all in the same way.

Time windows are then extracted from driving sessions, as already described in sec. 6.1.1. $N_{\text{e-up!}} = 164$ and $N_{\text{e-Golf}} = 1445$ samples are extracted in total. Similarly, signal standardization (eq. 6.3) is applied at the end, reusing the already computed $\bar{\mathbf{s}}^{(\text{train})}$ and $\hat{\sigma}_{\mathbf{s}}^{(\text{train})}$. As already done for synthetic data, feature extraction is also performed on the two resulting real time windows datasets (sec. 6.2).

6.2 Feature extraction

Feature extraction provides a static representation for each time series dataset, making it easier to train regression models. To perform feature extraction, three approaches may be taken:

- MiniRocket's feature extraction for multivariate time series (sec. 3.3.3)
- OLS feature extraction (sec. 3.5.1)
- Theil-Sen feature extraction (sec. 3.5.2)

Each approach has its pros and cons. MiniRocket extracts 9,996 features from each time series and is considered the state-of-the-art for TSER problems. However, it needs to be fit to training data in order to use it; moreover, its transform operation is computationally expensive, as almost 10,000 convolutions per time window need to be computed. OLS feature extraction[2] generates 3 features per

[2]The key idea that motivates feature extraction based on OLS and Theil-Sen linear regression to time windows of EV monitoring data has been discussed in sec. 3.5.

time series, does not need to be fit on training data and is computationally cheaper than MiniRocket. However, it is not robust to outliers in the V-I-SOC space. Theil-Sen feature extraction[2] is similar to OLS, but is more robust to outliers. However, it is more computationally demanding than both OLS and MiniRocket.

Due to the huge number of features extracted by MiniRocket, PCA is applied to MiniRocket-transformed datasets. By visualizing the scree plot for the two synthetic training sets (fig. 6.1), we observe that with just 32 features out of the original 9,996 we can explain over 90% of the total variance in the data. A different PCA problem is solved for each of the two synthetic training sets. The first 32 principal components of the training sets are then used for all PCA transforms. Since PCA requires standardized data, time series data transformed through MiniRocket is standardized before the PCA transform[3]. Furthermore, the PCA transform does not necessarily output standardized data, so standardization is applied also after every PCA transform[3].

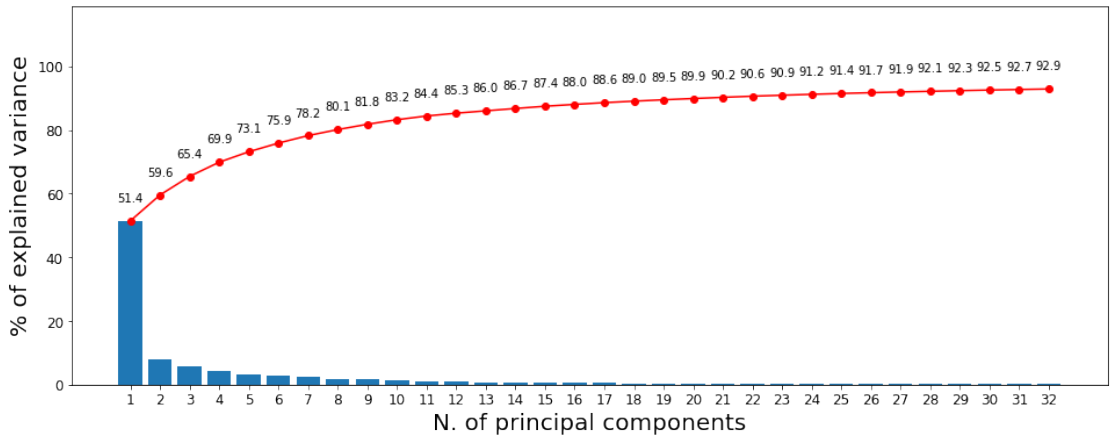


Figure 6.1: Scree plot for the MiniRocket-transformed e-Golf training set. The scree plot for the e-up! training set is very similar, thus it is not reported.

6.3 Training regression models

Several regression models can be selected for predicting the SOH from a transformed time window. In sec. 3.6, three models were discussed:

- Ridge regression via SGD (RR, sec. 3.6.1)
- Random forest (RF, sec. 3.6.2)

[3]with sample mean and sample standard deviation defined by the synthetic training sets

- Feed-forward neural network (NN, sec. 3.6.3)

Three instances of each model are trained, one for each feature extraction method (for a total of 9 regression models per EV model). The training set used is the one introduced in sec. 6.1.1. Grid search with 5-fold cross-validation was performed to tune some of the hyperparameters of ridge regression and random forest models. The neural network hyperparameters were instead tuned by "trial and error", due to the higher computational cost of training such models; 10% of the training set was set aside before training the neural network, to validate its performances at the end of each epoch. The optimal hyperparameters found are reported in appendix B.3. The workstation used to perform the training has the following specifications: Intel Xeon W-2155 CPU @ 3.30 GHz (10 cores, 20 threads), 64GB RAM, NVIDIA Quadro P4000 GPU (8GB GDDR5).

To speed up training, parallel computing has been performed by distributing operations across all the CPU cores available. Specifically, it has been applied during the cross-validation of ridge regression and random forest models.

6.4 Performance evaluation

The trained regression models have been tested on the held out test sets of synthetic data and on the real datasets. The performances of each model are evaluated in terms of MAE (sec. 3.7). For each feature extraction method the average feature extraction time is reported, as this metric is critical for a real-time SOH estimation procedure. Results are reported in tab. 6.1.

All the feature extraction methods proposed are very fast to apply (< 1 s per time window); the time for predicting the SOH value happens to be negligible ($< 10^{-3}$ s per time window) for all regression models, thus it was not reported. We can therefore conclude that every possible combination of a single feature extraction method with a single regression model defines a SOH estimation procedure which fulfils the real-time requirement. We observe that RF+OLS and RF+TS perform very well on synthetic data. However, all regression procedures see a drastic decrease in performances when tested on field data from real EVs. It may seem that MR+RF is the best performing procedure for SOH estimation of real EVs, achieving a relatively low MAE of 2.36. However, the R^2 coefficient between the real and predicted SOH values is negative for all procedures applied on the real e-Golf dataset[4], meaning that the average of the real SOH values is still a better estimator.

[4]whereas it doesn't make any mathematical sense to compute the R^2 in the case of the e-up! dataset, as the only represented SOH value is 0.88; see sec. 3.7

	Synthetic			Real			
	RR	RF	NN	RR	RF	NN	avg. time (s)
e-up!							
MR	2.40	2.17	2.12	9.70	5.22	8.56	0.053
OLS	4.28	0.90	1.36	5.10	5.70	10.67	$1.9 \cdot 10^{-3}$
TS	4.13	0.75	1.23	9.28	7.91	10.88	0.54
e-Golf							
MR	3.68	3.23	2.18	6.96	2.62	9.91	0.046
OLS	4.37	2.36	2.91	5.23	5.56	7.81	$1.8 \cdot 10^{-3}$
TS	4.35	1.76	2.14	4.19	3.85	4.44	0.47

Table 6.1: MAE of the SOH values predicted with the proposed regression procedures. Column names refer to regression models, row names refer to feature extraction methods. The last column is the average feature extraction time from a single time window.

We claim that this underwhelming result may be attributed to the very nature of the available data, and not on the procedure itself. Specifically:

- the key assumption made (namely that operating points lay on a 3D plane uniquely defined by the current SOH) may be accurate for synthetically generated data, but not enough for field data from a real EV. This would indicate that the EV Simulink model described in sec. 4 is not sufficiently complex as to replicate the electrical and mechanical behavior of a real EV. In a sense, this problem is unavoidable: even if we made said Simulink model "richer" and closer to reality, we would have a hard time finding specific experimental data to parametrize it.

- the Simulink model may have overfit the experimental data used in the parameter estimation procedure (sec. 4.2.1). Also in this case, the Simulink model may not replicate the behavior of a real EV accurately. To deal with this issue, we may repeat the parameter estimation using more battery test driving sessions.

- a higher amount and/or more diverse data is required to train the regression models. In this sense, we could generate more synthetic data in more varied driving conditions and check if retrained models give better SOH predictions.

Many possible enhancements exist, which could theoretically increase the performances of the novel SOH estimation procedure introduced in this work. They are investigated briefly in the final chapter, as a reference for potential future works.

An overview of the approach followed to train and test the proposed SOH estimation procedure is visualized in fig. 6.2.

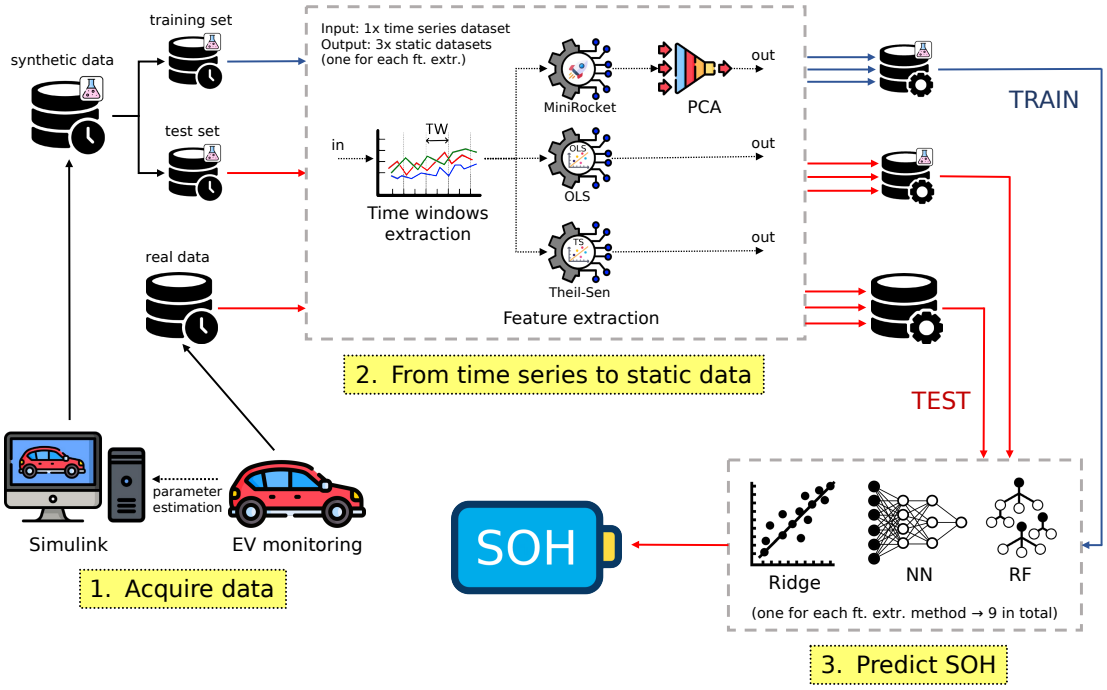


Figure 6.2: Overview of the data pipeline designed to train and test the proposed SOH estimation procedure.

Chapter 7

Conclusion

In this work, a novel data-driven approach to Li-ion battery pack SOH estimation has been proposed. Due to a lack of publicly available datasets of EV field data, a Simulink model of an EV has been built and parametrized according to the technical specifications of two real EVs (VW e-up!, VW e-Golf). The model has been used to simulate a high number of driving sessions in different driving conditions, collecting synthetically-generated EV field data. Specifically, the resulting synthetic dataset consists of voltage, current and SOC measurements, recorded at a high sampling rate. Multiple fixed-length time windows were extracted from each synthetic driving session and three feature extraction methods (MiniRocket, OLS regression, Theil-Sen regression) were applied on them, thus obtaining a static representation of time series data. The transformed data was finally used to train three different regression models (ridge regression, random forest, feed-forward neural network).

The proposed procedure has been tested both on a held-out test set of synthetic data and on a real dataset of EV monitoring data, acquired from a private EV fleet management company. The models generally achieved a low MAE on synthetic data (as low as 0.75%, obtained with Theil-Sen feature extraction followed by a random forest regression model). Moreover, the procedure has been shown to be very fast in making predictions (with at most half a second per time window), making it suitable for monitoring SOH in real-time. However, performances on real data were underwhelming. Supposedly, the poor generalization capabilities exhibited by the trained models are due to the low quantity, quality and/or diversity of the synthetically generated data, perhaps indicating that the Simulink EV model is not as close to reality as expected. Despite the promising results achieved on synthetic data, further work is needed to improve the performances of the proposed SOH estimation procedure on real EV data.

7.1 Future work

Even though the performances of the procedure applied on real data were below expectations, we claim that there exist many possible improvements to our approach, which could theoretically enhance the regression models' generalization ability.

The main improvements regard the quantity, quality and diversity of the data used for training. Specifically, we may:

- use the Simulink EV model to further generate synthetic data in more diverse driving conditions (e.g.: simulating more driving cycles and finer battery aging levels), thus increasing the size of the training set;

- increase the sampling rate. We used a sampling rate of 5 Hz (i.e. a measurement every 0.2 seconds), but higher sampling rates can be tested, enhancing the reliability of the measurements at the price of a higher computational cost for feature extraction;

- implement data augmentation techniques specific for time series data [187] to virtually increase the size of the training set;

- enhance the reliability of the Simulink EV model. For example, we could model the battery pack at the module or cell level (instead of approximating it to a single high-voltage battery cell), and we could explicitly implement the thermal management/cooling system of an EV (instead of relying on a simple battery thermal model). However, this is viable only upon acquiring more fine-grained EV specifications, such as the specific electrical and geometrical layout of the cells inside the battery pack, its positioning inside the car, the specific features of the BMS and of the battery thermal management system (BTMS) in use;

- improve the thermal model of the current Simulink EV model, so that the temperature dependence on the SOH is correctly accounted for. This improvement would make it possible to exploit also the extracted temperature measurements to predict SOH.

- train the regression models on a "hybrid" training set, composed of both synthetically-generated and real data.

- acquire new real data, over a wider time period (i.e. a higher amount of different SOH values between 100% and 80%). In fact, with a sufficiently extensive and representative amount of real data we could entirely avoid generating synthetic training data.

Moreover, also the proposed algorithms could be improved. We may:

- extract longer time windows, in order to capture a wider set of operating points in different driving conditions. This comes at the cost of an increased computational time for applying feature extraction.
- with respect to the feature extraction method based on linear regression in the V-I-SOC space, experiment with different types of regression other than the linear one (e.g. Bayesian linear regression, polynomial regression).
- experiment with other regression models, such as support vector regression, Gaussian process regression and deep neural networks. Interestingly, there exist machine learning models which can at the same time learn how to extract relevant features from time series data and how to use them to perform extrinsic regression. Among them, a CNN known as InceptionTime [90, 188] achieves state of the art performances on many TSER datasets. Using deep neural networks would also enable the use of transfer learning [189], a widely used methodology for increasing a model's generalization ability with just a little amount of target-domain data.

Appendix A

EV Simulink model - figures

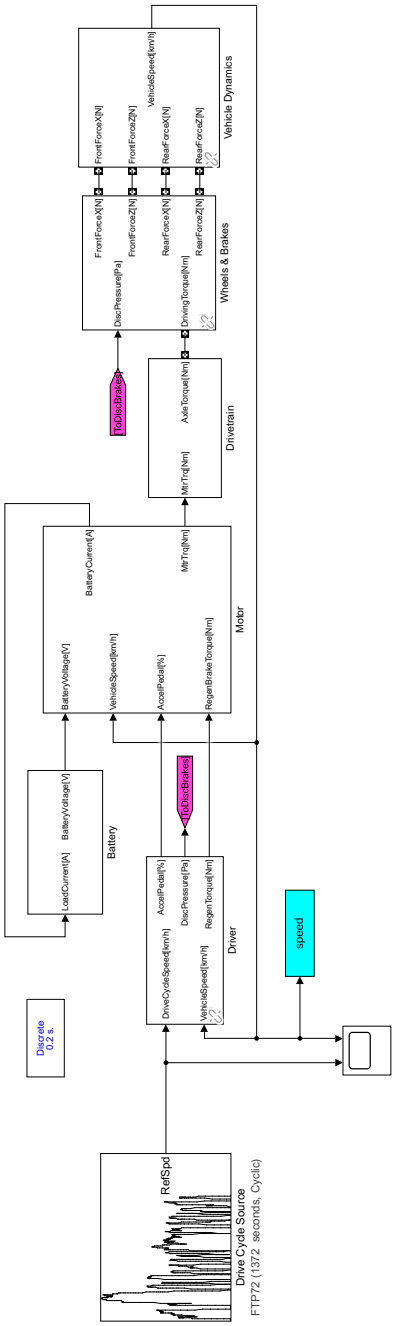


Figure A.1: Simulink EV model

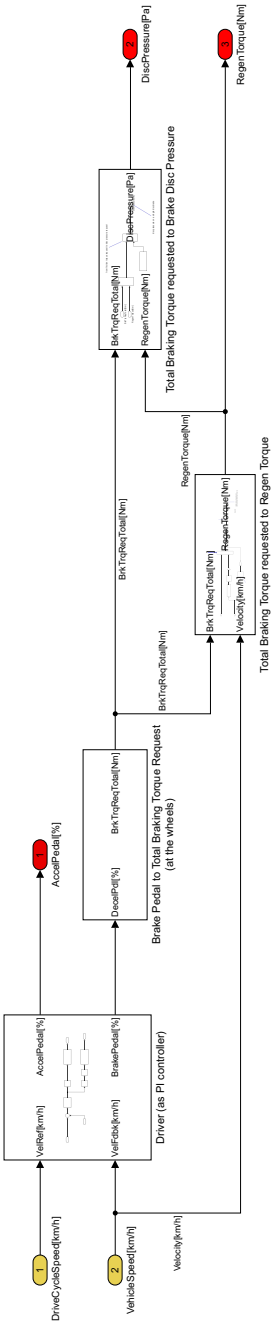


Figure A.2: Simulink EV model - driver and braking subsystems

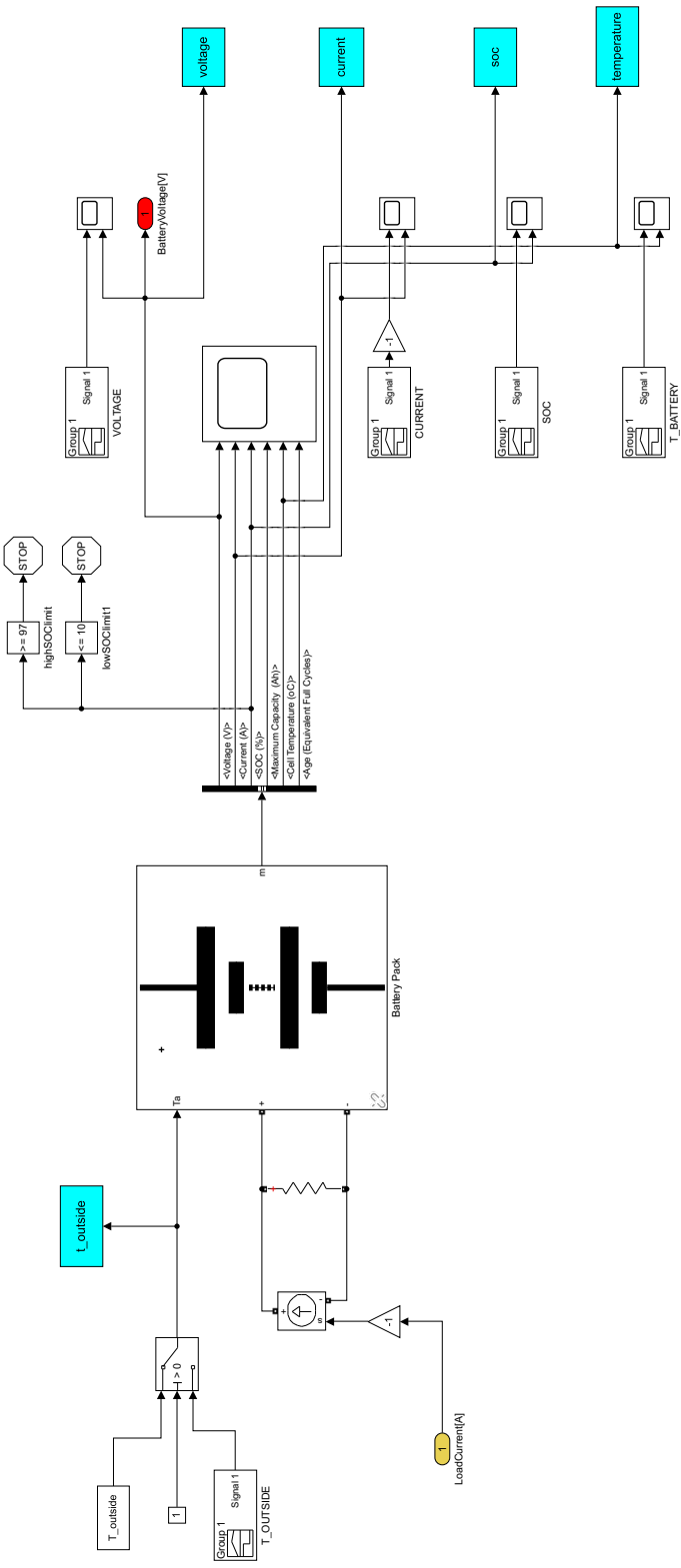


Figure A.3: Simulink EV model - battery pack subsystem

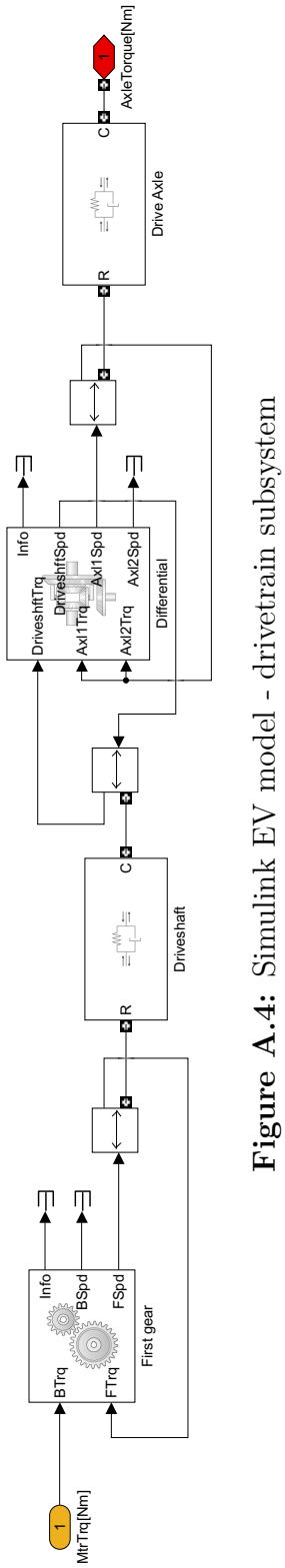


Figure A.4: Simulink EV model - drivetrain subsystem

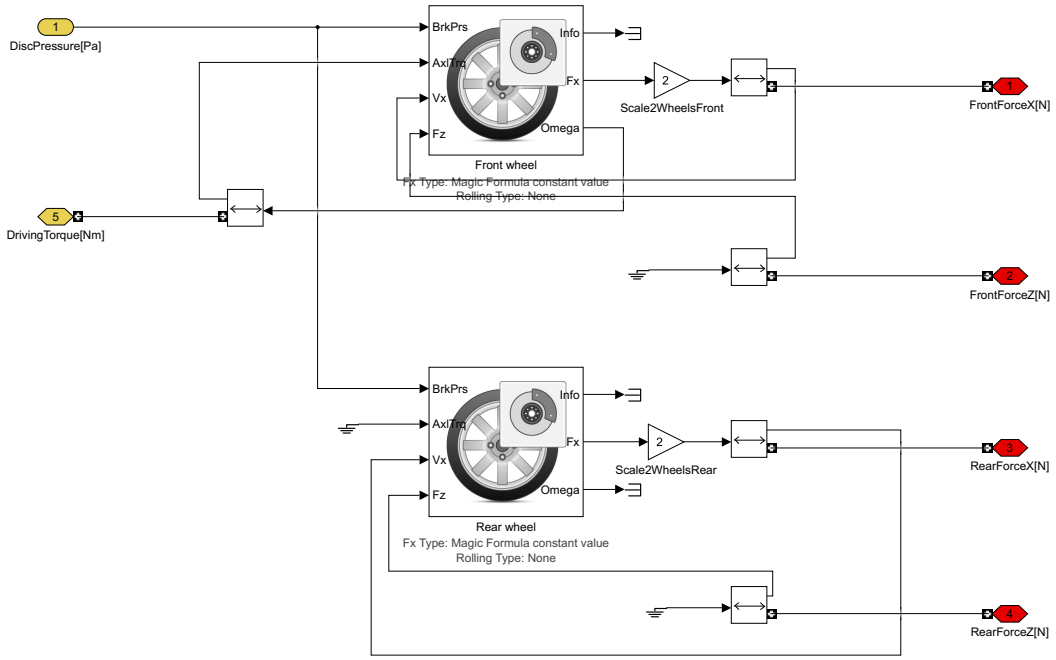


Figure A.5: Simulink EV model - wheels subsystem

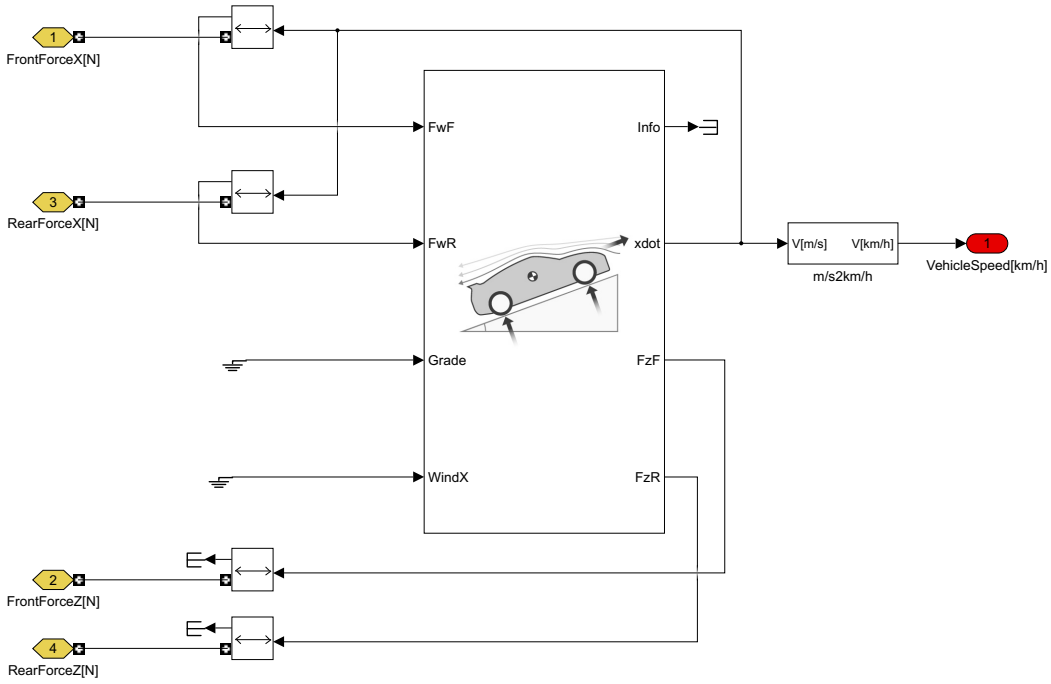


Figure A.6: Simulink EV model - vehicle body subsystem

Appendix B

Additional notes

B.1 Spatial median

Spatial medians are a family of multivariate medians, i.e. an extension of the notion of univariate median to multivariate distributions. Multiple Theil-Sen estimation (sec. 3.5.2) uses a particular spatial median defined via a depth function, first introduced in [123].

Let $\mathbf{Z} \in \mathbb{R}^d$ be a random vector with joint probability distribution Q. The statistical spatial depth is defined as

$$D_{\text{sp}}(\mathbf{z}, Q) = 1 - \|\mathbb{E}_Q S(\mathbf{z} - \mathbf{Z})\|, \qquad \mathbf{z} \in \mathbb{R}^d$$

where $S(\mathbf{v}) = \mathbf{v}/\|\mathbf{v}\|$ ($S(\mathbf{0}) = \mathbf{0}$) is the spatial sign function, i.e. the projection of $\mathbf{v} \in \mathbb{R}^k$ to a k-sphere.

For a random sample $\mathbf{Z_1}, \ldots, \mathbf{Z_n}$ drawn according to Q, the sample spatial depth is defined as

$$\tilde{D}_{\text{sp}}(\mathbf{z}, Q_n) = 1 - \left\| \frac{1}{n} \sum_{i=1}^{n} S(\mathbf{z} - \mathbf{Z_i}) \right\|, \qquad \mathbf{z} \in \mathbb{R}^d$$

where Q_n is the empirical distribution over the random sample. Then, the statistical (sample) spatial median m ($\tilde{m}$) is the maximizer of the statistical (sample) spatial depth, i.e.

$$m = \arg\sup_{x \in \mathbb{R}^d} D_{\text{sp}}(\mathbf{x}, Q) \qquad \tilde{m} = \arg\sup_{x \in \mathbb{R}^d} \tilde{D}_{\text{sp}}(\mathbf{z}, Q_n)$$

B.2 Time windows extraction

The algorithm for extracting time windows from a dataset of driving sessions' data is the following

Algorithm 2 Time windows extraction

```
# Input arguments
dataset: the dataset from which to extract time windows
length: duration of the time windows [s] (default: 300)
slide: time between the starting points of two consecutive time windows [s] (default: length, i.e. non-overlapping time windows)
freq: sampling rate of the measurements [Hz] (default: 5)
max_SOC: time windows do not contain SOC values above this one (default: 100)
min_SOC: time windows do not contain SOC values under this one (default: 0)
random_starting_point: choose the starting point of the first time window at random in the interval [0, length] (default: true)

# Function
time_windows_dataset ← empty list
for driving session ∈ dataset do
    duration ← duration of the current driving session
    if random_starting_point then
        start ← 𝒰(0, length)
    else
        start ← 0
    end if
    end ← start + length
    while end ≤ duration do
        time_window ← [start, end]
        if SOC values in time_window are between max_SOC and min_SOC then
            append data of time_window to time_windows_dataset
        end if
        start ← start + slide
        end ← end + slide
    end while
end for
return time_windows_dataset
```

B.3 Hyperparameter tuning

The hyperparameters chosen for each regression model are reported in the following table.

	Ridge			Random forest		Neural network					
	α	η_0	p	max. tree depth	n. trees	neurons per layer	max. epochs	α	η_0	μ	P
e-up!											
MiniRocket	0.0278	0.001	0.25	30	1000	$(120, 60, 40)$	300	0.0001	0.1	0.9	5
OLS	0.01	0.0464	0.25	50	500	$(40, 20)$	300	0.0001	0.1	0.9	5
Theil-Sen	0.01	0.0464	0.25	unlimited	500	$(40, 20)$	300	0.0001	0.1	0.9	5
e-Golf											
MiniRocket	0.01	0.0008	0.25	30	1000	$(120, 60, 40)$	300	0.0001	0.1	0.9	5
OLS	0.01	0.0464	0.25	15	500	$(40, 20)$	300	0.0001	0.1	0.9	5
Theil-Sen	0.01	0.0464	0.25	15	500	$(40, 20)$	300	0.0001	0.1	0.9	5

Table B.1: Hyperparameters chosen for each regression model and training set. α is the regularization factor, η_0 is the initial learning rate. The ridge regression model is trained via SGD with learning rate computed at each iteration as $\eta(i) = \eta_0/i^p$. The neural network is trained via SGD with learning rate divided by 10 whenever the training loss doesn't decrease enough for P consecutive epochs.

Bibliography

[1] International Energy Agency (IEA). *Net Zero by 2050*. May 2022. URL: https://www.iea.org/reports/net-zero-by-2050 (visited on Nov. 11, 2022) (cit. on p. 1).

[2] International Energy Agency (IEA). *Greenhouse Gas Emissions from Energy Data Explorer*. Nov. 2022. URL: https://www.iea.org/data-and-statistics/data-tools/greenhouse-gas-emissions-from-energy-data-explorer (visited on Nov. 11, 2022) (cit. on p. 1).

[3] International Energy Agency (IEA). *Transport sector CO2 emissions by mode in the Sustainable Development Scenario*. Oct. 2022. URL: https://www.iea.org/data-and-statistics/charts/transport-sector-co2-emissions-by-mode-in-the-sustainable-development-scenario-2000-2030 (visited on Nov. 11, 2022) (cit. on p. 1).

[4] European Environment Agency (EEA). *Transport and environment report 2021 – Decarbonising road transport — the role of vehicles, fuels and transport demand*. Feb. 2022. URL: https://www.eea.europa.eu/publications/transport-and-environment-report-2021 (visited on Nov. 11, 2022) (cit. on p. 1).

[5] Nansi Xue, Wenbo Du, Thomas A. Greszler, Wei Shyy, and Joaquim R.R.A. Martins. «Design of a lithium-ion battery pack for PHEV using a hybrid optimization method». In: *Applied Energy* 115 (2014), pp. 591–602. ISSN: 0306-2619. DOI: 10.1016/j.apenergy.2013.10.044 (cit. on p. 1).

[6] Gonçalo dos Reis, Calum Strange, Mohit Yadav, and Shawn Li. «Lithium-ion battery data and where to find it». In: *Energy and AI* 5 (2021), p. 100081. ISSN: 2666-5468. DOI: 10.1016/j.egyai.2021.100081 (cit. on pp. 2, 14).

[7] Lip Huat Saw, Yonghuang Ye, and Andrew A.O. Tay. «Integration issues of lithium-ion battery into electric vehicles battery pack». In: *Journal of Cleaner Production* 113 (2016), pp. 1032–1045. ISSN: 0959-6526. DOI: 10.1016/j.jclepro.2015.11.011 (cit. on p. 2).

[8] Micah S. Ziegler and Jessika E. Trancik. «Re-examining rates of lithium-ion battery technology improvement and cost decline». In: *Energy Environ. Sci.* 14 (4 2021), pp. 1635–1651. DOI: 10.1039/D0EE02681F (cit. on p. 2).

[9] *Batterie des Porsche Taycan*. 2022. URL: https://presse.porsche.de/prod/presse_pag/PressResources.nsf/Content?ReadForm&languageversionid=878434 (visited on Nov. 12, 2022) (cit. on p. 2).

[10] Hossam A. Gabbar, Ahmed M. Othman, and Muhammad R. Abdussami. «Review of Battery Management Systems (BMS) Development and Industrial Standards». In: *Technologies* 9.2 (2021). ISSN: 2227-7080. DOI: 10.3390/technologies9020028 (cit. on p. 2).

[11] Achim Kampker, Heiner H. Heimes, Mathias Ordung, Christoph Lienemann, Ansgar Hollah, and Nemanja Sarovic. *Evaluation of a Remanufacturing for Lithium Ion Batteries from Electric Cars*. Version 10006102. Nov. 2016. DOI: 10.5281/zenodo.1128215 (cit. on p. 3).

[12] Walter A. van Schalkwijk and Bruno Scrosati. «Advances in Lithium Ion Batteries Introduction». In: *Advances in Lithium-Ion Batteries*. Boston, MA: Springer US, 2002, pp. 1–5. ISBN: 978-0-306-47508-5. DOI: 10.1007/0-306-47508-1_1 (cit. on p. 3).

[13] Subrahmanyam Goriparti, Ermanno Miele, Francesco De Angelis, Enzo Di Fabrizio, Remo Proietti Zaccaria, and Claudio Capiglia. «Review on recent progress of nanostructured anode materials for Li-ion batteries». In: *Journal of Power Sources* 257 (2014), pp. 421–443. ISSN: 0378-7753. DOI: 10.1016/j.jpowsour.2013.11.103 (cit. on p. 3).

[14] Can Cao, Zhuo-Bin Li, Xiao-Liang Wang, Xin-Bing Zhao, and Wei-Qiang Han. «Recent Advances in Inorganic Solid Electrolytes for Lithium Batteries». In: *Frontiers in Energy Research* 2 (2014). ISSN: 2296-598X. DOI: 10.3389/fenrg.2014.00025 (cit. on p. 3).

[15] Davide Castelvecchi. *Electric cars and batteries: How will the world produce enough?* https://www.nature.com/articles/d41586-021-02222-1. (visited on Nov. 12, 2022). Aug. 2021 (cit. on pp. 3, 4).

[16] *A Glossary of Battery Terms*. https://www.batteryspace.com/BatteryGlossary.aspx. (visited on Nov. 12, 2022) (cit. on p. 4).

[17] Feng Leng, Cher Ming Tan, and Michael Pecht. «Effect of Temperature on the Aging rate of Li Ion Battery Operating above Room Temperature». In: *Sci Rep* 5 (Aug. 2015), p. 12967 (cit. on p. 4).

[18] Dennis Doerffel and Suleiman Abu Sharkh. «A critical review of using the Peukert equation for determining the remaining capacity of lead-acid and lithium-ion batteries». In: *Journal of Power Sources* 155.2 (2006), pp. 395–400. ISSN: 0378-7753. DOI: 10.1016/j.jpowsour.2005.04.030 (cit. on p. 4).

[19] *A Guide to Understanding Battery Specifications*. http://web.mit.edu/evt/summary_battery_specifications.pdf. (visited on Nov. 12, 2022). Dec. 2008 (cit. on p. 5).

[20] Jacqueline S. Edge et al. «Lithium ion battery degradation: what you need to know». In: *Phys. Chem. Chem. Phys.* 23 (14 2021), pp. 8200–8221. DOI: 10.1039/D1CP00359C (cit. on p. 5).

[21] Alberto Berrueta, Julio Pascual, Idoia San Martín, Pablo Sanchis, and Alfredo Ursúa. «Influence of the Aging Model of Lithium-Ion Batteries on the Management of PV Self-Consumption Systems». In: *2018 IEEE International Conference on Environment and Electrical Engineering and 2018 IEEE Industrial and Commercial Power Systems Europe (EEEIC / I&CPS Europe)*. 2018, pp. 1–5. DOI: 10.1109/EEEIC.2018.8493778 (cit. on pp. 5, 6).

[22] Markus Schindler, Johannes Sturm, Sebastian Ludwig, Axel Durdel, and Andreas Jossen. «Comprehensive Analysis of the Aging Behavior of Nickel-Rich, Silicon-Graphite Lithium-Ion Cells Subject to Varying Temperature and Charging Profiles». In: *Journal of The Electrochemical Society* 168.6 (June 2021), p. 060522. DOI: 10.1149/1945-7111/ac03f6 (cit. on p. 6).

[23] Ziyou Song, Xiao-Guang Yang, Niankai Yang, Fanny Pinto Delgado, Heath Hofmann, and Jing Suna. «A study of cell-to-cell variation of capacity in parallel-connected lithium-ion battery cells». In: *eTransportation* 7 (2021), p. 100091. ISSN: 2590-1168. DOI: 10.1016/j.etran.2020.100091 (cit. on p. 6).

[24] Mohamed Daowd, Noshin Omar, Peter Van Den Bossche, and Joeri Van Mierlo. «Passive and active battery balancing comparison based on MATLAB simulation». In: *2011 IEEE Vehicle Power and Propulsion Conference*. 2011, pp. 1–7. DOI: 10.1109/VPPC.2011.6043010 (cit. on p. 6).

[25] Wei Liu, Tobias Placke, and K.T. Chau. «Overview of batteries and battery management for electric vehicles». In: *Energy Reports* 8 (2022), pp. 4058–4084. ISSN: 2352-4847. DOI: 10.1016/j.egyr.2022.03.016 (cit. on p. 6).

[26] Samveg Saxena, Caroline Le Floch, Jason MacDonald, and Scott Moura. «Quantifying EV battery end-of-life through analysis of travel needs with vehicle powertrain models». In: *Journal of Power Sources* 282 (2015), pp. 265–276. ISSN: 0378-7753. DOI: 10.1016/j.jpowsour.2015.01.072 (cit. on p. 6).

[27] Samuel Greenbank and David Howey. «Automated Feature Extraction and Selection for Data-Driven Models of Rapid Battery Capacity Fade and End of Life». In: *IEEE Transactions on Industrial Informatics* 18.5 (May 2022), pp. 2965–2973. DOI: 10.1109/tii.2021.3106593 (cit. on p. 6).

[28] Shida Jiang and Zhengxiang Song. «A review on the state of health estimation methods of lead-acid batteries». In: *Journal of Power Sources* 517 (2022), p. 230710. ISSN: 0378-7753. DOI: 10.1016/j.jpowsour.2021.230710 (cit. on pp. 9–12).

[29] Zuolu Wang, Guojin Feng, Dong Zhen, Fengshou Gu, and Andrew Ball. «A review on online state of charge and state of health estimation for lithium-ion batteries in electric vehicles». In: *Energy Reports* 7 (2021), pp. 5141–5161. ISSN: 2352-4847. DOI: 10.1016/j.egyr.2021.08.113 (cit. on pp. 9, 11, 12).

[30] Kong Soon Ng, Chin-Sien Moo, Yi-Ping Chen, and Yao-Ching Hsieh. «Enhanced coulomb counting method for estimating state-of-charge and state-of-health of lithium-ion batteries». In: *Applied Energy* 86.9 (2009), pp. 1506–1511. ISSN: 0306-2619. DOI: 10.1016/j.apenergy.2008.11.021 (cit. on p. 9).

[31] Daniel Juarez-Robles, Anjul Arun Vyas, Conner Fear, Judith A. Jeevarajan, and Partha P. Mukherjee. «Overcharge and Aging Analytics of Li-Ion Cells». In: *Journal of The Electrochemical Society* 167.9 (June 2020), p. 090547. DOI: 10.1149/1945-7111/ab9569 (cit. on p. 9).

[32] Chao Lyu, Tao Zhang, Weilin Luo, Gang Wei, Baozhao Ma, and Lixin Wang. «SOH Estimation of Lithium-ion Batteries Based on Fast Time Domain Impedance Spectroscopy». In: *2019 14th IEEE Conference on Industrial Electronics and Applications (ICIEA)*. 2019, pp. 2142–2147. DOI: 10.1109/ICIEA.2019.8834119 (cit. on p. 10).

[33] Alessandro Falai, Tiziano Alberto Giuliacci, Daniela Anna Misul, and Pier Giuseppe Anselma. «Reducing the Computational Cost for Artificial Intelligence-Based Battery State-of-Health Estimation in Charging Events». In: *Batteries* 8.11 (Nov. 2022), p. 209. ISSN: 2313-0105. DOI: 10.3390/batteries8110209 (cit. on p. 10).

[34] F. Huet. «A review of impedance measurements for determination of the state-of-charge or state-of-health of secondary batteries». In: *Journal of Power Sources* 70.1 (1998), pp. 59–69. ISSN: 0378-7753. DOI: 10.1016/S0378-7753(97)02665-7 (cit. on p. 10).

[35] Nassim Noura, Loïc Boulon, and Samir Jemeï. «A Review of Battery State of Health Estimation Methods: Hybrid Electric Vehicle Challenges». In: *World Electric Vehicle Journal* 11.4 (Oct. 2020), p. 66. DOI: 10.3390/wevj11040066 (cit. on pp. 10, 12).

[36] Linfeng Zheng, Jianguo Zhu, Dylan Dah-Chuan Lu, Guoxiu Wang, and Tingting He. «Incremental capacity analysis and differential voltage analysis based state of charge and capacity estimation for lithium-ion batteries». In: *Energy* 150 (2018), pp. 759–769. ISSN: 0360-5442. DOI: 10.1016/j.energy.2018.03.023 (cit. on p. 10).

[37] Xiaoyu Li, Changgui Yuan, Xiaohui Li, and Zhenpo Wang. «State of health estimation for Li-Ion battery using incremental capacity analysis and Gaussian process regression». In: *Energy* 190 (2020), p. 116467. ISSN: 0360-5442. DOI: 10.1016/j.energy.2019.116467 (cit. on p. 10).

[38] Zeyu Ma, Zhenpo Wang, Rui Xiong, and Jiuchun Jiang. «A mechanism identification model based state-of-health diagnosis of lithium-ion batteries for energy storage applications». In: *Journal of Cleaner Production* 193 (2018), pp. 379–390. ISSN: 0959-6526. DOI: 10.1016/j.jclepro.2018.05.074 (cit. on p. 10).

[39] Yi Li, Mohamed Abdel-Monem, Rahul Gopalakrishnan, Maitane Berecibar, Elise Nanini-Maury, Noshin Omar, Peter van den Bossche, and Joeri Van Mierlo. «A quick on-line state of health estimation method for Li-ion battery with incremental capacity curves processed by Gaussian filter». In: *Journal of Power Sources* 373 (2018), pp. 40–53. ISSN: 0378-7753. DOI: 10.1016/j.jpowsour.2017.10.092 (cit. on p. 10).

[40] Ji'ang Zhang, Ping Wang, Yushu Liu, and Ze Cheng. «Variable-Order Equivalent Circuit Modeling and State of Charge Estimation of Lithium-Ion Battery Based on Electrochemical Impedance Spectroscopy». In: *Energies* 14.3 (2021). ISSN: 1996-1073. DOI: 10.3390/en14030769 (cit. on p. 11).

[41] Htet Aung, Kay-Soon Low, and Jing Jun Soon. «State-of-charge estimation using particle swarm optimization with inverse barrier constraint in a nanosatellite». In: *2015 IEEE 10th Conference on Industrial Electronics and Applications (ICIEA)*. 2015, pp. 1–6. DOI: 10.1109/ICIEA.2015.7334074 (cit. on p. 11).

[42] Lei Yao, Shiming Xu, Aihua Tang, Fang Zhou, Junjian Hou, Yanqiu Xiao, and Zhijun Fu. «A Review of Lithium-Ion Battery State of Health Estimation and Prediction Methods». In: *World Electric Vehicle Journal* 12.3 (Aug. 2021), p. 113. ISSN: 2032-6653. DOI: 10.3390/wevj12030113 (cit. on pp. 11, 12).

[43] Yi Li, Kailong Liu, Aoife M. Foley, Alana Zülke, Maitane Berecibar, Elise Nanini-Maury, Joeri Van Mierlo, and Harry E. Hoster. «Data-driven health estimation and lifetime prediction of lithium-ion batteries: A review». In: *Renewable and Sustainable Energy Reviews* 113 (2019), p. 109254. ISSN: 1364-0321. DOI: 10.1016/j.rser.2019.109254 (cit. on p. 12).

[44] Filippo Boni. «Lithium-ion Battery Diagnostics and State of Health Estimation for Electric Vehicles Applications». MA thesis. Politecnico di Torino, 2021 (cit. on pp. 12, 26).

[45] Robert R. Richardson, Christoph R. Birkl, Michael A. Osborne, and David A. Howey. «Gaussian Process Regression for In Situ Capacity Estimation of Lithium-Ion Batteries». In: *IEEE Transactions on Industrial Informatics* 15.1 (2019), pp. 127–138. DOI: 10.1109/TII.2018.2794997 (cit. on p. 13).

[46] Xuning Feng, Caihao Weng, Xiangming He, Xuebing Han, Languang Lu, Dongsheng Ren, and Minggao Ouyang. «Online State-of-Health Estimation for Li-Ion Battery Using Partial Charging Segment Based on Support Vector Machine». In: *IEEE Transactions on Vehicular Technology* 68.9 (2019), pp. 8583–8592. DOI: 10.1109/TVT.2019.2927120 (cit. on p. 13).

[47] Yuanyuan Li, Shouming Zhong, Qishui Zhong, and Kaibo Shi. «Lithium-Ion Battery State of Health Monitoring Based on Ensemble Learning». In: *IEEE Access* 7 (2019), pp. 8754–8762. DOI: 10.1109/ACCESS.2019.2891063 (cit. on p. 13).

[48] Chao Hu, Gaurav Jain, Puqiang Zhang, Craig Schmidt, Parthasarathy Gomadam, and Tom Gorka. «Data-driven method based on particle swarm optimization and k-nearest neighbor regression for estimating capacity of lithium-ion battery». In: *Applied Energy* 129 (2014), pp. 49–55. ISSN: 0306-2619. DOI: 10.1016/j.apenergy.2014.04.077 (cit. on p. 13).

[49] Kirandeep Kaur, Akhil Garg, Xujian Cui, Surinder Singh, and Bijaya Ketan Panigrahi. «Deep learning networks for capacity estimation for monitoring SOH of Li-ion batteries for electric vehicles». In: *International Journal of Energy Research* 45.2 (2021), pp. 3113–3128. DOI: 10.1002/er.6005 (cit. on p. 13).

[50] Meru A. Patil, Piyush Tagade, Krishnan S. Hariharan, Subramanya M. Kolake, Taewon Song, Taejung Yeo, and Seokgwang Doo. «A novel multistage Support Vector Machine based approach for Li ion battery remaining useful life estimation». In: *Applied Energy* 159 (2015), pp. 285–297. ISSN: 0306-2619. DOI: 10.1016/j.apenergy.2015.08.119 (cit. on p. 13).

[51] Gae-Won You, Sangdo Park, and Dukjin Oh. «Diagnosis of Electric Vehicle Batteries Using Recurrent Neural Networks». In: *IEEE Transactions on Industrial Electronics* 64.6 (2017), pp. 4885–4893. DOI: 10.1109/TIE.2017.2674593 (cit. on p. 13).

[52] B. Saha and K. Goebel. *Battery Data Set*. https://www.nasa.gov/content/prognostics-center-of-excellence-data-set-repository. NASA Prognostics Data Repository, NASA Ames Research Center, Moffett Field, CA. 2007 (cit. on p. 13).

[53] Yaxiang Fan, Fei Xiao, Chaoran Li, Guorun Yang, and Xin Tang. «A novel deep learning framework for state of health estimation of lithium-ion battery». In: *Journal of Energy Storage* 32 (2020), p. 101741. ISSN: 2352-152X. DOI: 10.1016/j.est.2020.101741 (cit. on pp. 13, 26).

[54] Alessandro Aliberti, Filippo Boni, Alessandro Perol, Marco Zampolli, Rémi Jacques Philibert Jaboeuf, Paolo Tosco, Enrico Macii, and Edoardo Patti. «Comparative Analysis of Neural Networks Techniques for Lithium-ion Battery SOH Estimation». In: *2022 IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC)*. 2022, pp. 1355–1361. DOI: 10.1109/COMPSAC54236.2022.00214 (cit. on p. 13).

[55] Robert R. Richardson, Michael A. Osborne, and David A. Howey. «Battery health prediction under generalized conditions using a Gaussian process transition model». In: *Journal of Energy Storage* 23 (2019), pp. 320–328. ISSN: 2352-152X. DOI: 10.1016/j.est.2019.03.022 (cit. on p. 13).

[56] Brian Ospina Agudelo, Walter Zamboni, and Eric Monmasson. «Application domain extension of incremental capacity-based battery SoH indicators». In: *Energy* 234 (2021), p. 121224. ISSN: 0360-5442. DOI: 10.1016/j.energy.2021.121224 (cit. on p. 13).

[57] C Birkl. *Oxford Battery Degradation Dataset 1* (cit. on p. 13).

[58] Wei Liu, Yan Xu, and Xue Feng. «A Hierarchical and Flexible Data-Driven Method for Online State-of-Health Estimation of Li-Ion Battery». In: *IEEE Transactions on Vehicular Technology* 69.12 (2020), pp. 14739–14748. DOI: 10.1109/TVT.2020.3037088 (cit. on p. 13).

[59] Chengqi She, Yang Li, Changfu Zou, Torsten Wik, Zhenpo Wang, and Fengchun Sun. «Offline and Online Blended Machine Learning for Lithium-Ion Battery Health State Estimation». In: *IEEE Transactions on Transportation Electrification* 8.2 (2022), pp. 1604–1618. DOI: 10.1109/TTE.2021.3129479 (cit. on p. 13).

[60] Mingjie Shi, Jun Xu, Chuanping Lin, and Xuesong Mei. «A fast state-of-health estimation method using single linear feature for lithium-ion batteries». In: *Energy* 256 (2022), p. 124652. ISSN: 0360-5442. DOI: 10.1016/j.energy.2022.124652 (cit. on p. 13).

[61] Darius Roman, Saurabh Saxena, Valentin Robu, Michael Pecht, and David Flynn. «Machine learning pipeline for battery state-of-health estimation». In: *Nature Machine Intelligence* 3.5 (May 2021), pp. 447–456. ISSN: 2522-5839. DOI: 10.1038/s42256-021-00312-3 (cit. on pp. 13, 26).

[62] Qianglong Li, Dezhi Li, Kun Zhao, Licheng Wang, and Kai Wang. «State of health estimation of lithium-ion battery based on improved ant lion optimization and support vector regression». In: *Journal of Energy Storage* 50 (2022), p. 104215. ISSN: 2352-152X. DOI: 10.1016/j.est.2022.104215 (cit. on p. 13).

[63] Bixiong Huang, Haiyu Liao, Yiquan Wang, Xintian Liu, and Xiao Yan. «Prediction and evaluation of health state for power battery based on Ridge linear regression model». In: *Science Progress* 104.4 (2021), p. 00368504211059047. DOI: 10.1177/00368504211059047 (cit. on p. 13).

[64] Jong-Hyun Lee and In-Soo Lee. «Estimation of Online State of Charge and State of Health Based on Neural Network Model Banks Using Lithium Batteries». In: *Sensors* 22.15 (2022). ISSN: 1424-8220. DOI: 10.3390/s22155536 (cit. on p. 13).

[65] Duo Yang, Yujie Wang, Rui Pan, Ruiyang Chen, and Zonghai Chen. «A Neural Network Based State-of-Health Estimation of Lithium-ion Battery in Electric Vehicles». In: *Energy Procedia* 105 (2017). 8th International Conference on Applied Energy, ICAE2016, 8-11 October 2016, Beijing, China, pp. 2059–2064. ISSN: 1876-6102. DOI: 10.1016/j.egypro.2017.03.583 (cit. on p. 13).

[66] Alfonso Russo, Andrea Fiore, and Marco Aprea. «Predicting battery health capacity through machine learning techniques: SVR, Random Forest and Fully-connected network». In: (Sept. 2019) (cit. on p. 13).

[67] Ephrem Chemali, Phillip J. Kollmeyer, Matthias Preindl, Youssef Fahmy, and Ali Emadi. «A Convolutional Neural Network Approach for Estimation of Li-Ion Battery State of Health from Charge Profiles». In: *Energies* 15.3 (Feb. 2022), p. 1185. ISSN: 1996-1073. DOI: 10.3390/en15031185 (cit. on pp. 14, 26).

[68] Manali Raman, V. Champa, and V. Prema. «State of Health Estimation of Lithium Ion Batteries using Recurrent Neural Network and its Variants». In: *2021 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT)*. 2021, pp. 1–6. DOI: 10.1109/CONECCT52877.2021.9622557 (cit. on p. 14).

[69] Penghua Li, Zijian Zhang, Qingyu Xiong, Baocang Ding, Jie Hou, Dechao Luo, Yujun Rong, and Shuaiyong Li. «State-of-health estimation and remaining useful life prediction for the lithium-ion battery based on a variant long short term memory neural network». In: *Journal of Power Sources* 459 (2020), p. 228069. ISSN: 0378-7753. DOI: 10.1016/j.jpowsour.2020.228069 (cit. on p. 14).

[70] Lucian Ungurean, Mihai V. Micea, and Gabriel Cârstoiu. «Online state of health prediction method for lithium-ion batteries, based on gated recurrent unit neural networks». In: *International Journal of Energy Research* 44.8 (2020), pp. 6767–6777. DOI: 10.1002/er.5413 (cit. on p. 14).

[71] Prakash Venugopal and Vigneswaran T. «State-of-Health Estimation of Li-ion Batteries in Electric Vehicle Using IndRNN under Variable Load Condition». In: *Energies* 12.22 (2019). ISSN: 1996-1073. DOI: 10.3390/en12224338 (cit. on p. 14).

[72] Lukas Merkle, Michael Pöthig, and Florian Schmid. «Estimate e-Golf Battery State Using Diagnostic Data and a Digital Twin». In: *Batteries* 7.1 (2021). ISSN: 2313-0105. DOI: 10.3390/batteries7010015. URL: https://www.mdpi.com/2313-0105/7/1/15 (cit. on p. 14).

[73] Lingjun Song, Keyao Zhang, Tongyi Liang, Xuebing Han, and Yingjie Zhang. «Intelligent state of health estimation for lithium-ion battery pack based on big data analysis». In: *Journal of Energy Storage* 32 (2020), p. 101836. ISSN: 2352-152X. DOI: 10.1016/j.est.2020.101836 (cit. on pp. 14, 26).

[74] Oussama Attia, Makram Khalil, and Chokri Ben Salah. «Modeling, simulation and analysis of BEV and FCEV using Matlab/Simulink». In: *2022 IEEE 9th International Conference on Sciences of Electronics, Technologies of Information and Telecommunications (SETIT)*. 2022, pp. 212–217. DOI: 10.1109/SETIT54465.2022.9875536 (cit. on p. 14).

[75] Feyijimi Adegbohun, Annette von Jouanne, Ben Phillips, Emmanuel Agamloh, and Alex Yokochi. «High Performance Electric Vehicle Powertrain Modeling, Simulation and Validation». In: *Energies* 14.5 (2021). ISSN: 1996-1073. DOI: 10.3390/en14051493. URL: https://www.mdpi.com/1996-1073/14/5/1493 (cit. on p. 14).

[76] Tibor Kiss, Jason Lustbader, and Daniel Leighton. «Modeling of an Electric Vehicle Thermal Management System in MATLAB/Simulink». In: *SAE Technical Paper Series*. SAE International, Apr. 2015. DOI: 10.4271/2015-01-1708 (cit. on p. 14).

[77] A. A. Abulifa, R. K. Raja Ahmad, A. Che Soh, M. A. M. Radzi, and M. K. Hassan. «Modelling and simulation of battery electric vehicle by using MATLAB-Simulink». In: *2017 IEEE 15th Student Conference on Research and Development (SCOReD)*. 2017, pp. 383–387. DOI: 10.1109/SCORED.2017.8305360 (cit. on p. 14).

[78] MathWorks. *Mapped Motor Simulink block*. https://it.mathworks.com/help/autoblks/ug/electric-vehicle-reference-application.html. [Retrieved November 11, 2022.] 2022 (cit. on p. 14).

[79] MathWorks Student Competitions Team. *MATLAB and Simulink Racing Lounge: Vehicle Modeling*. https://it.mathworks.com/matlabcentral/fileexchange/63823-matlab-and-simulink-racing-lounge-vehicle-modeling. [Retrieved September 19, 2022.] 2022 (cit. on pp. 14, 52, 59, 70).

[80] Haskell B. Curry. «The method of steepest descent for non-linear minimization problems». In: *Quarterly of Applied Mathematics* 2.3 (1944), pp. 258–261. DOI: 10.1090/qam/10667 (cit. on p. 16).

[81] Jeremy Jordan. *Setting the learning rate of your neural network.* Mar. 2018. URL: https://www.jeremyjordan.me/nn-learning-rate/ (visited on Nov. 7, 2022) (cit. on p. 17).

[82] jason Brownlee. *Using Learning Rate Schedules for Deep Learning Models in Python with Keras*. June 2022. URL: https://machinelearningmastery.com/using-learning-rate-schedules-deep-learning-models-python-keras/ (visited on Nov. 7, 2022) (cit. on p. 16).

[83] Ayoosh Kathuria. *Intro to optimization in deep learning: Gradient Descent*. 2019. URL: https://blog.paperspace.com/intro-to-optimization-in-deep-learning-gradient-descent/ (visited on Nov. 7, 2022) (cit. on p. 18).

[84] Sebastian Ruder. *An overview of gradient descent optimization algorithms*. 2016. DOI: 10.48550/ARXIV.1609.04747 (cit. on p. 18).

[85] Andrew Ng. *Improving Deep Neural Networks: Hyperparameter Tuning, Regularization and Optimization*. URL: https://www.coursera.org/learn/deep-neural-network (visited on Nov. 7, 2022) (cit. on pp. 19, 20).

[86] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. «On the importance of initialization and momentum in deep learning». In: *Proceedings of the 30th International Conference on Machine Learning*. Ed. by Sanjoy Dasgupta and David McAllester. Vol. 28. Proceedings of Machine Learning Research 3. Atlanta, Georgia, USA: PMLR, June 2013, pp. 1139–1147. URL: https://proceedings.mlr.press/v28/sutskever13.html (cit. on p. 19).

[87] J. A. Nelder and R. Mead. «A Simplex Method for Function Minimization». In: *The Computer Journal* 7.4 (Jan. 1965), pp. 308–313. ISSN: 0010-4620. DOI: 10.1093/comjnl/7.4.308 (cit. on p. 20).

[88] Jeffrey C. Lagarias, James A. Reeds, Margaret H. Wright, and Paul E. Wright. «Convergence Properties of the Nelder–Mead Simplex Method in Low Dimensions». In: *SIAM Journal on Optimization* 9.1 (1998), pp. 112–147. DOI: 10.1137/S1052623496303470 (cit. on pp. 21, 22).

[89] Wikipedia contributors. *Nelder–Mead method — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Nelder%E2%80%93Mead_method&oldid=1114246182. [Online; accessed 7-November-2022]. 2022 (cit. on p. 22).

[90] Chang Wei Tan, Christoph Bergmeir, François Petitjean, and Geoffrey I. Webb. «Time series extrinsic regression». In: *Data Mining and Knowledge Discovery* 35.3 (May 2021), pp. 1032–1060. ISSN: 1573-756X. DOI: 10.1007/s10618-021-00745-9 (cit. on pp. 24, 25, 27, 95).

[91] Martin Längkvist, Lars Karlsson, and Amy Loutfi. «A review of unsupervised feature learning and deep learning for time-series modeling». In: *Pattern Recognition Letters* 42 (2014), pp. 11–24. ISSN: 0167-8655. DOI: 10.1016/j.patrec.2014.01.008 (cit. on pp. 25, 26).

[92] Friedrich von Bülow, Joshua Mentz, and Tobias Meisen. «State of health forecasting of Lithium-ion batteries applicable in real-world operational conditions». In: *Journal of Energy Storage* 44 (2021), p. 103439. ISSN: 2352-152X. DOI: 10.1016/j.est.2021.103439 (cit. on p. 26).

[93] Tingting Xu, Zhen Peng, and Lifeng Wu. «A novel data-driven method for predicting the circulating capacity of lithium-ion battery under random variable current». In: *Energy* 218 (2021), p. 119530. ISSN: 0360-5442. DOI: 10.1016/j.energy.2020.119530 (cit. on p. 26).

[94] Meru A. Patil, Piyush Tagade, Krishnan S. Hariharan, Subramanya M. Kolake, Taewon Song, Taejung Yeo, and Seokgwang Doo. «A novel multistage Support Vector Machine based approach for Li ion battery remaining useful life estimation». In: *Applied Energy* 159 (2015), pp. 285–297. ISSN: 0306-2619. DOI: 10.1016/j.apenergy.2015.08.119 (cit. on p. 26).

[95] Angelo Bonfitto, Ethelbert Ezemobi, Nicola Amati, Stefano Feraco, Andrea Tonoli, and Shailesh Hegde. «State of Health Estimation of Lithium Batteries for Automotive Applications with Artificial Neural Networks». In: *2019 AEIT International Conference of Electrical and Electronic Technologies for Automotive (AEIT AUTOMOTIVE)*. 2019, pp. 1–5. DOI: 10.23919/EETA.2019.8804567 (cit. on p. 26).

[96] Yitao Wu, Qiao Xue, Jiangwei Shen, Zhenzhen Lei, Zheng Chen, and Yonggang Liu. «State of Health Estimation for Lithium-Ion Batteries Based on Healthy Features and Long Short-Term Memory». In: *IEEE Access* 8 (2020), pp. 28533–28547. DOI: 10.1109/ACCESS.2020.2972344 (cit. on p. 26).

[97] Limei Wang, Chaofeng Pan, Liang Liu, Yong Cheng, and Xiuliang Zhao. «On-board state of health estimation of LiFePO4 battery pack through differential voltage analysis». In: *Applied Energy* 168 (2016), pp. 465–472. ISSN: 0306-2619. DOI: 10.1016/j.apenergy.2016.01.125 (cit. on p. 26).

[98] Gae-won You, Sangdo Park, and Dukjin Oh. «Real-time state-of-health estimation for electric vehicle batteries: A data-driven approach». In: *Applied Energy* 176 (2016), pp. 92–103. ISSN: 0306-2619. DOI: 10.1016/j.apenergy.2016.05.051 (cit. on p. 26).

[99] Xiaopeng Tang, Changfu Zou, Ke Yao, Guohua Chen, Boyang Liu, Zhenwei He, and Furong Gao. «A fast estimation algorithm for lithium-ion battery state of health». In: *Journal of Power Sources* 396 (2018), pp. 453–458. ISSN: 0378-7753. DOI: https://doi.org/10.1016/j.jpowsour.2018.06.036 (cit. on p. 26).

[100] Caihao Weng, Yujia Cui, Jing Sun, and Huei Peng. «On-board state of health monitoring of lithium-ion batteries using incremental capacity analysis with support vector regression». In: *Journal of Power Sources* 235 (2013), pp. 36–44. ISSN: 0378-7753. DOI: 10.1016/j.jpowsour.2013.02.012 (cit. on p. 26).

[101] Alberto Barragán Moreno. «Machine Learning-based Online State-of-Health Estimation of Electric Vehicle Batteries». MA thesis. Denmark: Aalborg University, 2021 (cit. on p. 26).

[102] Sheng Shen, Mohammadkazem Sadoughi, Meng Li, Zhengdao Wang, and Chao Hu. «Deep convolutional neural networks with ensemble learning and transfer learning for capacity estimation of lithium-ion batteries». In: *Applied Energy* 260 (2020), p. 114296. ISSN: 0306-2619. DOI: 10.1016/j.apenergy.2019.114296 (cit. on p. 26).

[103] Jungsoo Kim, Jungwook Yu, Minho Kim, Kwangrae Kim, and Soohee Han. «Estimation of Li-ion Battery State of Health based on Multilayer Perceptron: as an EV Application». In: *IFAC-PapersOnLine* 51.28 (2018). 10th IFAC Symposium on Control of Power and Energy Systems CPES 2018, pp. 392–397. ISSN: 2405-8963. DOI: 10.1016/j.ifacol.2018.11.734 (cit. on p. 26).

[104] Yang Li and Jili Tao. «CNN and transfer learning based online SOH estimation for lithium-ion battery». In: *2020 Chinese Control And Decision Conference (CCDC)*. 2020, pp. 5489–5494. DOI: 10.1109/CCDC49329.2020.9164208 (cit. on p. 26).

[105] Angus Dempster, François Petitjean, and Geoffrey I. Webb. «ROCKET: exceptionally fast and accurate time series classification using random convolutional kernels». In: *Data Mining and Knowledge Discovery* 34.5 (June 2020), pp. 1454–1495. DOI: 10.1007/s10618-020-00701-z (cit. on p. 26).

[106] Angus Dempster, Daniel F. Schmidt, and Geoffrey I. Webb. «MiniRocket: A Very Fast (Almost) Deterministic Transform for Time Series Classification». In: *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. ACM, Aug. 2021. DOI: 10.1145/3447548.3467231 (cit. on pp. 26, 30).

[107] Ali Raza. *Type of convolutions: Deformable and Transformable Convolution*. Towards Data Science. July 2019. URL: https://towardsdatascience.com/type-of-convolutions-deformable-and-transformable-convolution-1f660571eb91 (visited on Oct. 24, 2022) (cit. on p. 28).

[108] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016 (cit. on p. 27).

[109] Vincent Dumoulin and Francesco Visin. *A guide to convolution arithmetic for deep learning*. 2016. DOI: 10.48550/ARXIV.1603.07285 (cit. on p. 28).

[110] Wikipedia contributors. *Reciprocal distribution — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Reciprocal_distribution&oldid=1097983132. [Online; accessed 28-October-2022]. 2022 (cit. on p. 30).

[111] Laurens van der Maaten, Eric Postma, and Jaap van den Herik. «Dimensionality reduction: a comparative review». In: *J Mach Learn Res* 10 (2009), pp. 66–71 (cit. on p. 31).

[112] Jonathon Shlens. *A Tutorial on Principal Component Analysis*. 2014. DOI: 10.48550/ARXIV.1404.1100 (cit. on p. 31).

[113] Amir Ali. *Dimensionality Reduction (PCA and LDA) with Practical Implementation*. Mar. 2019. URL: https://medium.com/machine-learning-researcher/dimensionality-reduction-pca-and-lda-6be91734f567 (visited on Nov. 3, 2022) (cit. on p. 33).

[114] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. New York, NY, USA: Springer New York Inc., 2001. DOI: 10.1007/978-0-387-84858-7 (cit. on pp. 35, 37, 40, 41, 44).

[115] Joseph W. McKean. «Robust Analysis of Linear Models». In: *Statistical Science* 19.4 (2004), pp. 562–570. DOI: 10.1214/088342304000000549 (cit. on pp. 36, 37).

[116] Henri Theil. «A Rank-Invariant Method of Linear and Polynomial Regression Analysis». In: *Henri Theil's Contributions to Economics and Econometrics: Econometric Theory and Methodology*. Ed. by Baldev Raj and Johan Koerts. Dordrecht: Springer Netherlands, 1992, pp. 345–381. ISBN: 978-94-011-2546-8. DOI: 10.1007/978-94-011-2546-8_20 (cit. on pp. 37, 38).

[117] Pranab Kumar Sen. «Estimates of the Regression Coefficient Based on Kendall's Tau». In: *Journal of the American Statistical Association* 63.324 (1968), pp. 1379–1389. DOI: 10.1080/01621459.1968.10480934 (cit. on pp. 37, 38).

[118] Xin Dang, Hanxiang Peng, Xueqin Wang, and Heping Zhang. «Theil-Sen Estimators in a Multiple Linear Regression Model». In: 2009 (cit. on pp. 37, 38).

[119] Scikit-learn docs. *TheilSenRegressor*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.TheilSenRegressor.html (visited on Nov. 5, 2022) (cit. on p. 38).

[120] Xueqin Wang and Qiqing Yu. «Unbiasedness of the Theil–Sen estimator». In: *Journal of Nonparametric Statistics* 17.6 (2005), pp. 685–695. DOI: 10.1080/10485250500039452 (cit. on p. 38).

[121] Rand Wilcox. «A Note on the Theil-Sen Regression Estimator When the Regressor Is Random and the Error Term Is Heteroscedastic». In: *Biometrical Journal* 40.3 (1998), pp. 261–268. DOI: 10.1002/(SICI)1521-4036(199807)40:3<261::AID-BIMJ261>3.0.CO;2-V (cit. on p. 38).

[122] Jason Brownlee. *Robust Regression for Machine Learning in Python*. Oct. 2020. URL: https://machinelearningmastery.com/robust-regression-for-machine-learning-in-python/ (visited on Nov. 5, 2022) (cit. on p. 39).

[123] Probal Chaudhuri. «On a Geometric Notion of Quantiles for Multivariate Data». In: *Journal of the American Statistical Association* 91.434 (1996), pp. 862–872. ISSN: 01621459. URL: http://www.jstor.org/stable/2291681 (visited on Nov. 5, 2022) (cit. on pp. 39, 101).

[124] *Ridge regression*. Feb. 2020. URL: https://machinelearningjourney.com/index.php/2020/02/13/ridge-regression/ (visited on Nov. 5, 2022) (cit. on p. 41).

[125] Arthur E. Hoerl and Robert W. Kennard. «Ridge Regression: Biased Estimation for Nonorthogonal Problems». In: *Technometrics* 12.1 (1970), pp. 55–67. DOI: 10.1080/00401706.1970.10488634 (cit. on p. 40).

[126] Tin Kam Ho. «Random decision forests». In: *Proceedings of 3rd International Conference on Document Analysis and Recognition*. Vol. 1. 1995, 278–282 vol.1. DOI: 10.1109/ICDAR.1995.598994 (cit. on p. 41).

[127] *Trajan* (user). *Ridge regression*. Oct. 2020. URL: https://stats.stackexchange.com/questions/490185/decision-tree-regression-and-splits (visited on Nov. 5, 2022) (cit. on p. 42).

[128] Leo Breiman. *Classification and Regression Trees (1st ed.)* Routledge, 1984. DOI: 10.1201/9781315139470 (cit. on p. 42).

[129] Scikit-learn docs. *RandomForestRegressor*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html (visited on Nov. 5, 2022) (cit. on p. 43).

[130] Leo Breiman. «Bagging predictors». In: *Machine Learning* 24.2 (Aug. 1996), pp. 123–140. ISSN: 1573-0565. DOI: 10.1007/BF00058655 (cit. on p. 43).

[131] Tin Kam Ho. «The random subspace method for constructing decision forests». In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20.8 (1998), pp. 832–844. DOI: 10.1109/34.709601 (cit. on p. 43).

[132] Antanas Verikas, Evaldas Vaiciukynas, Adas Gelzinis, James Parker, and M Charlotte Olsson. «Electromyographic Patterns during Golf Swing: Activation Sequence Profiling and Prediction of Shot Effectiveness». In: *Sensors (Basel)* 16.4 (Apr. 2016) (cit. on p. 44).

[133] CS231n: Convolutional Neural Networks for Visual Recognition. *Neural Networks 1*. URL: https://cs231n.github.io/neural-networks-1/ (visited on Nov. 6, 2022) (cit. on pp. 47, 48).

[134] Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. «Automatic differentiation in machine learning: a survey». In: (2015). DOI: 10.48550/ARXIV.1502.05767 (cit. on p. 46).

[135] Wikipedia contributors. *Vanishing gradient problem — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Vanishing_gradient_problem&oldid=1119053971. [Online; accessed 6-November-2022]. 2022 (cit. on p. 47).

[136] Vinod Nair and Geoffrey E. Hinton. «Rectified Linear Units Improve Restricted Boltzmann Machines». In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*. ICML'10. Haifa, Israel: Omnipress, 2010, pp. 807–814. ISBN: 9781605589077 (cit. on p. 47).

[137] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*. 2015. DOI: 10.48550/ARXIV.1502.01852 (cit. on p. 47).

[138] Timo Stöttner. *Why Data should be Normalized before Training a Neural Network*. May 2019. URL: https://towardsdatascience.com/why-data-should-be-normalized-before-training-a-neural-network-c626b7f66c7d (visited on Nov. 7, 2022) (cit. on p. 47).

[139] Alexei Botchkarev. «A New Typology Design of Performance Metrics to Measure Errors in Machine Learning Regression Algorithms». In: *Interdisciplinary Journal of Information, Knowledge, and Management* 14 (2019), pp. 045–076. DOI: 10.28945/4184 (cit. on p. 48).

[140] Xinmei Yuan, Chuanpu Zhang, Guokai Hong, Xueqi Huang, and Lili Li. «Method for evaluating the real-world driving energy consumptions of electric vehicles». In: *Energy* 141 (2017), pp. 1955–1968. ISSN: 0360-5442. DOI: https://doi.org/10.1016/j.energy.2017.11.134 (cit. on p. 51).

[141] Edis Osmanbasic. *EV powertrain components*. Feb. 2021. URL: https://www.renaultgroup.com/en/news-on-air/news/learn-all-you-need-to-know-about-the-motor-of-an-electric-car/ (visited on Sept. 21, 2022) (cit. on p. 52).

[142] EVreporter. *EV powertrain components*. Jan. 2020. URL: https://evreporter.com/ev-powertrain-components/ (visited on Sept. 21, 2022) (cit. on p. 53).

[143] John G. Hayes and G. Abas Goodarzi. *Electric Powertrain: energy systems, power electronics and drives for hybrid, electric and fuel cell vehicles*. Wiley, 2018. DOI: 10.1002/9781119063681.ch5 (cit. on pp. 52, 54, 55).

[144] MathWorks. *Mapped Motor Simulink block*. https://it.mathworks.com/help/autoblks/ref/mappedmotor.html. [Retrieved September 21, 2022.] 2022 (cit. on p. 53).

[145] Chen Lv, Hong Wang, and Dongpu Cao. «Chapter 8 - Brake-Blending Control of EVs». In: *Modeling, Dynamics and Control of Electrified Vehicles*. Ed. by Hui Zhang, Dongpu Cao, and Haiping Du. Woodhead Publishing, 2018, pp. 275–308. ISBN: 978-0-12-812786-5. DOI: 10.1016/B978-0-12-812786-5.00008-2 (cit. on p. 55).

[146] Poria Fajri, Shoeib Heydari, and Nima Lotfi. «Optimum low speed control of regenerative braking for electric vehicles». In: *2017 IEEE 6th International Conference on Renewable Energy Research and Applications (ICRERA)*. 2017, pp. 875–879. DOI: 10.1109/ICRERA.2017.8191185 (cit. on p. 55).

[147] Wikipedia contributors. *Regenerative braking — Wikipedia, The Free Encyclopedia*. [Online; accessed 20-September-2022]. 2022. URL: https://en.wikipedia.org/wiki/Regenerative_braking (cit. on p. 55).

[148] Wikipedia contributors. *Disc brake — Wikipedia, The Free Encyclopedia*. [Online; accessed 20-September-2022]. 2022. URL: https://en.wikipedia.org/wiki/Disc_brake (cit. on p. 57).

[149] Carlos M. Gonzalez. *What's the Difference Between Friction and Regenerative Car Brakes?* Mar. 2016. URL: https://www.machinedesign.com/mechanical-motion-systems/article/21831982/whats-the-difference-between-friction-and-regenerative-car-brakes (visited on Sept. 21, 2022) (cit. on p. 57).

[150] Bob Sorokanich. *Why Don't Electric Cars Have Multi-Gear Transmissions?* Aug. 2017. URL: https://www.roadandtrack.com/new-cars/car-technology/a12019034/why-dont-electric-cars-have-multi-gear-transmissions/ (visited on Sept. 21, 2022) (cit. on p. 58).

[151] Mehrdad Ehsani, Yimin Gao, Stefano Longo, and Kambiz Ebrahimi. *Modern Electric, Hybrid Electric, and Fuel Cell Vehicles, third edition*. CRC Press, Feb. 2018. DOI: 10.1201/9780429504884 (cit. on p. 58).

[152] Erik Cheever. *Gears and Systems with both Rotation and Translation*. URL: https://lpsa.swarthmore.edu/Systems/MechRotating/RotMechSysGears.html#Gears (visited on Sept. 21, 2022) (cit. on p. 58).

[153] Jinglai Wu, Jiejunyi Liang, Jiageng Ruan, Nong Zhang, and Paul D. Walker. «Efficiency comparison of electric vehicles powertrains with dual motor and single motor input». In: *Mechanism and Machine Theory* 128 (2018), pp. 569–585. DOI: 10.1016/j.mechmachtheory.2018.07.003 (cit. on p. 59).

[154] MathWorks. *Gearbox Simulink block*. https://it.mathworks.com/help/autoblks/ref/gearbox.html. [Retrieved September 22, 2022.] 2022 (cit. on p. 59).

[155] MathWorks. *Torsional Compliance Simulink block*. https://it.mathworks.com/help/autoblks/ref/torsionalcompliance.html. [Retrieved September 22, 2022.] 2022 (cit. on p. 59).

[156] MathWorks. *Open Differential Simulink block*. https://it.mathworks.com/help/autoblks/ref/opendifferential.html. [Retrieved September 22, 2022.] 2022 (cit. on p. 59).

[157] MathWorks. *Longitudinal Wheel*. https://it.mathworks.com/help/autoblks/ref/longitudinalwheel.html. [Retrieved September 22, 2022.] 2022 (cit. on p. 60).

[158] Egbert Bakker, Lars Nyborg, and Hans B. Pacejka. «Tyre Modelling for Use in Vehicle Dynamics Studies». In: *SAE International Congress and Exposition*. SAE International, Feb. 1987. DOI: 10.4271/870421 (cit. on p. 60).

[159] Hans Pacejka. *Tire and Vehicle Dynamics*. 3rd ed. Butterworth-Heinemann, Apr. 2012 (cit. on p. 61).

[160] MathWorks. *Vehicle body 1DOF Longitudinal*. https://it.mathworks.com/help/autoblks/ref/vehiclebody1doflongitudinal.html. [Retrieved September 24, 2022.] 2022 (cit. on pp. 61, 62).

[161] MathWorks. *Generic battery model*. https://it.mathworks.com/help/sps/powersys/ref/battery.html. [Retrieved September 26, 2022.] 2022 (cit. on pp. 62, 63, 67, 71).

[162] Lluc Canals Casals, Marta Rodríguez, Cristina Corchero, and Rafael E. Carrillo. «Evaluation of the End-of-Life of Electric Vehicle Batteries According to the State-of-Health». In: *World Electric Vehicle Journal* 10.4 (2019). ISSN: 2032-6653. DOI: 10.3390/wevj10040063. URL: https://www.mdpi.com/2032-6653/10/4/63 (cit. on p. 65).

[163] Ali Zenati, Philippe Desprez, Hubert Razik, and Stéphane Rael. «A methodology to assess the State of Health of lithium-ion batteries based on the battery's parameters and a Fuzzy Logic System». In: *2012 IEEE International Electric Vehicle Conference*. 2012, pp. 1–6. DOI: 10.1109/IEVC.2012.6183268 (cit. on p. 65).

[164] Bolun Xu, Alexandre Oudalov, Andreas Ulbig, Göran Andersson, and Daniel S. Kirschen. «Modeling of Lithium-Ion Battery Degradation for Cell Life Assessment». In: *IEEE Transactions on Smart Grid* 9.2 (2018), pp. 1131–1140. DOI: 10.1109/TSG.2016.2578950 (cit. on p. 66).

[165] Ilyès Miri, Abbas Fotouhi, and Nathan Ewin. «Electric vehicle energy consumption modelling and estimation—A case study». In: *International Journal of Energy Research* 45.1 (July 2020), pp. 501–520. DOI: 10.1002/er.5700 (cit. on p. 66).

[166] Volkswagen. *Electric and hybrid driving modes*. 2022. URL: https://www.volkswagen.co.uk/en/electric-and-hybrid/software-and-technology/driving-technology/electric-driving-modes.html (visited on Oct. 10, 2022) (cit. on p. 66).

[167] Ze Perfs. *Volkswagen e-up! (2017-2020) technical specifications and performance figures*. Jan. 2021. URL: https://zeperfs.com/it/fiche8711-vw-up-e-up.htm (visited on Oct. 10, 2022) (cit. on p. 70).

[168] Ze Perfs. *Volkswagen e-Golf (2020) technical specifications and performance figures*. Aug. 2020. URL: https://zeperfs.com/en/fiche6939-vw-golf-vii-e-golf.htm (visited on Oct. 10, 2022) (cit. on p. 70).

[169] Yaşar Demirel. «5.2 Energy Conservation». In: *Comprehensive Energy Systems*. Ed. by Ibrahim Dincer. Oxford: Elsevier, 2018, pp. 45–90. ISBN: 978-0-12-814925-6. DOI: 10.1016/B978-0-12-809597-3.00505-8 (cit. on p. 70).

[170] EVTUN.COM. *Volkswagen e-Golf range extender and regenerative force improvement*. URL: https://evtun.com/ev-chips-194/volkswagen-e-golf-range-extender.html (visited on Oct. 10, 2022) (cit. on p. 70).

[171] Physics Forums. *How much (roughly) brake torque can be produced by a regular car*. URL: https://www.physicsforums.com/threads/how-much-roughly-brake-torque-can-be-produced-by-a-regular-car.577591/ (visited on Oct. 10, 2022) (cit. on p. 70).

[172] Goonet Exchange. *VW e-up! technical specifications*. URL: https://www.goo-net-exchange.com/catalog/VOLKSWAGEN__EUP/10093510/ (visited on Oct. 10, 2022) (cit. on p. 70).

[173] Volkswagen Newspress. *2017 e-Golf technical specifications*. URL: https://newspress-vwusamedia.s3.amazonaws.com/documents%2Foriginal%2F6456-18417786758fdfc77c7df6.pdf (visited on Oct. 10, 2022) (cit. on pp. 70, 71).

[174] Xinmei Yuan, Chuanpu Zhang, Guokai Hong, Xueqi Huang, and Lili Li. «Method for evaluating the real-world driving energy consumptions of electric vehicles». In: *Energy* 141 (2017), pp. 1955–1968. ISSN: 0360-5442. DOI: 10.1016/j.energy.2017.11.134 (cit. on p. 70).

[175] Anthony Stark. *How to calculate wheel radius*. URL: https://x-engineer.org/calculate-wheel-radius/ (visited on Oct. 10, 2022) (cit. on p. 70).

[176] Volkswagen Newsroom. *up! four-door – up! exterior*. URL: https://www.volkswagen-newsroom.com/en/up-four-door-driving-presentation-3073/up-four-door-up-exterior-3092 (visited on Oct. 10, 2022) (cit. on p. 70).

[177] Tesla Bjørn Test Procedure. *VW e-up TBTP*. URL: https://tbtp-ev.github.io/vw-e-up.html (visited on Oct. 11, 2022) (cit. on p. 70).

[178] Dominik Czernia. *Car center of mass calculator*. URL: https://www.omnicalculator.com/physics/car-mass-center (visited on Oct. 11, 2022) (cit. on p. 70).

[179] Tesla Bjørn Test Procedure. *VW e-Golf 36 kWh TBTP*. URL: https://tbtp-ev.github.io/vw-e-golf.html (visited on Oct. 11, 2022) (cit. on p. 70).

[180] EVSpecifications. *2016 e-up! technical specifications*. URL: https://www.evspecifications.com/en/model/c0467a (visited on Oct. 10, 2022) (cit. on pp. 70, 71).

[181] Inside EVs. *The e-Golf Is Dead Soon: Long Live The VW Golf VIII GTE And The PHEV*. URL: https://insideevs.com/news/378473/egolf-dead-golf-viii-phev/ (visited on Oct. 10, 2022) (cit. on p. 70).

[182] Kevin Davis and John G. Hayes. «Comparison of Lithium-Ion Battery Pack Models Based on Test Data from Idaho and Argonne National Laboratories». In: *2020 IEEE Energy Conversion Congress and Exposition (ECCE)*. 2020, pp. 5626–5632. DOI: 10.1109/ECCE44975.2020.9236373 (cit. on p. 71).

[183] Green Car Congress. *Volkswagen's first two production battery-electric vehicles debut at Frankfurt*. URL: https://www.greencarcongress.com/2013/09/20130911-vw.html (visited on Oct. 11, 2022) (cit. on p. 71).

[184] Maximilian Fichtner. «Recent Research and Progress in Batteries for Electric Vehicles». In: *Batteries & Supercaps* 5.2 (2022), e202100224. DOI: 10.1002/batt.202100224 (cit. on p. 71).

[185] MathWorks. *Parameter Estimation*. https://it.mathworks.com/help/sldo/parameter-estimation.html. [Retrieved October 11, 2022.] 2022 (cit. on p. 72).

[186] Tim Barlow, Stephen Latham, Ian S. McCrae, and Paul Boulter. «A reference book of driving cycles for use in the measurement of road vehicle emissions». In: *TRL Published Project Report* (2009) (cit. on p. 78).

[187] Brian Kenji Iwana and Seiichi Uchida. «An empirical survey of data augmentation for time series classification with neural networks». In: *PLOS ONE* 16.7 (July 2021). Ed. by Friedhelm Schwenker, e0254841. DOI: 10.1371/journal.pone.0254841 (cit. on p. 94).

[188] Hassan Ismail Fawaz et al. «InceptionTime: Finding AlexNet for time series classification». In: *Data Mining and Knowledge Discovery* 34.6 (Nov. 2020), pp. 1936–1962. ISSN: 1573-756X. DOI: 10.1007/s10618-020-00710-y (cit. on p. 95).

[189] Karl Weiss, Taghi M. Khoshgoftaar, and DingDing Wang. «A survey of transfer learning». In: *Journal of Big Data* 3.1 (May 2016), p. 9. ISSN: 2196-1115. DOI: 10.1186/s40537-016-0043-6 (cit. on p. 95).