



Data-Driven State Estimation Algorithms for a 48V Battery Management System

Pedro Gabriel Araújo Monteiro

WORKING VERSION

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

External Supervisor: Doctor Cláudio Filipe Azevedo Pinto

FEUP Supervisor: Professor Rui Manuel Esteves Araújo

July 29, 2021

Resumo

A estimação do estado de carga, estado de saúde e parâmetros de uma célula de lítio são um desafio complexo para designers de sistemas de gestão de baterias, devido ao seu comportamento não linear a diferentes condições de operação e níveis de envelhecimento. Como uma solução possível, múltiplas soluções inteligência artificial têm sido propostas nestes últimos anos. Estes métodos apresentam vantagens sobre os mais comuns métodos baseados num modelo da bateria, pois não requerem um conhecimento extenso e estudo prévio do comportamento interno de uma célula de lítio. Porém, muitas destas soluções inteligência artificial não poderiam ser consideradas devido a sua complexidade. O número elevado de parâmetros necessários e/ou elevado consumo memória durante estimação dificultariam a aplicação destes métodos. Portanto, nesta dissertação, modelos baseados em Feedforward Neural Networks (FNN) são propostos para estimação online do estado de carga e estado de saúde, com um método eficiente para pré-processamento dos inputs do modelo e com um baixo número parâmetros necessários, usando apenas medições online de tensão, corrente e temperatura durante o seu funcionamento normal. Para estimação dos parâmetros da célula, um algoritmo online Recursive Least Squares with Forgetting Factor (RLS-FF) é usado, resultando em baixo erro estimação tensão. Como um método para poder validar de forma prática as soluções propostas, um setup de teste foi desenvolvido que permitisse emular o comportamento tensão e corrente de uma bateria lítio 48V. Os algoritmos propostos foram então testados e validados, usando dados adquiridos de datasets públicos online, e com dados experimentais disponibilizados pela empresa Continental Engineering Solutions.

Abstract

Li-ion battery State-of-Charge (SOC), State-of-Health (SOH) and cell parameter estimation are a complex challenge for battery management systems designers, due to the battery's non-linear behaviour at different operating conditions and ageing levels. As a possible solution, multiple machine learning models have been proposed for states estimation throughout the years. These provide an advantage over more common model based methods, as they do not require a deep knowledge and study of the battery's internal behaviour. However, many of these proposed models could not be considered due to their complexity. The high number of required stored parameters and/or elevated memory consumption during estimation may pose challenges to the application of these methods. Therefore, in this thesis, Feedforward Neural Network (FNN) models are proposed for online SOC and SOH estimation, with an efficient method for online input preprocessing and low parameter requirement in storage, using solely voltage, current and temperature measurements taken during normal operation of the Battery Management System (BMS). For cell parameter estimation, an online Recursive Least Squares with Forgetting Factor (RLS-FF) algorithm is used, resulting in low online voltage prediction error. As a practical validation method, a hardware setup was designed and developed to emulate the voltage and current behaviour of a Li-ion 48V battery pack. The proposed algorithms were then tested and validated, using publicly available online battery datasets, and experimental data collected and made available by Continental Engineering Solutions.

Acknowledgements

Many were the people that, in their own way, contributed and have to sincerely thank for the positive outcome of my dissertation.

In a direct contribution to the final outcome of the thesis, I have to thank my supervisors Professor Rui Esteves Araújo and Doctor Cláudio Pinto, for all the attention and suggestions given during the developing phase of the work done in this thesis and on this final document.

In an indirect contribution to the final outcome of the thesis, I have to thank my parents, brothers, friends, and my girlfriend Marina Afonso, for all the support given to me during the development of this dissertation.

Pedro Gabriel Araújo Monteiro

“Compassion is the basis of morality.”

Arthur Schopenhauer

Contents

1	Motivation	1
1.1	Context	1
1.2	The need for battery management systems	1
1.3	Goals of the thesis	2
1.4	Document structure	3
2	State of the Art	5
2.1	Structure and main functions	5
2.2	Balancing topologies overview	11
2.2.1	Passive topologies	11
2.2.2	Active topologies	12
2.2.3	Balancing topologies summary	19
2.3	Li-ion battery models	21
2.4	State estimation algorithms	25
2.4.1	State-of-Charge algorithms	25
2.4.2	State-of-Health algorithms	42
2.4.3	Power prediction	46
2.5	Online cell parameterization strategies	48
3	Algorithms and Implementation	53
3.1	Feedforward Neural Networks	53
3.1.1	Forward propagation	53
3.1.2	Gradient descent - Supervised Network Learning	54
3.1.3	Activation functions	61
3.2	State-of-Charge estimation	62
3.2.1	Proposed Model	63
3.3	State-of-Health estimation	65
3.3.1	Proposed Model	65
3.4	Cell model parameter estimation	69
3.4.1	Proposed Algorithm	69
4	Results and Discussion	73
4.1	State-of-Charge estimation results	73
4.1.1	Model simulation results	73
4.1.2	Experimental results	78
4.2	State-of-Health estimation results	82
4.3	Cell model parameter estimation results	86

5	HW setup designed	91
5.1	Cell voltage emulation	92
5.1.1	Experimental results	96
5.2	Cell current emulation	99
5.2.1	Experimental results	101
6	Final Remarks and future work	103
6.1	Future work	104
	References	107
A	Developed training functions	121

List of Figures

1.1	Lithium-ion cell operating window, image taken from [9].	2
2.1	EV BMS general structure.	6
2.2	Distributed BMS structure.	7
2.3	EV BMS Components and Functions.	8
2.4	Balancing topologies.	11
2.5	Fixed shunting resistor.	12
2.6	Switched shunting resistor.	12
2.7	Complete shunting method.	13
2.8	Switched capacitor method.	13
2.9	Double switched capacitor method.	13
2.10	Cûk converter method.	14
2.11	PWM converter method.	14
2.12	Shunt inductor method.	15
2.13	Boost shunting method.	15
2.14	Cell to pack multiple transformers method.	16
2.15	Cell to pack switched transformer method.	16
2.16	Voltage multiplier method.	17
2.17	Pack to cell multiple transformers method.	17
2.18	Pack to cell switched transformer method.	17
2.19	Cell to pack to cell PWM converter method.	18
2.20	Cell to pack to cell single switched capacitor method.	18
2.21	Bidirectional multiple transformers method.	19
2.22	Bidirectional single switched transformer method.	19
2.23	Battery models reviewed in this work.	21
2.24	Rint model.	22
2.25	Randles model.	23
2.26	nRC model.	24
2.27	Example of a fractional order model.	25
2.28	SOC methods reviewed in this work.	25
2.29	Typical OCV-SOC curve in LiFePO ₄ cells, image from [28].	27
2.30	Typical OCV-SOC curve in NMC cells, image from [29].	27
2.31	PI observer structure, image from [32].	28
2.32	Dual-EKF structure for state and model parameters estimation, image from [40].	30
2.33	Feedforward neural network structure.	32
2.34	Recurrent neural network structure.	35
2.35	DDRN based SOC and SOH estimation method, image from [60].	36
2.36	LSTM layer unit.	37

2.37	BiLSTM structure and function, unfolded in time.	38
2.38	GRU layer unit.	39
2.39	SOH methods reviewed in this work.	42
2.40	Voltage and current waveform when discharge/charge pulse is applied [68].	43
2.41	ICA curve [73].	44
2.42	DVA curve [74].	44
2.43	SBPM model establishment schematic, image from [80].	46
2.44	SOP estimation block schematic, adapted from [86].	48
2.45	RLS diagram, adapted from [93].	51
3.1	Neuron model.	53
3.2	FNN general model used	63
3.3	Voltage waveforms during charging procedure.	66
3.4	Battery SOH measured at each cycle.	67
3.5	1RC pair model used.	69
4.1	Comparison validation results and training methods - one layer models.	75
4.2	Comparison validation results and training methods - two layer models.	76
4.3	Comparison validation results and training methods.	77
4.4	Model 25-25 neurons - Validation results at 25°C.	78
4.5	Training cycle 1 - Voltage, current and SOC data.	79
4.6	Training cycle 2 - Voltage, current and SOC data.	79
4.7	Validation cycle - Voltage, current and SOC data.	80
4.8	SOC prediction results - Validation cycle.	81
4.9	SOH results - Cell #5 multiple models prediction results.	83
4.10	SOH results - Cell #6 multiple models prediction results.	84
4.11	SOH results - Cell #7 multiple models prediction results.	84
4.12	SOH results - Cell #18 multiple models prediction results.	85
4.13	Voltage prediction results - UDDS Cycle at 25°C.	87
4.14	Parameter prediction results - UDDS Cycle at 25°C.	87
4.15	Voltage prediction results - US06 Cycle at 25°C.	88
4.16	Parameter prediction results - US06 Cycle at 25°C.	88
5.1	HW setup overview.	91
5.2	Circuit for single cell voltage V_c emulation.	92
5.3	Circuit when transistor is closed.	92
5.4	Circuit when transistor is open.	92
5.5	Schematic for cell voltage generation pcbs.	94
5.6	PCB for cell voltage generation.	95
5.7	Steady 4V cell output.	96
5.8	Rise from 2.5V to 4.2V cell output.	97
5.9	Multiple cell voltage output test.	98
5.10	Discharge cycle voltage emulated on a HW setup cell.	98
5.11	Differential voltage generation.	99
5.12	Schematic for differential voltage generation pcb.	100
5.13	PCB for differential voltage generation.	100
5.14	Differential voltage generation results.	101
A.1	Forward propagation function - page (1/2).	122

A.2	Forward propagation function - page (2/2).	123
A.3	Backward propagation function - page (1/2).	124
A.4	Backward propagation function - page (2/2).	125
A.5	SGD parameters update function.	126
A.6	SGD algorithm training function - page (1/2).	127
A.7	SGD algorithm training function - page (2/2).	128
A.8	SGDM parameters update function.	129
A.9	SGDM algorithm training function - page (1/2).	130
A.10	SGDM algorithm training function - page (2/2).	131
A.11	RMSPROP parameters update function.	132
A.12	RMSPROP algorithm training function - page (1/2).	133
A.13	RMSPROP algorithm training function - page (2/2).	134
A.14	ADAM parameters update function.	135
A.15	ADAM algorithm training function - page (1/2).	136
A.16	ADAM algorithm training function - page (2/2).	137

List of Tables

2.1	Cell balancing methods overview.	20
2.2	Comparison of different machine learning methods reviewed [24, 52].	41
3.1	NASA Battery dataset description.	68
4.1	Minimum total SOC validation error achieved.	74
4.2	SOC validation error at different temperatures.	74
4.3	Minimum total SOH validation error achieved for cell #5.	82
4.4	Minimum total SOH validation error achieved for cell #6.	83
4.5	Minimum total SOH validation error achieved for cell #7.	83
4.6	Minimum total SOH validation error achieved for cell #18.	85
4.7	RLS-FF Model voltage prediction error for multiple temperatures.	86

Abbreviations and Symbols

ADAM	Adaptive Moment Estimation
AEKF	Adaptive Extended Kalman Filter
ASGSMO	Adaptive Switching Gain Sliding Mode Observer
AUKF	Adaptive Unscented Kalman Filter
BMS	Battery Management System
BiLSTM	Bidirectional Long Short-Term Memory
BJDST	Beijing Dynamic Stress Test
CAN	Controller Area Network
CC-CV	Constant Current - Constant Voltage
CES	Continental Engineering Services
CPE	Constant Phase Element
CV	Constant Voltage
DCM	Discontinuous Current Mode
DDRN	Dynamically Driven Recurrent Neural network
DOD	Depth-of-Discharge
DST	Dynamic Stress Test
DVA	Differential Voltage Analysis
EIS	Electrochemical Impedance Spectroscopy
EKF	Extended Kalman Filter
ELM	Extreme Learning Machine
EMS	Energy Management System
EV	Electrical Vehicle
FNN	Feedforward Neural Network
FUDS	Federal Urban Driving Schedule
GPS	Global Positioning System
GSA	Gravitational Search Algorithm
GRU	Gated Recurrent Unit
HEV	Hybrid Electrical Vehicle
HPPC	Hybrid Pulse Power Characterization
HWFET	Highway Fuel Economy Driving Schedule
IC	Incremental Capacity
ICA	Incremental Capacity Analysis
ICs	Integrated Circuits
KF	Kalman Filter
Li-ion	Lithium Ion
LiFePO ₄	Lithium Ion Phosphate
LSA	Lighting Search Algorithm
LSTM	Long Short-Term Memory

LTO	Lithium-Titanate
LUT	Look-Up-Table
MAE	Mean Absolute Error
MCU	Microcontroller Unit
MSE	Mean Squared Error
MOSFET	Metal Oxide Semiconductor Field Effect Transistor
NARX	Non-linear autoregressive with exogenous input
NMC	Nickel Manganese Cobalt Oxide
OCV	Open Circuit Voltage
PDE	Partial Differential Equation
PF	Particle Filter
PIO	Proportional Integral Observer
PWM	Pulse-Width Modulation
RBF	Radial Basis Function
RELU	Rectified Linear Unit
RLS	Recursive Least-Squares
RMSE	Root Mean Squared Error
RMSP	Root Mean Squared Propagation
RNN	Recurrent Neural Network
SBPM	Sparse Bayesian Predictive Modeling
SGD	Stochastic Gradient Descent
SGDM	Stochastic Gradient Descent with Momentum
SMO	Sliding Mode Observer
SOA	Safe Operating Area
SOC	State-of-Charge
SOH	State-of-Health
SOP	State-of-Power
SPI	Serial Peripheral Interface
SPKF	Sigma Point Kalman Filter
SVM	Support Vector Machine
UDDS	Urban Dynamometer Driving Schedule
UKF	Unscented Kalman Filter

Chapter 1

Motivation

1.1 Context

The theme of this thesis was proposed by Continental Engineering Services (CES), and all the practical work done in this thesis was executed at the facilities of CES Porto.

The company already has a BMS product developed, and a solution implemented for battery state estimation. However, this theme was proposed to explore new approaches and expand the knowledge in the area during the process. The solutions proposed during the development phase of this thesis were integrated in the existing software and BMS developed by the company, and then evaluated according to performance metrics such as accuracy and memory usage.

1.2 The need for battery management systems

In today's world, batteries serve as power sources for a vast number of applications, ranging from low power applications, such as portable devices, all the way up to higher powered applications, such as electric vehicles and hybrid electric vehicles. A common need to all these battery powered applications is a Battery Management System (BMS), capable of assuring safety and efficiency during the battery usage.

Lithium-ion battery cells have a strict Safe Operating Area (SOA), which defines voltage, current and temperature limits for the cell's operation. Operating the battery cells outside the defined limits accelerates the battery aging process and can possibly endanger the user. Figure 1.1 showcases some of the hazards that can occur when the battery cells are operating outside the designated limits for voltage and temperature values.

To avoid these hazards, and assure an efficient use of the battery pack, a BMS is needed to constantly monitor voltage, current and temperature levels of each battery cell, and apply corrective measures to maintain each cell in their SOA.

Another important function of the BMS is battery states estimation, specifically the State-of-Charge (SOC), State-of-Health (SOH) and State-of-Power (SOP) estimates. Knowing these values allows the user to know the remaining charge of the battery pack, the aging level of the battery and

the amount of power the battery pack can instantaneously provide. Precise estimation of battery states allows a safer and more efficient use of the battery pack.

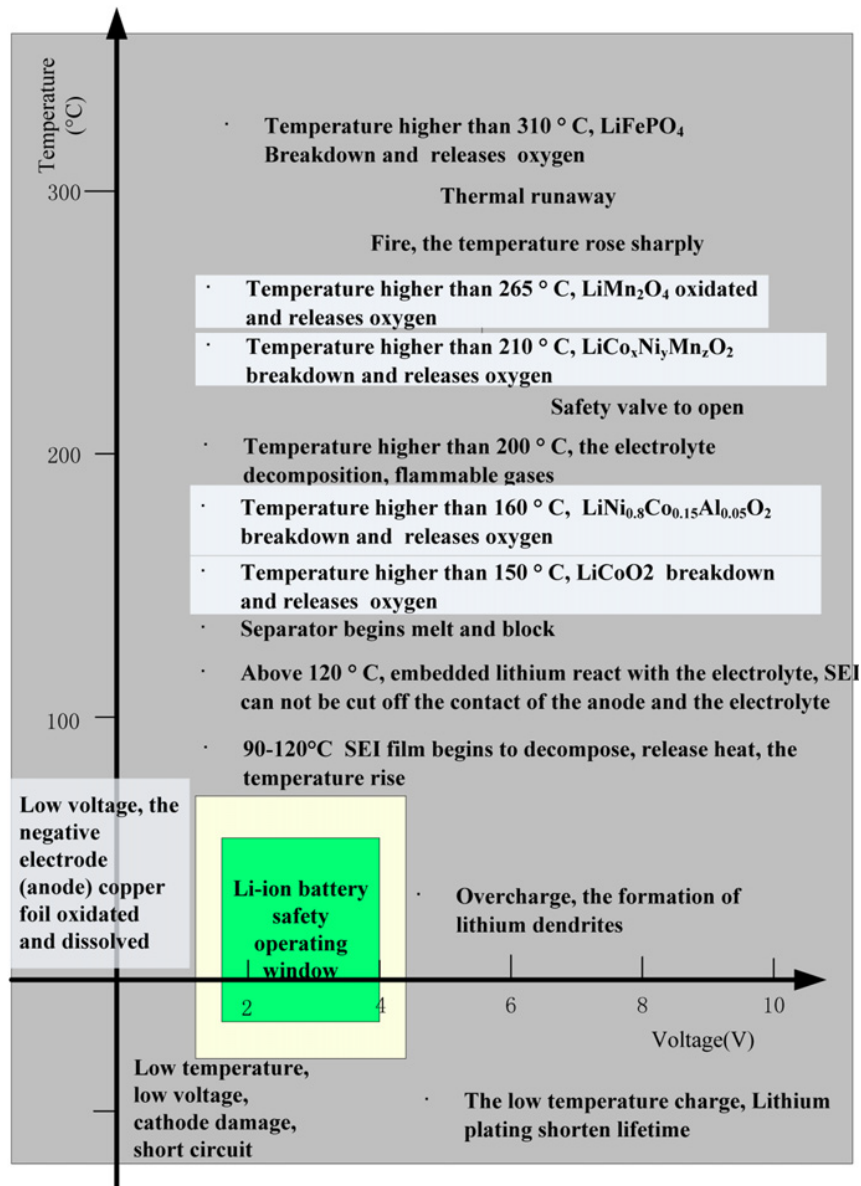


Figure 1.1: Lithium-ion cell operating window, image taken from [9].

1.3 Goals of the thesis

The three main goals of this thesis are the following:

1. Design and simulate machine learning algorithms for battery state and parameter estimation, using MATLAB software.
2. Implement these algorithms on an existing 48V BMS.

3. Test the developed algorithms using publicly available online battery datasets, and experimental data.

1.4 Document structure

The remainder of this document is divided between five chapters:

Chapter 2 contains the state of the art in BMS. In a first section, the general structure of a BMS is presented and the main functions of the BMS are reviewed and explained in detail. On the next section, battery balancing topologies are reviewed in detail and an evaluation is made, describing each method's strengths and weaknesses. In the third section, a critical review of common battery modeling techniques is provided. In the fourth section, state estimation techniques employed for lithium-ion battery cells are reviewed, with a focus on machine learning SOC estimation algorithms, which fit the main scope of the work to be done in the thesis. The last section is dedicated to online battery models parameter identification, where methods commonly used for online parameter identification are critically reviewed and described.

Chapter 3 contains a description of the algorithms used and implemented in this work. In a first section, Feedforward Neural Networks are reviewed in greater detail, with an explanation on its implementation and on the methods and general process used for neural network training. On the next section, the model used in this paper for estimating the SOC of the battery is presented, alongside all details required for its implementation. In the third section, the proposed model used for estimating the SOH of a battery is presented. The last section presents the algorithm used for estimating the parameters of the model in great detail.

Chapter 4 contains the simulation and practical results achieved with the proposed models. In a first section, the results achieved with the SOC model are presented, using an online available dataset, and using battery data made available by Continental Engineering Services. In the second section, the results obtained with the SOH model using data taken from an online available dataset are presented. The last section presents the estimation results with the parameter estimation algorithm used.

Chapter 5 presents the design and practical results for the HW setup, designed to emulate the voltage and current behaviour of a lithium-ion battery pack.

Chapter 6 presents the final remarks on the work developed during this thesis, and some comments on future work to be done in this topic.

Chapter 2

State of the Art

2.1 Structure and main functions

A BMS can be defined as an embedded system, composed of hardware elements such as sensors, actuators, contactors, connected to single or various processing units (microcontrollers), running software to monitor the battery pack, ensuring safety and efficiency on its use.

A BMS general structure is presented in Figure 2.1. The BMS front-end ICs, such as [1, 2, 3] are used to take voltage, temperature and current readings to send them back to the master controller, and receive commands regarding cell balancing. Depending on the number of cells, multiple BMS front-end ICs can be used side-by-side to monitor the battery cells. In Electrical Vehicles (EV), contactors on high and low side are used to be able to completely disconnect the battery pack, alongside a fuse to protect against high current surges. Data shared between these elements and the controller is passed through an isolation interface, so in case of a fault the controller is protected. Controller shares battery data to the rest of the vehicle systems through an external communication channel, such as a CAN Bus [4, 5, 6].

In terms of BMS architecture, two options are often referred: a centralized approach or a distributed approach.

In centralized approaches, a master controller is used to command all slave modules, in this case the BMS front-end ICs. The structure presented in Figure 2.1 is an example of a centralized approach. This type of architecture is the most commonly used in EV applications, along with passive cell balancing [7].

In distributed approaches, multiple controllers are used in parallel to monitor different cells of the battery pack and coordinate between themselves through a communication channel, as seen in Figure 2.2.

Benefits of distributed approaches over centralized ones come in its scalability to different sized battery packs in different applications, higher processing power and bringing computation as close to the cells as possible. This makes distributed approaches much more suitable for active balancing, since complex control patterns needed for PWM signals (to activate MOSFET switches of active balancing circuits), are very complicated to perform from a master central controller,

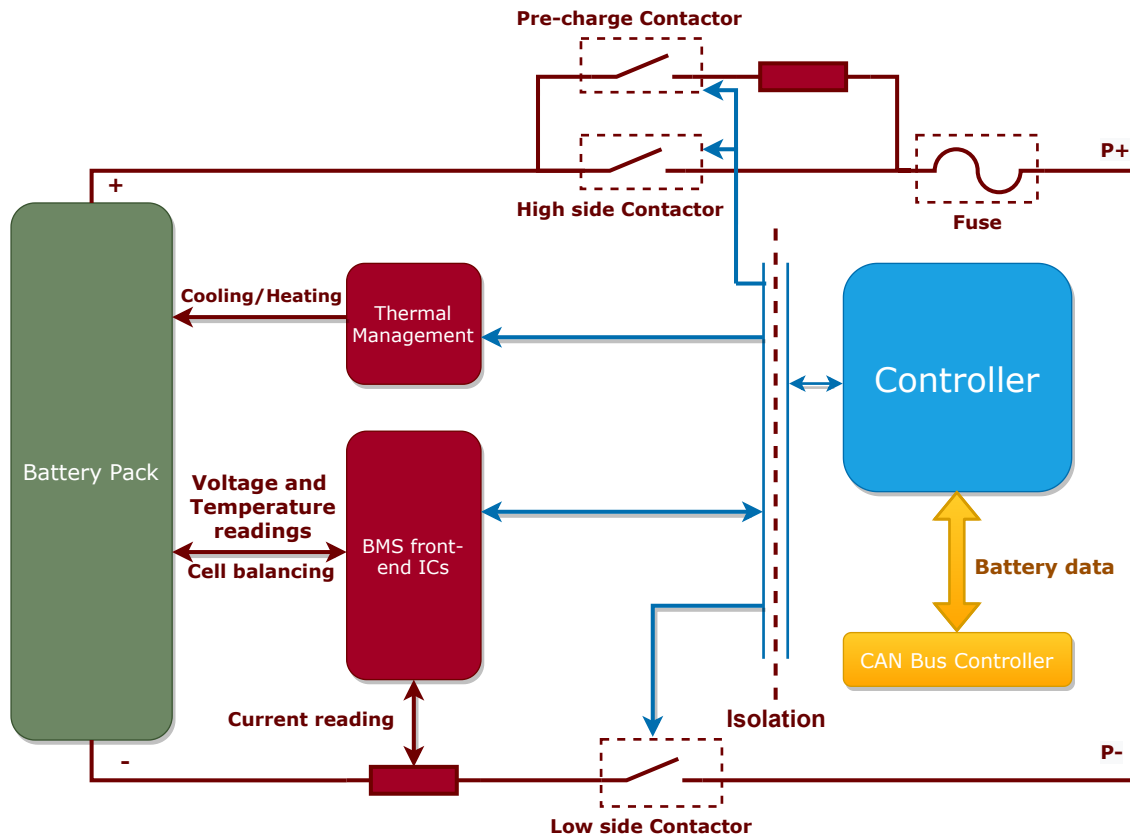


Figure 2.1: EV BMS general structure.

physically placed further away from the cells. In distributed approaches, with processing power as close to the cells as possible, this PWM generation can be performed locally at cell level, making active balancing topologies feasible [7].

Main disadvantages of distributed approaches over centralized ones come in its cost and size, since the need for multiple controllers makes it more expensive and occupy more space.

The main components and functions performed by the BMS are:

- **Data acquisition**
- **Cell/Battery monitoring and protection**
- **Cell balancing**
- **On board fault diagnosis**
- **Communications**
- **Battery state and parameter estimation**

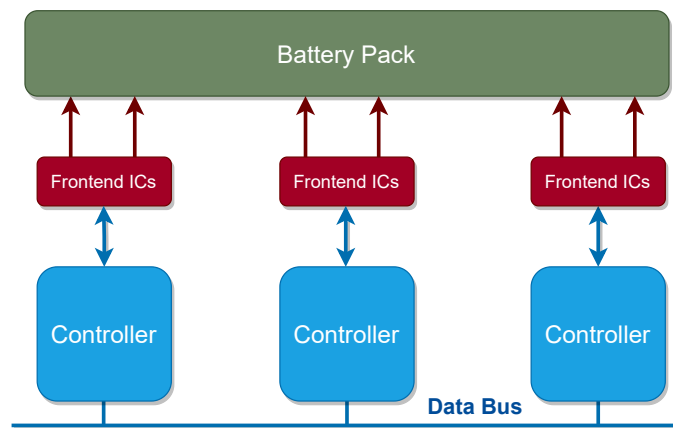


Figure 2.2: Distributed BMS structure.

Data acquisition

In this context, data acquisition is defined as the measurement, conditioning and storage of relevant data regarding battery performance, such as: voltage measurement across single cells and for the whole pack, current measurement across the battery pack and temperature measurement at each cell or in critical points, at proper sampling frequency and accuracy.

Typically, voltage measurements are performed across the whole pack and across each individual cell connected in series. For accurate state of charge estimation, very precise voltage measurement is required [8]. Cell voltage measurements are usually done by BMS front-end chips and can achieve measurement accuracy of $\pm 1\text{mV}$ [4].

Current measurement is usually performed across the whole pack, or across each parallel string of cells. Current measurement is commonly performed in one of two ways: shunt resistor or hall sensor [4]. Current measurement accuracy is also essential for state estimation accuracy, especially in methods that rely on *Coulomb Counting*, since any offset error in current measurement will be integrated over time and accumulate error [8].

Temperature measurements can be taken by placing temperature sensors at each individual cell or by placing fewer temperature sensors at optimal points of the battery pack, determined by simulations of heat transfer on the system. In this case, research is required to determine where to place the sensors and how many are needed to accurately determine temperature across the battery pack [4].

Cell/Battery monitoring and protection

As mentioned, lithium-ion batteries have a strict SOA regarding voltage, temperature and current. Exceeding any of these limits can accelerate battery degradation process, and elevate the risk of the battery pack catching fire, or even explode in some cases. Therefore, it's up to the BMS to keep the battery safe and efficient, by maintaining it in its SOA.

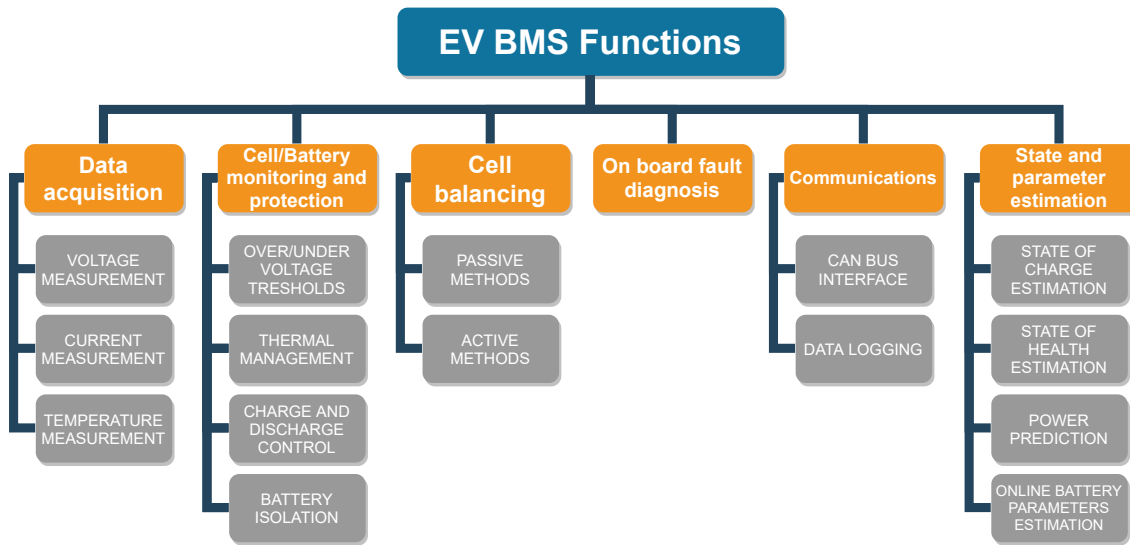


Figure 2.3: EV BMS Components and Functions.

Upper and lower voltage thresholds are defined by manufacturers of Li-ion batteries. The usual operating voltage of lithium-ion batteries is somewhere between 1.5 V and 4.2 V, varying depending on its chemistry [9]. Actions such as cell balancing and charge/discharge control, all contribute to maintain the cell within its voltage limits.

Thermal management actions are employed to secure the battery within temperature limits and to assure equal temperature distribution across the pack. These temperature limits can vary whether the battery is in charge or discharge operation [8]. Depending on required action, the BMS can decide the heating or cooling power to apply to the battery pack, or even if a shutdown is required.

Charge/discharge control is performed by the BMS. Optimal charge and discharge of Li-ion batteries assures the efficiency and life span of the battery pack. In charging state, CC-CV (Constant Current - Constant Voltage) charge method is recommended and in discharging state, DCM (Discontinuous Current Mode) control is recommended [5].

To be able to completely isolate the battery pack, electromagnetically driven contactors are used. Usually three contactors are used, one on the low side, one on the high side, and one typically referred as the pre-charge contactor, placed in parallel with the high side contactor (Figure 2.1). In some BMS, instead of contactors, two MOSFET transistors in series are used to signal charge or discharge operation. Although these contactors/transistors can be used to protect and isolate the battery pack, some passive safety mechanisms are also placed to assert continuous safe operation. These passive safety precautions are required to protect the battery against worst case scenarios, in the occurrence of electrical faults in the cells. These include positive temperature coefficient resettable fuses to protect against overcurrents and bimetallic circuit breakers to protect against short circuits at the cell [6].

Cell balancing

During normal operation, some cells may end up with fewer/more charge compared to adjacent cells, existing a cell charge imbalance across the pack. This can be due to unpredictable factors such as difference in parameters between same chemistry cells manufactured by the same company, or due to unequal temperature distribution across the battery pack. Cell imbalance can reduce the amount of energy that can be extracted from the battery pack, can accentuate temperature differences across the cells of the pack, and provoke aging and degradation difference between cells. Therefore, to efficiently use the battery pack, cell balancing methods must be used. These are commonly described in two ways: passive methods and active methods [8].

Passive methods can also be described as dissipative methods, since the excess charge of a cell in comparison with other cells is dissipated through a passive component as heat, until the less charged cells reach its charge level. This method is completely inefficient, since all energy is dissipated and wasted, and is slow compared to active methods. This method also only equalizes cells when charging.

Active methods can also be described as non-dissipative methods, since the excess charge of a cell in comparison with other cells is transferred among cells in various different ways, meaning that no energy is purposefully wasted away. The downside of these methods compared to passive methods is that they require much more components, therefore are complex and more expensive to implement.

On board fault diagnosis

During normal operation, many faults may occur that limit or remove functionality of the BMS, such as: actuator faults, sensor faults, over and under-voltage at the cell, excess current, faulty connections, insulation faults, communication and networking faults.

To be a reliable system, the BMS should be able to diagnose these faults and take necessary actions to deal with possible problems, using diagnosis technology to auto check itself [5, 9].

Communications

As the BMS will be part of a vehicle, it will have to be capable of receiving and transmitting commands and data to other electronic systems on the vehicle. With the BMS accessible through networking come useful and necessary features such as: online calibrating and monitoring, online code updates and drive range prediction, via the GPS combined with data on the online estimation of SOH and SOC of the battery, to inform the driver the charge remaining and the distance that the vehicle can continue to drive before the battery runs out of charge [5, 9].

In vehicles, the usual choice for inter-module communication between electronic systems is the CAN bus, due to its robustness and easiness to add more modules to the bus. In such a system, the BMS needs to have a CAN bus interface to connect to the vehicle bus and be a part of the network of connected modules on the vehicle [4].

Battery state and parameter estimation

State and parameter estimation is the crucial and most complex part of the BMS functions. Since these states and battery parameters cannot be directly measured, they are estimated and calculated in various different ways, which will be described in Sections 2.4 and 2.5. Accurate state estimation assures efficient and safe use of the battery.

State of charge is defined as the ratio between remaining charge on the battery cell and the total charge the cell can hold, being a value between 0% and 100%. State of charge is commonly estimated with measurements of voltage, current and temperature at the cells, and in some cases previous values of voltage, current and estimated state of charge. In a single cell SOC is defined as:

$$SOC = \frac{C}{C_t} \quad (2.1)$$

Where C is the current charge of the cell and C_t the charge at 100% SOC. The SOC of a battery pack with multiple cells connected in series could be defined as the average of the SOC of all cells or as the minimum value of SOC between all cells [9].

State of health defines the state of degradation of the battery due to its usage, could be defined as the ratio between total current max capacity and the rated capacity of a new cell.

$$SOH = \frac{C_{max}}{C_{rated}} \quad (2.2)$$

Where C_{max} is current max capacity of the cell, and C_{rated} is the rated capacity of a brand new cell.

For example, if the SOH of a cell is 90%, then that cell can only hold 90% of its rated charge that it could hold when it was still a new battery. Estimating the current SOH is important to predict its remaining useful life and understand when the battery is no longer suitable for its application. For most applications in EVs, when the battery SOH drops below 80%, the battery is no longer considered suitable [9].

Power prediction is an estimation of the power that the battery pack can supply over a limited time period, also referred as the current state of power of the battery. The battery power capability is limited by its SOA, defined by SOC, temperature, voltage and current limits. This value is estimated using the SOC, SOH and depends on the instantaneous operating conditions of the battery [10].

Most of the techniques used for state estimation rely on battery models, whose parameters are extracted from experimental data obtained from the battery. These model parameters vary non-linearly due to temperature at the cell, current SOC and more. Through extended use of the battery, these initially estimated parameters may no longer be used to accurately determine the battery states, therefore online cell parameterization techniques are used to maintain an accurate battery model throughout its use [11].

2.2 Balancing topologies overview

As presented in Figure 2.4, balancing topologies are divided between two main groups: the passive/dissipative methods, and the active/non-dissipative methods.

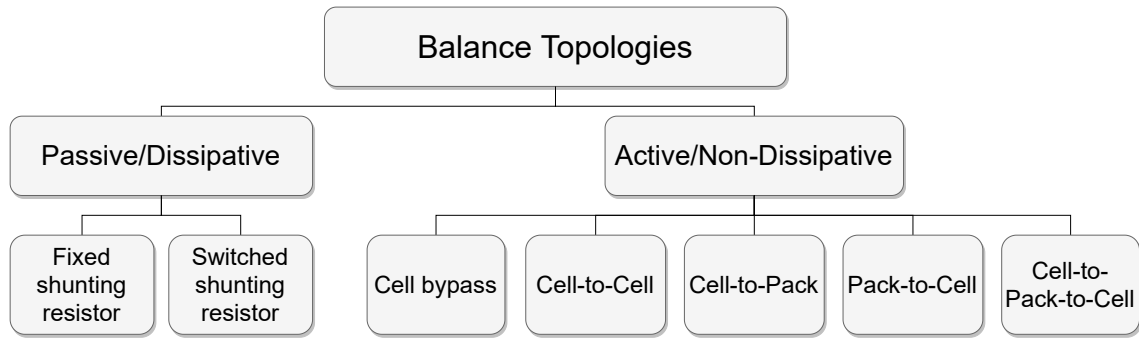


Figure 2.4: Balancing topologies.

2.2.1 Passive topologies

As mentioned before, the definition of passive balancing methods is essentially methods that dissipate excess energy of a cell compared to other cells, as heat, meaning that energy is purposefully wasted away in order to eventually obtain a balanced battery pack during charging operation. These methods are divided in two ways: fixed shunting resistor, which doesn't require any switches or control, and switched shunting resistor [12, 13].

Fixed shunting resistor method, represented in Figure 2.5, consists of connecting a resistor in parallel with each battery cell, so the current is continuously limited, or completely bypassed from the cell when the cell reaches its upper limit voltage threshold, dissipating energy as heat through the shunting resistor, meaning that the cells that already reached their upper voltage limit are no longer being charged while remaining cells are still being charged, so eventually charge balance across all cells will be reached. Main advantage of this method is its low cost and simplicity, since no external control is required. Main disadvantages are the continuous waste of energy during operation dissipated as heat, causing the temperature on the battery pack to rise, and its lower balancing speed compared to active methods [12, 13].

In the switched shunting resistor method, represented in Figure 2.6, the main difference compared to the fixed shunting resistor method is the addition of a switch in series with the resistor, which means that when the switches are off no energy is purposefully dissipated on the resistor, so the battery operates as usual when no switch is on. This added switch enables the BMS to bypass the current from the cell only when it senses imbalance between them, instead of continuously wasting energy during operation as in the fixed shunting resistor method, making this method much more energy efficient. Advantages are still its simplicity and low cost when compared with active methods. Disadvantages of this method are the added complexity due to switch control, the

waste of energy dissipated as heat, and although it is faster than the fixed shunting resistor, it is still slow compared to active methods [12, 13].

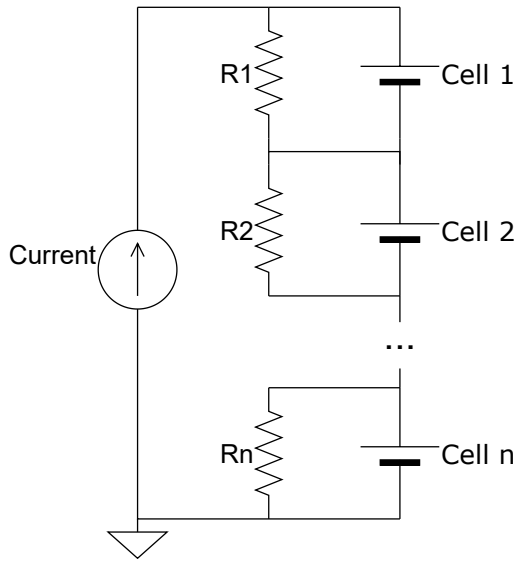


Figure 2.5: Fixed shunting resistor.

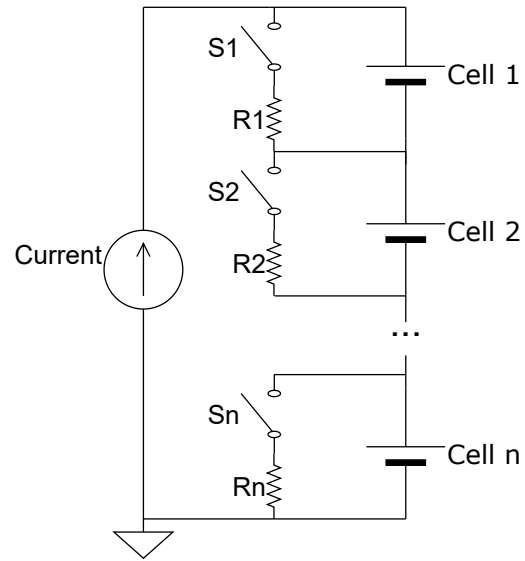


Figure 2.6: Switched shunting resistor.

2.2.2 Active topologies

As mentioned in this document, active balancing methods are methods that are capable of transferring charge from higher charged cells to lower charged cells, so no energy is purposefully wasted away. These methods are capable of balancing cells very quickly compared to passive methods, at the expense of complex control, higher cost and added components, such as capacitors, inductors, diodes and extra switches. These methods are commonly divided in five ways: cell bypass, cell-to-cell, cell-to-pack, pack-to-cell and cell-to-pack-to-cell [12, 14].

In cell bypass methods, current is bypassed from the cell when the cell reaches its upper voltage limit threshold, by creating an alternative path for the current, removing the cell from the current path, while remaining cells continue to be charged until they reach their upper voltage limit. One method is commonly referred [12, 14]:

- **Complete shunting method**, which involves putting a switch in series with the cell and another switch in parallel to the switch in series with cell, as seen in Figure 2.7, allowing for complete removal of the cell on the current path depending on which switch is on.

In cell-to-cell methods, excess energy of the most charged cells is transferred directly to the adjacent least charged cells in the pack. In cell-to-cell techniques, five methods are commonly referred [12, 13, 14]:

- **Switched capacitor method**, with circuit configuration presented in Figure 2.8. In this method two switching states are used: in the first state each capacitor is connected in parallel with its upper cell, being charged to the voltage of that cell; in the second stage each

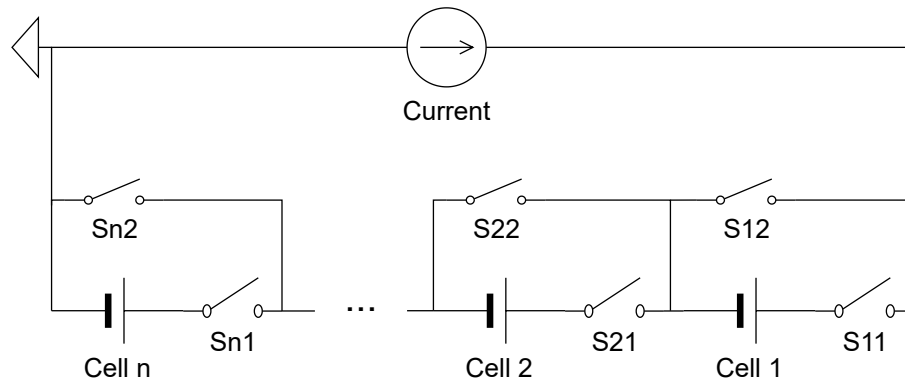


Figure 2.7: Complete shunting method.

capacitor is connected in parallel with its lower cell, being charged or discharged to that cell, depending on which cell has higher voltage. Therefore, charge will be transferred between adjacent cells using this method until they are at the same voltage/charge level.

- **Double switched capacitor method**, with circuit configuration presented in Figure 2.9. This method works the same way as the switched capacitor method, with the addition of another capacitor row, making it a faster method of balancing in comparison with the switched capacitor method, but more complex and expensive.

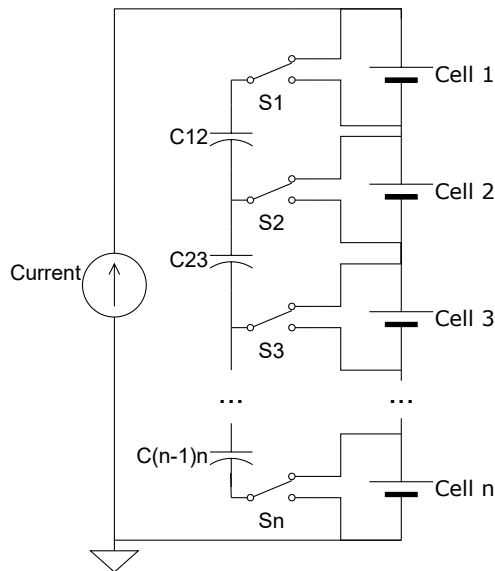


Figure 2.8: Switched capacitor method.

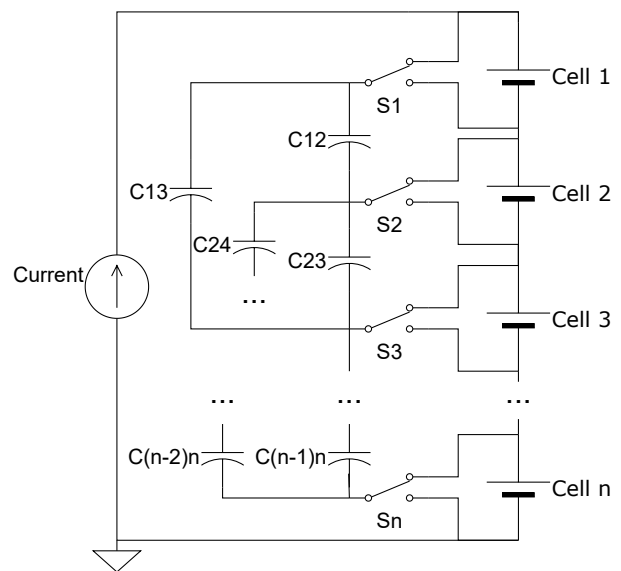


Figure 2.9: Double switched capacitor method.

- **Cûk converter method**, with circuit configuration presented in Figure 2.10. Each pair of adjacent cells is connected together by a converter module which allows transfer of energy between cells. The transfer of energy is achieved through the capacitor, where the higher charged cell charges the capacitor, then the energy stored in the capacitor is used to charge

adjacent less charged cell. Transistors are driven by a PWM signal, where altering the duty-cycle increases/decreases the energy transfer and its direction.

- **PWM converter method**, with circuit configuration presented in Figure 2.11. Each pair of adjacent cells is connected together by a converter module which allows transfer of energy between cells, through an inductor. The energy transfer and its direction is controlled by altering the duty cycle of the PWM signal used to control the transistors involved.

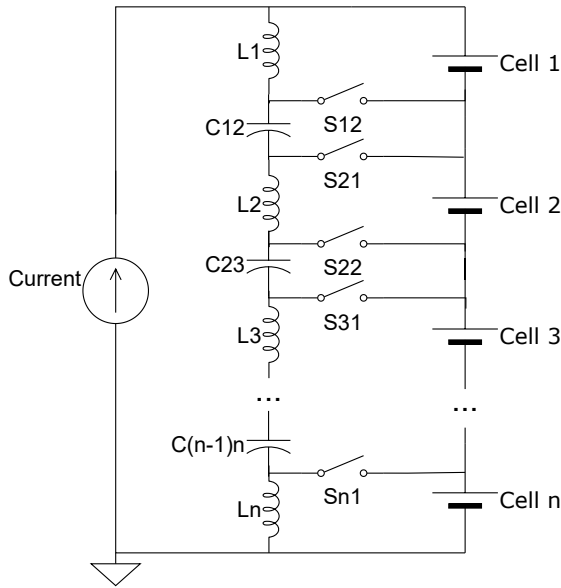


Figure 2.10: Cûk converter method.

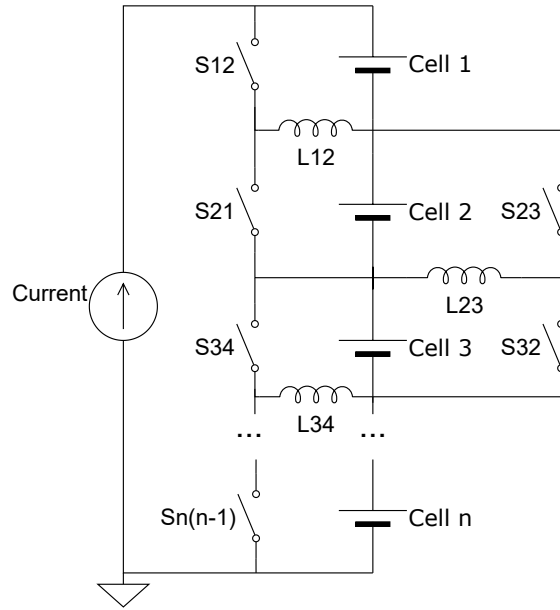


Figure 2.11: PWM converter method.

- **Quasi resonant converter method**, very similar configuration to the PWM converter method circuit, but instead of using an inductor, a resonant circuit is used for the transfer of energy between cells.

In cell-to-pack methods, excess energy of the most charged and highest voltage cell is transferred to the whole battery pack. In cell-to-pack techniques, five methods are commonly referred [12, 13, 14]:

- **Shunt inductor method**, with circuit configuration presented in Figure 2.12. When the BMS detects a higher voltage cell in comparison with the others, an inductor is placed in parallel with that cell, then the inductor is placed in parallel with the whole pack, with the purpose of transferring energy via the inductor from the highest charged cell to all the cells of the pack, where varying the duty-cycle of the switches enables more or less current to be drawn from the higher voltage cell to the whole pack.
- **Boost shunting method**, with circuit configuration presented in Figure 2.13. When the BMS detects a higher voltage cell in comparison with the others, the switch connected to that cell and the general switch S0 are controlled with a PWM signal and start transferring the excess energy of that cell to remaining cells in series.

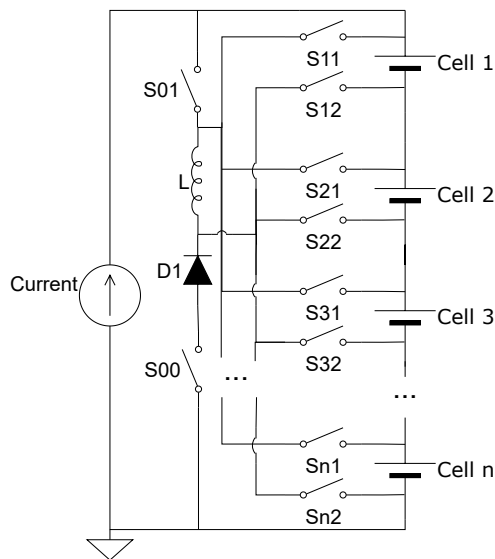


Figure 2.12: Shunt inductor method.

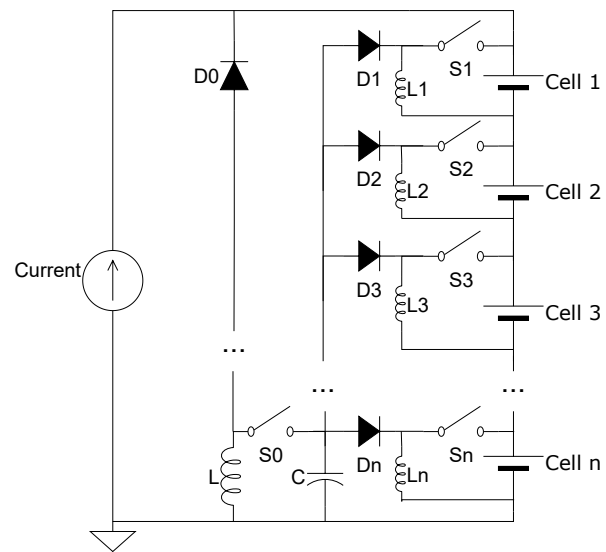


Figure 2.13: Boost shunting method.

- Multiple transformers method**, with circuit configuration presented in Figure 2.14. In this configuration, an isolated converter for each cell is obtained, in the circuit composed of the switch in series with the secondary side of the transformer and the primary sides of all transformers placed in parallel. When imbalance conditions are verified for a cell, energy is transferred to the pack from that cell by controlling the switch associated to that cell. In a first phase, the secondary side associated to the most charged cell is placed in parallel with that cell by activating its switch, storing energy as magnetic flux in the transformer; in the second phase, the switch associated with the cell is turned off, and the switch connected to the primary sides of the transformers is turned on, passing the energy stored in the transformer to the whole pack by flowing current from the primary side of the transformer to the pack.
- Multi-secondary windings transformer method**, with a very similar configuration to the multiple transformers method configuration, but instead of having multiple primary sides of the transformers connected in parallel, a single primary side coil is used combined with multiple secondary side coils, which are connected to the cells. When imbalance conditions are verified for a cell, energy is transferred to the pack from that cell by controlling the switch associated to that cell. Switch control is similar to the multiple transformers method.
- Switched transformer method**, with circuit configuration presented in Figure 2.15. Similar to the multi-secondary windings transformer method configuration, but instead of having multiple secondary sides of the transformer for each individual cell, a single secondary side coil is used for all the cells. The secondary side is connected through two switches to each individual cell, allowing each cell to be placed in parallel with the secondary side of the transformer. Switch control is similar to the multiple transformers method.

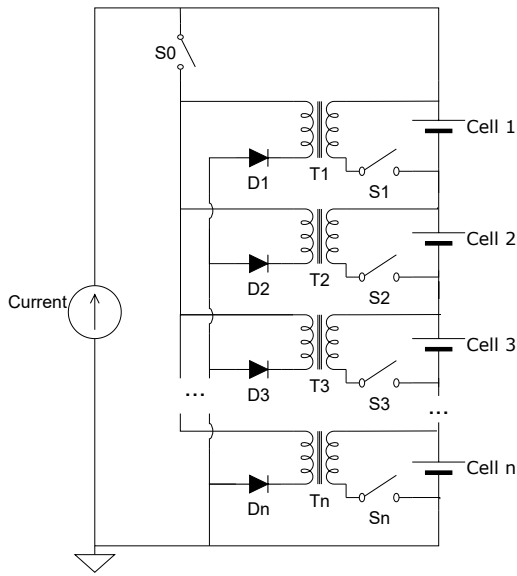


Figure 2.14: Cell to pack multiple transformers method.

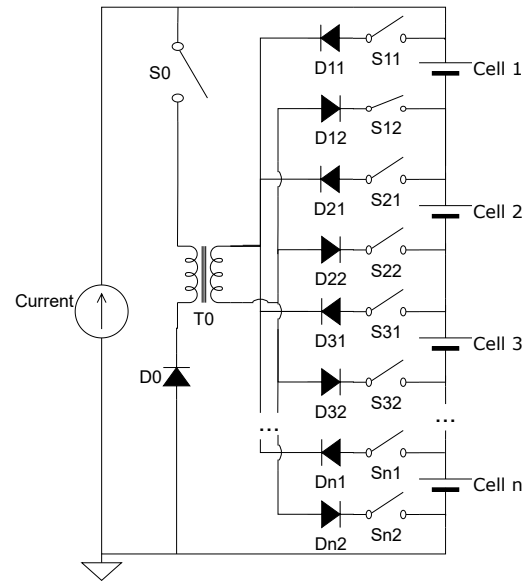


Figure 2.15: Cell to pack switched transformer method.

In pack-to-cell methods, energy is transferred from the whole battery pack to the least charged cell. In pack-to-cell techniques, four methods are commonly referred [12, 13, 14]:

- **Voltage multiplier method**, with circuit configuration presented in Figure 2.16. The switch in the image is controlled by a PWM signal, so the circuit has two phases: in the first, when the switch is on, each cell is discharged to the capacitors through the low side diode in series with the cell, charging the capacitors; in the second phase, when the switch is off, charging current is fed to the capacitors and the cells through the high side diode, and each cell will receive higher or lower current depending on its voltage level compared with remaining cells.
- **Multiple transformers method**, with circuit configuration presented in Figure 2.17. When imbalance is detected, the switch is activated and current flows through the primary sides of the transformers connected in parallel, inducing currents in the secondary sides of the transformers. Secondary sides with lesser reactance because of lower voltage at the cells will have more induced current than other secondary sides with higher reactance due to higher voltage cells, therefore lower voltage, and less charged cells, will receive more current than higher charged cells.
- **Multi-secondary windings transformer method**, with a very similar configuration to the multiple transformers method configuration, but instead of multiple primary sides of transformers connected in parallel, a single primary side coil for multiple secondary side coils is used.

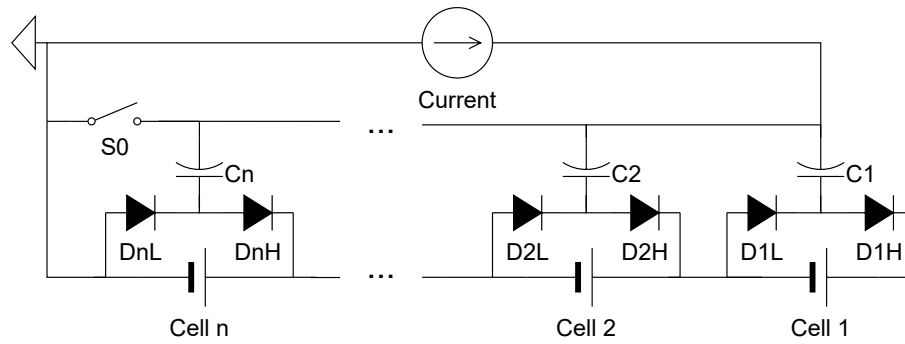


Figure 2.16: Voltage multiplier method.

- **Switched transformer method**, with circuit configuration presented in Figure 2.18. Same principle as the multi-secondary windings transformer method, but instead of multiple secondary sides of transformers connected to each cell, a single secondary side coil for multiple cells is used. Each cell can then be connected by switches to the secondary side, enabling another mode of operation where one individual cell can be targeted by activating its switches and connecting them to the secondary side of the transformer.

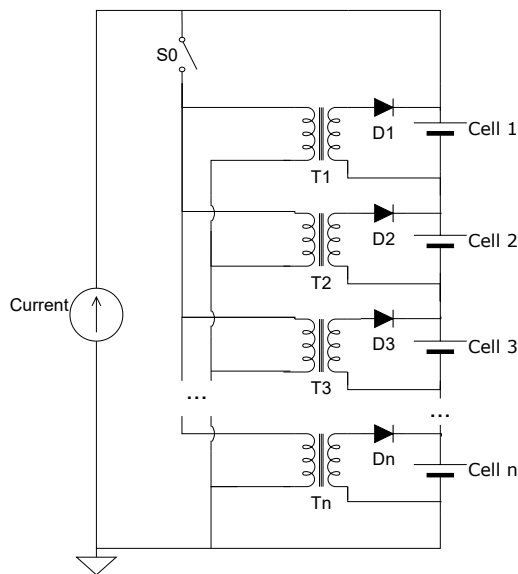


Figure 2.17: Pack to cell multiple transformers method.

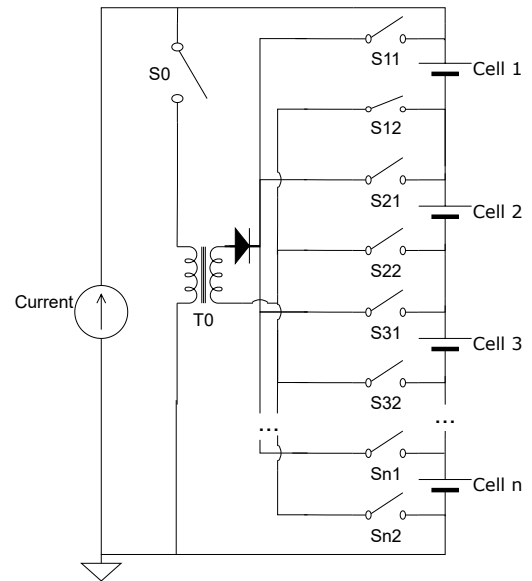


Figure 2.18: Pack to cell switched transformer method.

In cell-to-pack-to-cell methods, energy is transferred from a cell or group of cells to the whole pack, then from the whole pack to another cell or group of cells, or directly from a group of cells to another group of cells. In cell-to-pack-to-cell techniques, six methods are commonly referred [12, 13, 14]:

- **PWM converter method**, with circuit configuration presented in Figure 2.19. In this method buck-boost converters connect the cells to a single capacitor. Through switching

control of each inverter leg (two switches in series), currents can be generated in both directions in each inductor through differences in cell voltages and duty cycle of the switching waveform, which can be used to charge undercharged cells.

- **Single switched capacitor method**, with circuit configuration presented in Figure 2.20. Similar to the cell-to-cell switched capacitor method, but now only one capacitor is used, so in first phase the highest voltage cell is placed in parallel with the capacitor to charge the capacitor to its voltage, then in second phase the lowest charged cell is placed in parallel with the capacitor to be charged by it. Difference in comparison with the switched capacitor method is that now any cell can be charged by any cell, instead of only being able to charge adjacent cells.
- **Single switched inductor method**, with circuit configuration very similar to the single switched capacitor method, but the component used to store energy and transfer it is an inductor now instead of a capacitor.

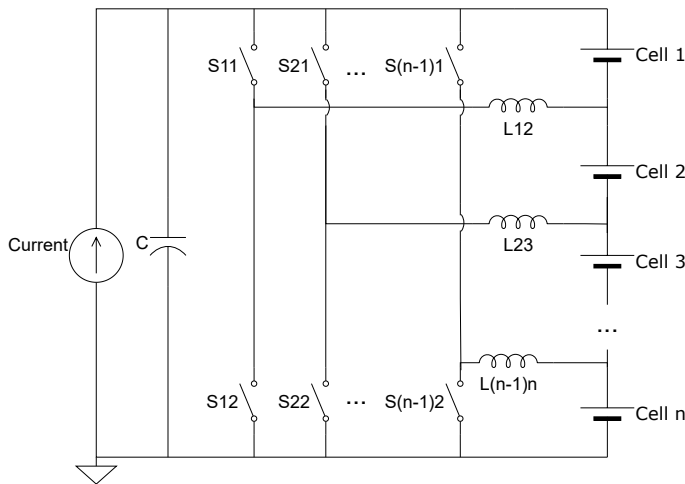


Figure 2.19: Cell to pack to cell PWM converter method.

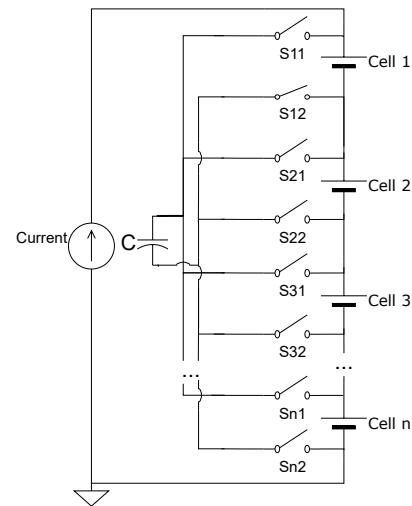


Figure 2.20: Cell to pack to cell single switched capacitor method.

- **Bidirectional multiple transformers method**, with circuit configuration presented in Figure 2.21. This method is the combination of the cell-to-pack multiple transformers method and the pack-to-cell multiple transformers method, enabling cell-to-pack or pack-to-cell energy transfer operations.
- **Bidirectional multi-secondary windings transformer method**, with a very similar configuration to the bidirectional multiple transformers method configuration, but instead of multiple primary sides of transformers connected in parallel, a single primary side coil for multiple secondary side coils is used. This method is the combination of the cell-to-pack multi-secondary windings transformer method and the pack-to-cell multi-secondary windings transformer method, enabling cell-to-pack or pack-to-cell energy transfer operations.

- **Bidirectional switched transformer method**, with circuit configuration presented in Figure 2.22. This method is the combination of the cell-to-pack switched transformer method and the pack-to-cell switched transformer method, enabling cell-to-pack or pack-to-cell energy transfer operations.

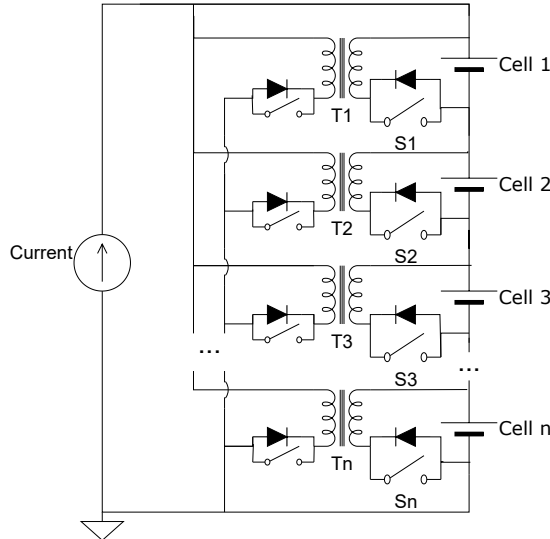


Figure 2.21: Bidirectional multiple transformers method.

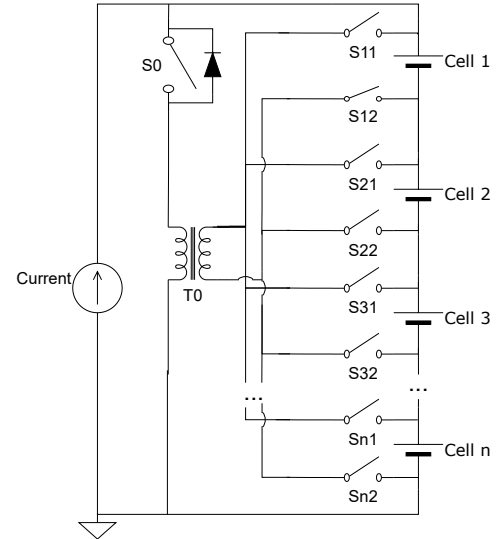


Figure 2.22: Bidirectional single switched transformer method.

2.2.3 Balancing topologies summary

In Table 2.1, a general overview of each method's characteristics is provided. Each method is characterized based on seven metrics: cost (M1), implementation complexity (M2), efficiency (M3), equalization speed (M4), size (M5), switch stress (M6) and modularity (M7). For each metric, a rating between 0 and 3 is given, where 0 is the worst possible rating, and 3 is the best rating. For example, if a method receives a rating of 3 in cost, it means the method has a low implementation cost, compared to other methods.

Active topologies have clear advantages over passive topologies, when faster balancing and efficiency are the main goals. However, their much higher cost, complexity, size and possible safety issues have limited their use in EVs, where passive balancing is more prevalent [7].

Topology	Method	M1	M2	M3	M4	M5	M6	M7
Passive	Fixed shunting	3	3	0	0	3	3	3
	Switched shunting	3	3	0	1	3	1	3
Cell bypass	Complete shunting	3	3	2	3	3	1	3
Cell-to-Cell	Switched capacitor	3	3	3	1	3	1	3
	Double Switched capacitor	2	3	3	2	2	1	3
	Cûk converter	2	1	2	2	2	3	2
	PWM converter	2	1	2	2	2	3	2
	Quasi resonant converter	1	1	2	2	1	3	2
Cell-to-Pack	Shunt inductor	2	1	2	1	3	2	1
	Boost shunting	2	1	2	2	2	3	1
	Multiple transformers	1	1	1	2	0	2	3
	Multi-secondary windings transformer	1	1	1	1	0	2	1
	Switched transformer	1	1	1	1	1	2	2
Pack-to-Cell	Voltage multiplier	3	2	3	2	2	1	3
	Multiple transformers	1	3	1	2	0	2	3
	Multi-secondary windings transformer	1	3	1	2	0	2	1
	Switched transformer	1	1	1	1	1	2	2
Cell-to-Pack-to-Cell	PWM converter	1	1	2	2	2	2	3
	Single switched capacitor	3	1	3	1	3	2	1
	Single switched inductor	2	1	2	1	3	2	1
	Bidirectional multiple transformers	1	1	1	2	0	2	3
	Bidirectional multi-secondary windings transformer	1	1	1	2	0	2	1
	Bidirectional switched transformer	1	1	1	2	1	2	2

Table 2.1: Cell balancing methods overview.

M1 - Cost (0: Very expensive; 3: Cheap)

M2 - Implementation complexity (0: Very complex; 3: Simple)

M3 - Efficiency (0: None; 3: Good efficiency)

M4 - Equalization speed (0: Very slow; 3: Fast)

M5 - Size (0: Very big; 3: Small)

M6 - Switch stress (0: Very high; 3: Low)

M7 - Modularity (0: Very low; 3: High)

2.3 Li-ion battery models

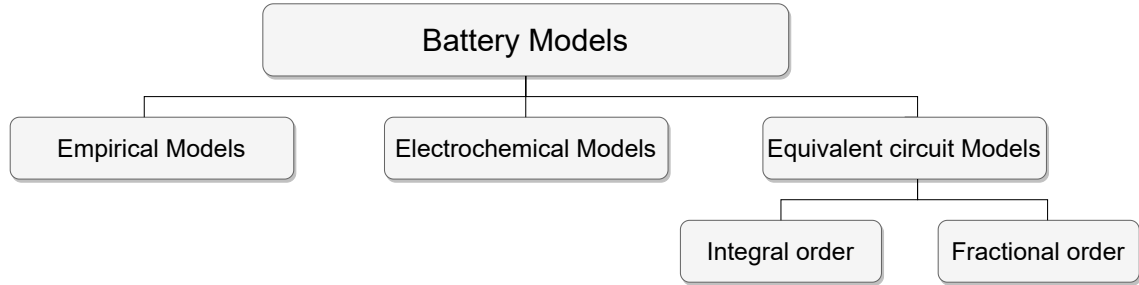


Figure 2.23: Battery models reviewed in this work.

Many state estimation techniques rely on accurate battery models, that emulate the electric, aging and thermal behaviour of the battery depending on its input values and the values of the parameters associated [11, 15].

Many research on this topic can be found, where battery models are proposed which have different accuracy and computer simulation cost. Naturally, the higher the complexity of the battery model, the higher its accuracy will be in emulating the battery behaviour. So, depending on its application, a balance must be found between the model complexity and its accuracy. Each of these models alter the way each state estimation algorithm and methods can be implemented. [11, 15].

As displayed in Figure 2.23, these battery models are commonly divided in three ways: empirical models, electrochemical models and equivalent circuit models [11, 15, 16, 17, 18, 19].

Empirical models

Empirical models describe the battery's internal physical behaviour using a mathematical equation or reduced order polynomial, so these models could be considered simplified electrochemical models. The mathematical equation used on these models correlates battery terminal voltage to SOC and current at the battery [15].

Four approaches are commonly referred in these types of models: the Shepherd model, the Unnewehr universal model, the Nernst model and the combined model (which is the combination of the three mentioned models) improving accuracy compared to each individual one [15, 16]. The combined model is described by the following equation in 2.3:

$$V_t = K_0 - \frac{K_1}{SOC_t} - K_2 * SOC_t + K_3 * \ln(SOC_t) + K_4 * \ln(1 - SOC_t) - i_t * R_s \quad (2.3)$$

Where V_t is the terminal voltage of the cell, i_t is the current flowing through the cell, R_s is the cell's internal resistance, and values K_0 , K_1 , K_2 , K_3 and K_4 are parameters used to describe the relation between OCV and SOC.

Electrochemical models

Electrochemical models can simulate and represent the inner reactions that occur inside the battery, defined through partial differential equations (PDEs) to be solved in time. Depending on the number and order of PDEs used to model the battery, higher or lower model complexity can be generated to achieve higher or lower state estimation accuracy results. These methods are considered as the most accurate way of representing battery behaviour. However, its complexity and computer simulation cost has limited its use in embedded applications, such as in EVs BMS [15, 16, 18].

Equivalent circuit models

Equivalent circuit models use electrical components such as voltage sources, resistors and capacitors to emulate battery behaviour. This type of battery model is often used in EVs applications, due to simplified mathematical equations compared to electrochemical models, with reduced computational burden that can be embedded in lower cost microprocessors, while still maintaining acceptable accuracy. In these circuits, each of the model parameters can be related to the electrochemical processes that are occurring in the battery cell [18].

These model methods can be divided in two groups: integral-order models and fractional-order models.

Integral-order electric equivalent models

In integral-order electric equivalent models, three types of models are commonly referred: Rint model, Randles model and nRC model (or thevenin circuit model) [11, 16, 18].

- **Rint model**, presented in Figure 2.24, consists of a voltage source V_{oc} representing the open circuit voltage of the cell, in series with a single resistor R representing internal resistance of the cell, where V_t is the voltage at the terminals of the cell. The value of both components depends on the current SOC, SOH and temperature of the battery cell. This is the simplest model used to represent the battery and can't represent the transient response of the battery in charge or discharge operations. In this model, the terminal voltage V_t is described as:

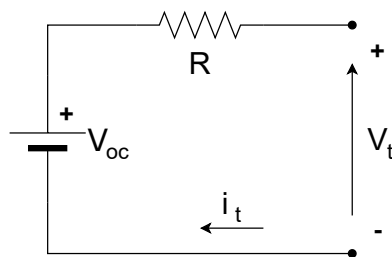


Figure 2.24: Rint model.

$$V_t = V_{oc} - R * i_t \quad (2.4)$$

Where V_{oc} is the OCV of the cell, that may vary with SOC and other operating conditions like current and temperature, V_t is the terminal voltage, R is the cell's internal series resistance and i_t is the current flowing through the cell.

- **Randles model**, presented in Figure 2.25, where the cell is considered as a large capacitor, C , which stores the charge of the cell. A parallel pair C_p and R_p is used to represent the transient response of the cell, R_s is used to model the internal series resistance and R_d is the self-discharge resistance. In this model, the terminal voltage V_t is described as:

$$V_t = V_{oc} - R_s * i_t - V_p \quad (2.5)$$

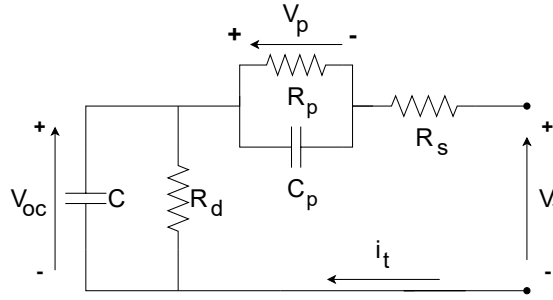


Figure 2.25: Randles model.

Where V_{oc} is the voltage at the capacitor C , V_t is the terminal voltage, V_p is the voltage at the R_p and C_p terminals, R_s is the cell's internal series resistance and i_t is the current flowing through the cell.

- **nRC model**, presented in Figure 2.26, the cell is modeled as a voltage source V_{oc} representing the open circuit voltage of the cell, in series with n number of parallel RC pairs to emulate the transient response of the cell, in series with resistance R_s representing internal resistance of the cell. The value of each component varies with the current SOC, SOH and temperature of the cell. The number of RC branches can be adjusted for each application, where higher number of RC pairs would lead to higher accuracy results and also higher model complexity. In this model, the terminal voltage V_t and voltage at the n th RC pair V_n are described as:

$$V_t = V_{oc} - R_s * i_t - V_1 - \dots - V_n \quad (2.6)$$

$$\dot{V}_n = -\frac{1}{R_n C_n} V_n + \frac{1}{C_n} i_t \quad (2.7)$$

Where V_{oc} is the OCV of the cell, that may vary with SOC and other operating conditions like current and temperature, V_t is the terminal voltage, V_1 is the voltage level at the R_1 and C_1 terminals, V_n is the voltage level at the terminals of the n th RC pair, R_s is the cell's internal series resistance and i_t is the current flowing through the cell.

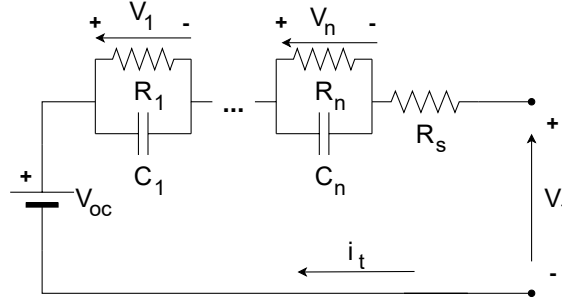


Figure 2.26: nRC model.

Multiple studies ([18, 20, 21, 22]) have been conducted to evaluate the performance of all these equivalent models presented. In all the studies reviewed, the nRC model with two RC pairs was shown to have the best performance out of all the models tested.

Fractional-order electric equivalent models

Electrochemical impedance spectroscopy (EIS) is a tool used for analyzing the condition and the electrochemical characteristics of the cell, where a small sinusoidal signal with varying frequency is applied to the cell, and resulting current is measured [23]. Using this tool, we can generate the EIS diagram to get the value of the lithium-ion cell impedance. The RC networks used in the nRC model are not adequate to emulate the battery characteristic throughout the whole frequency range, so to address this issue, constant phase elements (CPEs) are used instead of capacitors, more capable of representing the battery characteristic throughout the whole frequency range. In these models, the resistor and CPE pairs are used to emulate the charge transfer between the physical materials on the battery. Another element, referred as the Warburg impedance, is used to model the diffusion behaviour of the carriers [16, 19, 23]. In Figure 2.27, an example of a fractional order model is presented.

These models, compared to the nRC models, can achieve higher accuracy and are more capable of representing the physical phenomena occurring in the battery. However, due to their complex structure, complicated parameter identification process and slower simulation time, these methods are not as suitable for real-time online implementations in BMS as the nRC models [16, 19, 23].

Many different types of configurations can be used to model batteries using constant phase elements, as can be seen in [23].

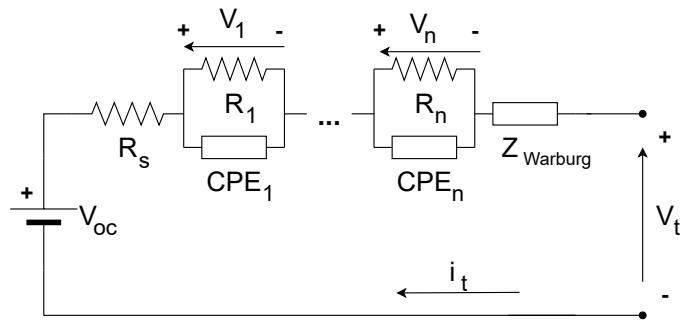


Figure 2.27: Example of a fractional order model.

2.4 State estimation algorithms

In this section, some state estimation algorithms will be briefly explained, along with some references to articles that employ those methods.

2.4.1 State-of-Charge algorithms

In this section, methods for SOC estimation will be reviewed, with special emphasis towards machine learning methods, which fit the main scope of the work to be done in this thesis.

As presented in Figure 2.28, these methods were divided in four groups: Conventional methods, Observer based methods, Adaptive filter methods and Machine learning methods [19, 24, 25, 26].

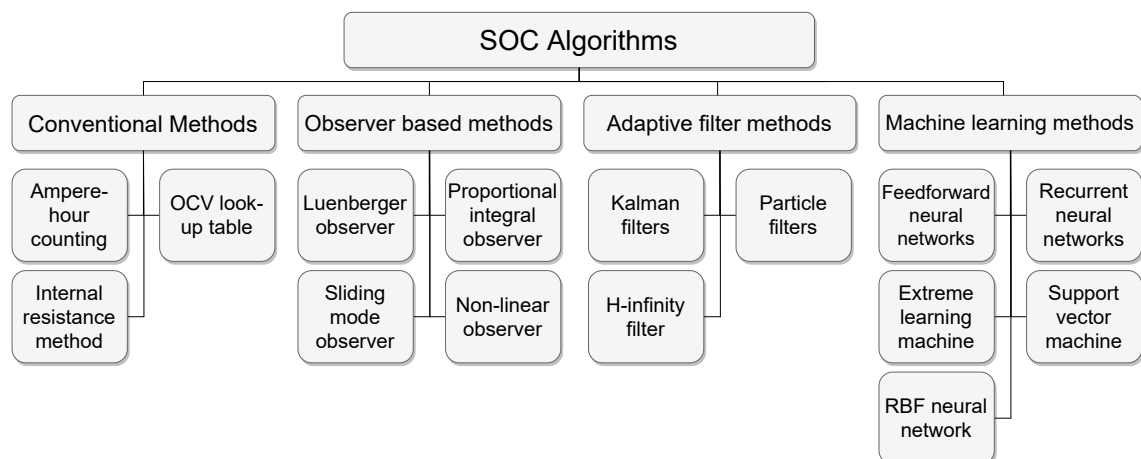


Figure 2.28: SOC methods reviewed in this work.

2.4.1.1 Conventional methods

Ampere-hour counting or *Coulomb Counting* method

One of the simplest methods used, *Coulomb Counting* consists of integrating the discharging or charging current on the battery cell, in order to calculate the accumulated or total charge that enters or leaves the battery [26].

In this method, SOC is calculated using the following equation:

$$SOC(t) = SOC(t_0) + \frac{1}{C_{max}} * \int_{t_0}^t \eta * i(\tau) d\tau \quad (2.8)$$

Where $SOC(t_0)$ is initial SOC at instant t_0 , C_{max} is maximum capacity of the battery, $i(\tau)$ is the current at the battery, positive when charging and negative when discharging, η is the coulombic efficiency, equal to one when the battery is discharging, and less than one when the battery is charging [9].

Great advantage of this method is its very low computational cost. Disadvantages include its inaccuracy when operating for longer periods of time, due to current sensor offset and drift errors, unknown initial SOC value, self-discharge rate and capacity fade [26]. In [27], an enhanced *Coulomb Counting* based method is proposed, in which the disadvantages of the traditional method are corrected, providing acceptable results after numerous cycles.

Open circuit voltage (OCV) method

In this method, the relation between SOC-OCV is used to estimate SOC of the battery cell, where a SOC-OCV curve is obtained experimentally and stored in the BMS. This method can achieve high precision, however a long rest time is needed before the battery reaches its relaxed state, so the OCV value can be accurately measured and estimate SOC based on that value. This resting time depends on current SOC and temperature, so it can vary from some minutes to several hours at a time. Another issue with this method, is that some chemistries of lithium-ion cells can have a very flat SOC-OCV curve in the middle regions, such as the $LiFePO_4$ batteries, as can be seen in Figure 2.29 and compared to another chemistry's OCV-SOC curve in Figure 2.30. This can lead to high error when estimating SOC on those chemistries using only this technique [25].

Internal resistance method

In this method, battery voltage and current flowing through the battery are used to estimate the internal resistance of the battery. To calculate this value, BMS needs to measure the value of voltage and current at small time intervals (<10ms). Due to its typical very low value, in the order of miliohms, accurate voltage and current sensors are needed to accurately calculate internal resistance. The internal resistance value also changes slowly through the whole range of SOC values, making it difficult to observe this value. Due to these factors, internal resistance method is generally not suitable for EV applications [25, 26].

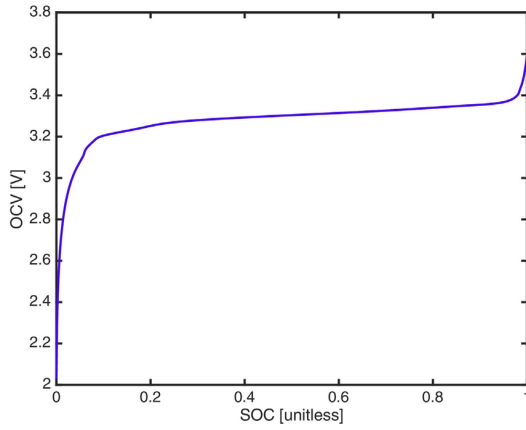


Figure 2.29: Typical OCV-SOC curve in LiFePO_4 cells, image from [28].

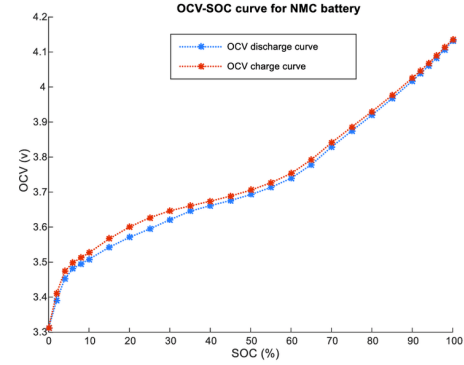


Figure 2.30: Typical OCV-SOC curve in NMC cells, image from [29].

2.4.1.2 Observer based methods

Luenberger observer

The Luenberger observer has been widely used to observe linear, non-linear and time varying systems, also finding its application in battery states estimation [19]. In [30], an adaptive Luenberger observer was used for online estimation of SOC on a lithium-ion battery pack, used in an EV. Results showed that this method can accurately estimate SOC without heavy computational load, with SOC error eventually converging to less than $\pm 2.5\%$. In [31], a fractional order electric equivalent battery model was used, similar to the model in Figure 2.27, combined with a Luenberger-type observer, to estimate battery SOC. The method was tested under different driving cycles, different aging levels and on different cells, being capable of accurately predicting the battery dynamics.

Proportional integral observer

The proportional integral observer (PIO) is an efficient method for state estimation in systems with unknown input disturbances. In this method, the measured voltage at the terminals of the battery is compared to the estimated voltage produced by the battery model, and the function of the PIO is to converge the estimated voltage, produced by the battery model, to the measured voltage, in a quick and precise way [19, 26]. In [32], a proportional integral observer is proposed to estimate the SOC of a lithium-ion battery, where a simple one RC battery model was used to model the battery. Under urban dynamometer driving schedule (UDDS) drive cycle, the SOC error obtained after convergence was never higher than $\pm 2\%$. The structure of the PIO used can be seen in Figure 2.31.

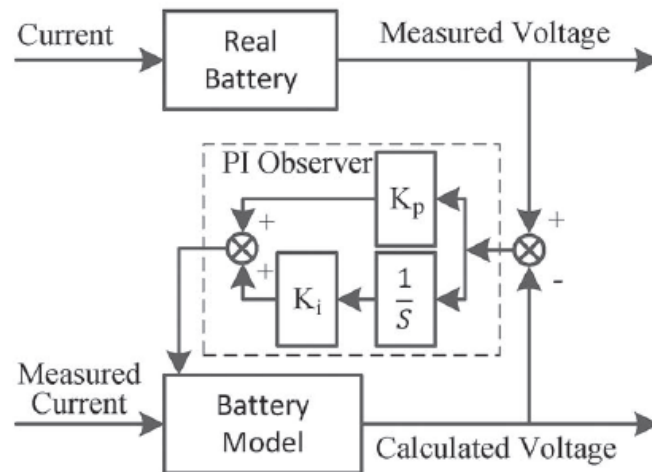


Figure 2.31: PI observer structure, image from [32].

Sliding mode observer (SMO)

The sliding mode observer has its origin in the sliding mode controller, maintaining its robust tracking performance despite possible model uncertainties and interference from external factors, such as environmental disturbances [19]. In [33], this method was first proposed, where the modeling errors caused by the battery electric equivalent circuit were compensated by the sliding mode observer implemented, showing robustness against model uncertainties and errors. In [34], an adaptive switching gain sliding mode observer (ASGSMO) is presented, which compared to traditional method SMO, has the capacity of minimising chattering levels in SOC estimation by utilizing a self-adjusted switching gain, while keeping the same characteristics of the traditional SMO method.

Non-linear observers

Non-linear observers have been used in multiple occasions to estimate the battery states. Similar to linear observer, but uses non-linear observation equations instead [26]. In [35], a non-linear observer for SOC estimation in EVs is presented, where the SOC estimation error eventually converged to 3% after wrong initial SOC value, and presented advantages over other Extended Kalman filter and Sliding mode observer methods in computational cost, convergence rate and estimation accuracy. In [36], a discrete non-linear observer is proposed, which presented the same advantages over EKF and SMO methods as the article in [35].

2.4.1.3 Adaptive filter based methods

Kalman filter methods (KF)

The Kalman filter approaches for state estimation in batteries have been widely used and researched. The Kalman filter algorithm is a recursive process with two steps, first step is to predict the system state and output, second step is to update the system state considering output error between the predicted output of the algorithm and the actual output. This behaviour highlights its most attractive feature, which is its self correcting nature. Since the battery OCV function is non-linear, the base Kalman filter cannot be directly used [19, 26]. The Kalman filter algorithm is described as the following procedure [37]:

1. Define linear state-space model for the system:

$$\begin{aligned}x_{k+1} &= A_k x_k + B_k u_k + w_k \\ y_k &= C_k x_k + D_k u_k + v_k\end{aligned}$$

Where x_k is the state vector, y_k is the system output vector, u_k is the input vector, w_k and v_k are the process noise and measurement noise, respectively, modeled as independent Gaussian white noise processes with zero-mean value.

2. Initialization, $k = 0$:

$$\begin{aligned}\hat{x}_0^+ &= \mathbb{E}[x_0] \\ P_{\hat{x},0}^+ &= \mathbb{E}[(x_0 - \hat{x}_0^+)(x_0 - \hat{x}_0^+)^T]\end{aligned}$$

Where $\hat{x}_0^+$ is the state estimation vector and $P_{\hat{x},0}^+$ is the state estimation error covariance, initialized with the best available information.

3. Computation process, $k = 1, 2, \dots$:

$$\begin{aligned}\text{State estimation time update: } \hat{x}_k^- &= A_{k-1} \hat{x}_{k-1}^+ + B_{k-1} u_{k-1} \\ \text{State error covariance time update: } P_{\hat{x},k}^- &= A_{k-1} P_{\hat{x},k-1}^+ A_{k-1}^T + Q_w \\ \text{Kalman gain matrix update: } L_k &= P_{\hat{x},k}^- C_k^T [C_k P_{\hat{x},k}^- C_k^T + R_v]^{-1} \\ \text{State estimation measurement update: } \hat{x}_k^+ &= \hat{x}_k^- + L_k [y_k - C_k \hat{x}_k^- - D_k u_k] \\ \text{State error covariance measurement update: } P_{\hat{x},k}^+ &= (I - L_k C_k) P_{\hat{x},k}^-\end{aligned}$$

Where Q_w and R_v are the covariance values of the noises w_k and v_k , respectively. At the end of iteration k , $\hat{x}_k^+$ holds the state estimation vector values and $P_{\hat{x},k}^+$ holds the state estimation error covariance, to be used in the beginning of the next iteration.

In [38], the Kalman filter was applied on a simple battery electrical equivalent model, managing to achieve SOC estimation error less than 5%. In [39], the Kalman filter method combined

with the open-circuit voltage and *Coulomb Counting* methods was implemented, to compensate for non-ideal factors present during operation. The method managed to achieve SOC error of $\pm 1.76\%$.

The Extended Kalman filter is an extension over the original Kalman filter method, which enables its direct use in non-linear applications, making it more suitable for state estimation in batteries. Uses first order Taylor series expansion and partial derivatives to linearize the cell model at each iteration. This method can suffer due to linearization errors in highly non-linear systems [25]. In [40], the EKF and the dual-EKF algorithms were employed for SOC estimation in two different commercial lithium-ion batteries, obtaining maximum SOC estimation error of 4%. In [41], EKF method was used on a second order RC electrical battery model, obtaining maximum SOC error of 4.2%. In [42], an Adaptive EKF (AEKF) was proposed for SOC estimation using a second order RC battery model, obtaining maximum SOC error of 3%. In [43], another AEKF method is implemented, where an Urban Dynamometer Driving Schedule (UDDS) test was used to validate the capabilities of the method, obtaining a maximum SOC estimation error of 2%.

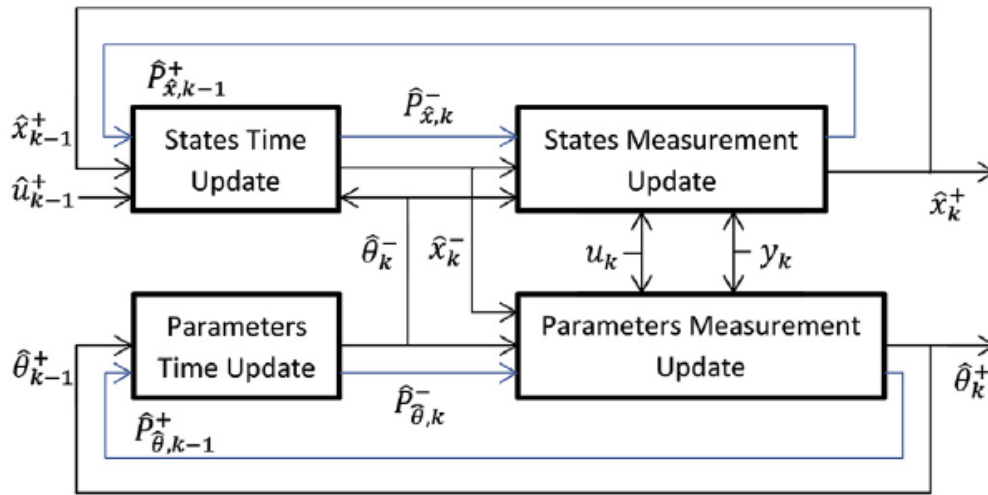


Figure 2.32: Dual-EKF structure for state and model parameters estimation, image from [40].

The EKF method applied to a highly non-linear model can result in significant errors due to linearization errors, so another possible solution for SOC estimation is to use the UKF method. The UKF is a type of Sigma Point Kalman Filters (SPKF), and is an upgraded version of the EKF, that performs the unscented transformation and discrete-time filtering algorithm to solve filtering problems, making it more suitable for highly non-linear systems and proved to be more effective in battery state estimation than the EKF [19]. In [44], the UKF method was used to adjust the model parameters and estimate SOC, the model was evaluated using data from LiFePO₄ batteries cycled under federal driving schedule and dynamic stress testing conditions. In [45], the EKF, AEKF, UKF and AUKF methods were implemented on a battery modeled through an extreme learning machine (ELM), where the results obtained showed that the AEKF and AUKF have the best convergence rate and the AUKF the highest accuracy.

Main disadvantage of the Kalman filter methods, is that process and measurement noises are modeled as white noise Gaussian distributions with zero mean value. In real applications, such as in a battery, this assumption may be wrong, possibly leading to problems in filter convergence and method accuracy. Other methods that don't face this issue are the particle filter based methods, which can achieve higher accuracy and robustness, at the cost of higher computational effort.

Particle filter method (PF)

The PF methods, also referred as sequential Monte Carlo methods, allow full representation of the states posterior distribution, enabling easy computation of any statistical estimates, allowing the method to deal with non-linear systems [46]. In many cases, it has proven to be a more accurate method than KF based ones, at the cost of additional computational effort [26]. A generic particle filter algorithm is described in [46]:

1. Sequential importance sampling step:

1. For $i = 1, \dots, N$, sample $\tilde{x}_k^{(i)} \sim q(x_k | x_{0:k-1}^{(i)}, y_{1:k})$, update trajectories $\tilde{x}_{0:k}^{(i)} \triangleq (\tilde{x}_k^{(i)}, x_{0:k-1}^{(i)})$
2. For $i = 1, \dots, N$, calculate importance weights up to normalizing constant

$$w_k^{(i)} = \frac{p(\tilde{x}_{0:k}^{(i)} | y_{1:k})}{q(\tilde{x}_k^{(i)} | x_{0:k-1}^{(i)}, y_{1:k}) p(\tilde{x}_{0:k-1}^{(i)} | y_{1:k-1})}$$

3. For $i = 1, \dots, N$, normalize weights: $\tilde{w}_k^{(i)} = w_k^{(i)} [\sum_{j=1}^N \tilde{w}_k^{(j)}]^{-1}$

2. **Selection step:** Multiply samples $\tilde{x}_{0:k}^{(i)}$ with high importance weights $\tilde{w}_k^{(i)}$, suppress samples $\tilde{x}_{0:k}^{(i)}$ with low importance weights $\tilde{w}_k^{(i)}$, to generate N random samples $\tilde{x}_{0:k}^{(i)}$ closely distributed according to $p(\tilde{x}_{0:k}^{(i)} | y_{1:k})$.
3. **MCMC step:** Apply a Markov transition kernel, with an invariant distribution produced by $p(x_{0:k}^{(i)} | y_{1:k})$ to generate $x_{0:k}^{(i)}$.

Where $x_{0:k}^{(i)}$ are a set of N weighted particles/samples, $i = 0, 1, 2, \dots, N$, that approximate posterior distribution $p(x_{0:k} | y_{1:k})$, and are sampled from an importance proposal distribution $q(x_{0:k} | y_{1:k})$. Full explanation presented in [46].

In [47], a PF was used for SOC estimation, where two models were used to represent the system dynamics, one was the state transmission model and the other a measurement model, which described the relation between SOC and the different measured values such as terminal voltage, current and temperature. Results showed the method is effective and significantly faster than the EKF method also implemented in that paper. In [48], a Cubature PF was proposed, where it obtained better accuracy and robustness than the Cubature KF implemented in that paper. In [49], an Unscented PF was proposed for SOC estimation, where it was evaluated by comparing it to other EKF, UKF and PF methods, and shown improvement over the UKF method.

H ∞ filter

The H ∞ filter is a simply designed model, which under the right conditions can provide strong robustness. This method's accuracy can be compromised due to battery aging, hysteresis and temperature effects [25]. In [50], the Adaptive H ∞ filter is proposed for SOC estimation in lithium-ion battery, where it performed better in accuracy, time efficiency and computational cost than the AEKF and the square-root UKF methods. In [51], model parameters and SOC were updated using dual H ∞ filters, with the method resulting in maximum SOC estimation error of 1%.

2.4.1.4 Machine learning methods

Feedforward neural networks (FNN)

Feedforward neural networks have one of the simplest structures of the existing neural networks, with the capacity to model any system without any previous knowledge of the system, due to its adaptability and learning capacity, making it an attractive choice to model non-linear systems such as a battery. In Figure 2.33, a typical FNN structure is presented.

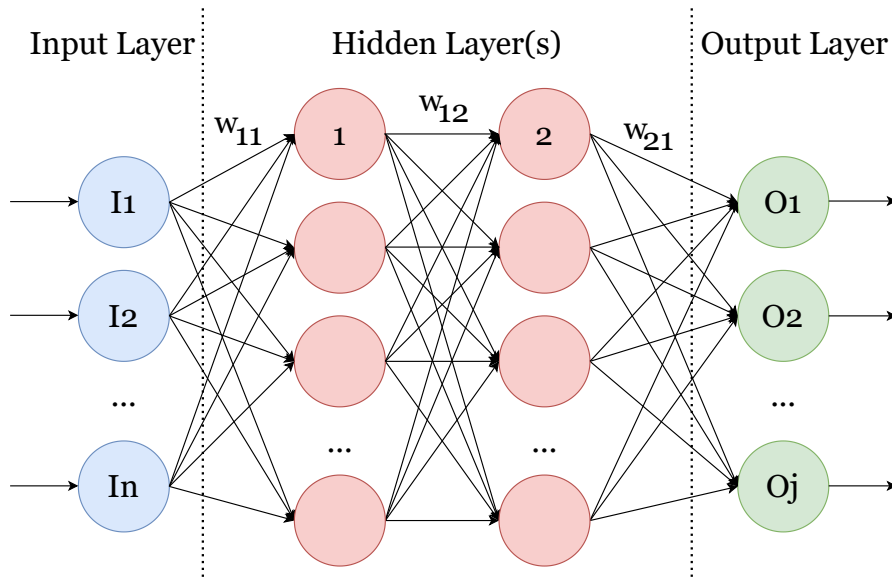


Figure 2.33: Feedforward neural network structure.

The FNN presents three layers: input layer, output layer and one or more hidden layers, in which all can have any number of neurons. An additional bias neuron is added to each layer, which is a special type of neuron that always outputs a value of one. A non-linear activation function must be chosen for the hidden layer and output neurons. Two common choices for the activation functions are the *sigmoid* function (2.10) and the *tanh* (hyperbolic tangent) function (2.11). Each connection between neurons (arrows in the image) has an associated weight, which determines how much of the preceding neuron's output value is passed to the connected neuron of the next layer. Each neuron produces an output value based on the output values and weights

associated to all the neurons in the previous layer [24, 52]. The output y_j of a neuron j in the hidden or output layer can be given by:

$$x_j = \sum_{i=1}^{n_i} (w_{ji} * y_i) + \theta_j \quad (2.9)$$

$$y_j = \sigma(x_j) = \frac{1}{1 + e^{-x_j}} \quad (2.10)$$

$$y_j = \tanh(x_j) = \frac{2}{1 + e^{-2x_j}} - 1 \quad (2.11)$$

Where x_j is the net value, calculated before applying the activation function to determine the output value; w_{ji} is the weight value of the connection from a neuron i in the previous layer to the neuron j ; y_i is the output value of the neuron i in the previous layer; n_i is the number of neurons in the previous layer; and θ_j is the bias value of the neuron j [52, 53].

Network training is commonly done using the back-propagation algorithms, to adjust the weights of the network. One of the most common algorithms used is the gradient descent method. In this algorithm, an error function is computed at each training iteration, by comparing the expected output with the generated output of the network. The weights are then adjusted as to minimize this error. Training is stopped when the loss value is sufficiently low or a max number of *epochs* have been reached. The error function used for training the network is usually the squared error function (2.12) [53, 54]:

$$E = \frac{1}{2} * \sum_{i=1}^{n_o} (\bar{y}_i - y_i)^2 \quad (2.12)$$

Where $\bar{y}_i$ is the expected output, y_i is the network output, n_o is the number of outputs and E is the loss value. Each weight in the network is then adjusted in an attempt to minimize this error value E , by adjusting each weight in the opposite direction of the loss function gradient [53]:

$$w_i^+ = w_i + \Delta w_i \quad (2.13)$$

$$\Delta w_i = -\alpha \frac{\partial E}{\partial w_i} \quad (2.14)$$

Where w_i^+ is the new calculated weight, Δw_i is the variation in the weight and α is the defined learning rate.

In [53], a Deep FNN was used for direct SOC estimation with the battery measurements fed as inputs for the network. Dataset used in training included battery data recorded at different temperatures between -20°C and 25°C. Tested in different dynamic driving cycle profiles at different and dynamic temperatures, obtaining MAE of 1.10% at 25°C and MAE of 2.17% at -20°C, with the lowest MAE obtained of 0.61% when tested with HWFET data.

In [55], a FNN was trained to correlate battery SOC to its OCV, where the FNN is fed the OCV as input and outputs SOC. To estimate OCV and feed it to the network, a 1 RC pair equivalent circuit model was used, which estimated OCV based on the measured battery terminal voltage. The equivalent circuit model parameters were online estimated using a forgetting factor recursive least-squares technique. Model was trained and tested under different driving cycle profiles at the same temperature 25°C, obtaining MAE of 0.3325% when tested with the FUDS data.

In [56], a FNN based battery model was developed to estimate SOC based on battery voltage and current, combined with an UKF used to reduce errors in the FNN based SOC estimation. The method was tested under multiple temperatures and different driving profiles, obtaining lowest RMSE of 1.4% when tested with the FUDS data.

Extreme learning machine (ELM)

ELM has a very similar structure to the FNN, but the difference comes in its training method, since the ELM uses the Moore-Penrose generalized inverse or pseudo-inverse matrix, instead of the back-propagation algorithm [24].

In [57], an ELM was used for SOC estimation, where a gravitational search algorithm (GSA) was used to optimize the number of hidden layer neurons of the ELM. Model was trained and tested under multiple driving cycle schedules at temperatures 25°C and 45°C, obtaining lowest MAE of 0.48% when tested with the BJDST data at 45°C.

Radial basis function (RBF) neural networks

The RBF network has a very similar structure to FNN, but has only a single hidden layer, and the activation function used for the neurons in the hidden layer network is a radiated Gaussian kernel function [24], where the output of the i hidden neuron and the output of the k output neuron are described by:

$$\phi_i(x) = G(\|x - w_i\|) = \exp\left(-\frac{\|x - w_i\|^2}{\sigma_i^2}\right) \quad (2.15)$$

$$y_k = \sum_{i=1}^j W_{ik} * \phi_i(x) \quad (2.16)$$

Where $\phi_i(x)$ is the output of the i hidden neuron, x is the input vector of the network, w_i represents the center value of the i hidden neuron, σ_i the width of the i hidden neuron, y_k the output of the k output neuron, W_{ik} the associated weight of the connection between i hidden neuron and the k output neuron, and j the number of hidden neurons [52].

In [58], a RBF network combined with the AEKF method were used to estimate SOC, tested in multi-cell battery pack. Trained and tested under DST and HPPC current profiles at a single temperature, obtained maximum SOC estimation error of 2%.

In [59], a RBF network combined with the H^∞ method were used to directly estimate SOC. Tested under constant current charging conditions at a single temperature, the proposed method obtained RMSE of 0.7%.

Support vector machine (SVM)

The SVM is a model used to solve classification and regression problems. In battery SOC estimation, a regression learning method is needed to minimize error function, so the regression variation of SVM, support vector regression, can be used to create a battery model for state estimation [24, 52].

In [60], a SVM model is implemented for SOC estimation. The model was trained with data obtained from the battery under a dynamic current profile, and tested under DST profile conditions at a single temperature level, managing to obtain a RMSE of 0.4%.

Recurrent neural networks (RNN)

RNN are a type of neural network which uses information of input, output and/or hidden states used in previous iterations. Neural networks are made recurrent by passing a hidden state, output, or past inputs/outputs as input to the network, creating a loop in the network, as can be seen in Figure 2.34.

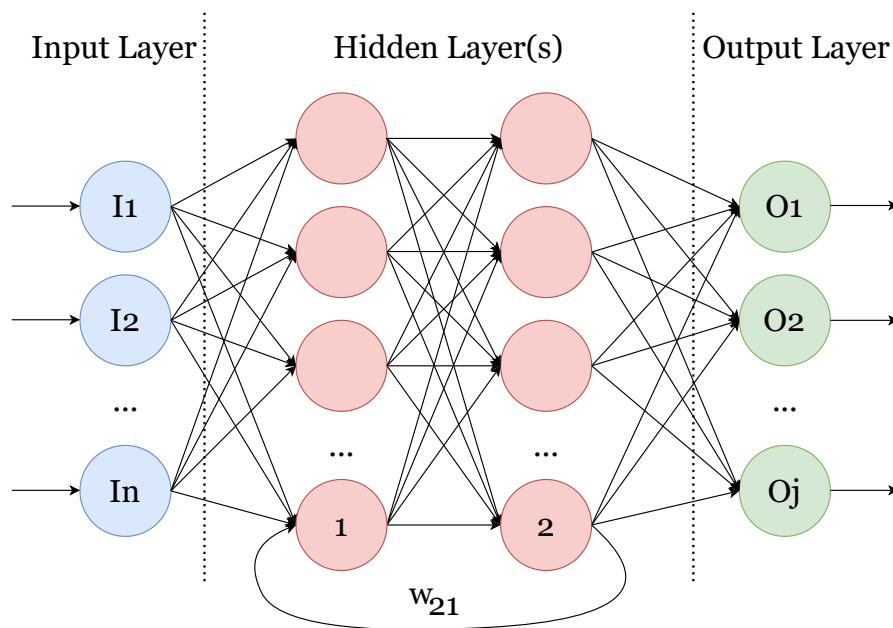


Figure 2.34: Recurrent neural network structure.

The adding of the connection described by the weight w_{21} turned the FNN depicted in Figure 2.33 into a RNN. This type of network proves to be very effective in predicting non-linear systems and dealing with short time dependencies of the inputs, however it can't deal with long-term dependencies and suffers from the vanishing gradient problem during back-propagation training.

To deal with long-term dependencies, Gated RNN can be used, which add the ability to store older and relevant information to its state, making these methods more appropriate for problems that require longer sequential data, such as speech recognition, language modeling, image captioning, and in this case could be applied to state estimation in batteries [24].

Non-linear autoregressive with exogenous input (NARX) neural networks are a type of RNN, which use past values of the output of the network as inputs to the network [52]. In [61], a Dynamically Driven Recurrent neural network (DDRN), which uses the NARX architecture, is proposed for SOC and SOH estimation. The network is tested on two different li-ion batteries, a lithium iron phosphate (LiFePO_4) and a lithium-titanate (LTO) battery. The network was tested under dynamic charge and discharge conditions at multiple temperatures, obtaining lowest RMSE of 0.3045% at 0°C on the LiFePO_4 battery. In [62], a NARX network is proposed for SOC estimation, where a lighting search algorithm (LSA) was used to optimize input delays, feedback delays and number of hidden layer neurons on the network. The network was tested using FUDS and US06 drive cycles at multiple temperatures, obtaining lowest RMSE and MAE of 0.52% and 0.34%, respectively, when tested on the FUDS drive cycle at temperature 45°C.

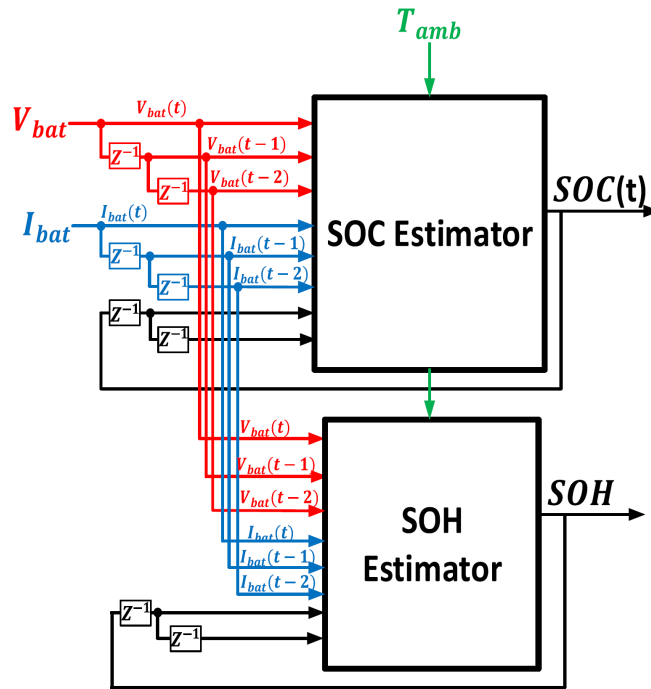


Figure 2.35: DDRN based SOC and SOH estimation method, image from [60].

Long short-term memory (LSTM) neural networks are a type of Gated RNN, which use memory units, the LSTM layer, instead of the usual neurons in the hidden layer, enabling it to store and transfer information from previous iterations and current iteration, to be used in future iterations [24, 52]. Each unit has a memory cell c_k , which stores relevant information from past and current iterations. The memory cell value is altered at each iteration by the value of the hidden state h_{k-1} ,

received from the previous iteration, and the input vector X_k . The LSTM unit decides how much of the hidden state value h_{k-1} and input X_k is going to be written to the memory cell, by the use of two gates: the forget gate and the input gate. The forget gate will decide how much of the current value stored in the memory cell c_{k-1} is going to be forgotten/deleted, and the input gate decides how much of the value of h_{k-1} and X_k is going to be written to the memory cell. A third gate is used, the output gate, which determines how much of the value stored in the memory cell c_k will be passed as new hidden state h_k , to be used in the next iteration [63]. Figure 2.36 depicts the typical structure of the LSTM units used in LSTM networks:

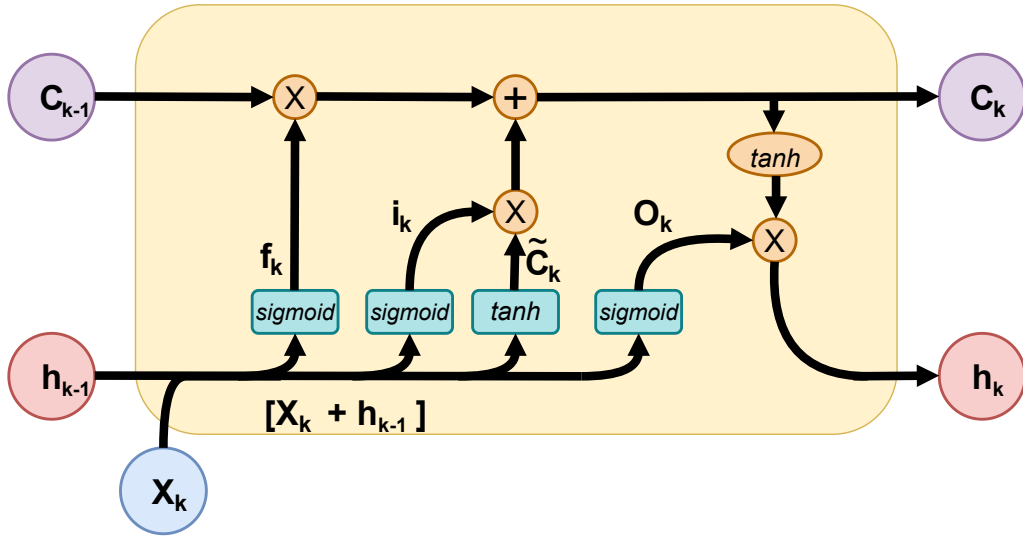


Figure 2.36: LSTM layer unit.

The *sigmoid* blocks represent *sigmoid* neural network layers, which receive the values of input vector X_k and hidden state h_{k-1} and output a value between $[0, 1]$, using the *sigmoid* activation function (2.10). Each gate is composed of a *sigmoid* block connected to a multiplication block. The idea is that each *sigmoid* block will output a value between $[0, 1]$, to multiply the other input value at the multiplication block, deciding how much of that value will be passed on. f_k , i_k and O_k are the output values of these blocks associated to the forget gate, input gate and output gate, respectively. $\tilde{c}_k$ is the candidate value for c_k , whose value will be added to the memory cell according to the value of i_k . The *tanh* block, that outputs candidate value $\tilde{c}_k$, represents a *tanh* neural network layer, which uses the *tanh* activation function (2.11). Each of these neural network layers have a set of associated weights, to be adjusted during the training phase [63].

In [63], a LSTM network was used to directly estimate SOC based on measured voltage, temperature and current on the battery. The network was tested with dynamic drive cycles at multiple temperatures, obtaining MAE of 0.573%. In [64], a stacked LSTM network was proposed to directly estimate SOC through voltage, temperature and current measurements. The network was trained and tested on multiple driving cycles at a single temperature, with multiple different initial SOC values, and managed to obtain 0.84% MAE under the FUDS driving cycle.

Bidirectional long short-term memory (BiLSTM) neural networks are a type of LSTM, in which each BiLSTM layer is now composed of two LSTM layers, where in one LSTM layer, called the forward layer, the input vectors are fed sequentially from the earliest time step available to the current time step, while in the other layer, called the backwards layer, the input vectors are fed sequentially from the current time step to the earliest time step available, meaning that each layer is processing the state and inputs in a reverse order in comparison with each other. The output at a given iteration will be a combination (concatenate, multiply, sum or average) of the hidden state of both layers. This new structure makes the method capable of analyzing dependencies from both temporal ends of data provided. This type of network finds a great use in text translation, where the final words of a sentence can alter the context of the whole sentence [24]. The Figure 2.37 depicts the structure of a BiLSTM, unfolded across three time steps.

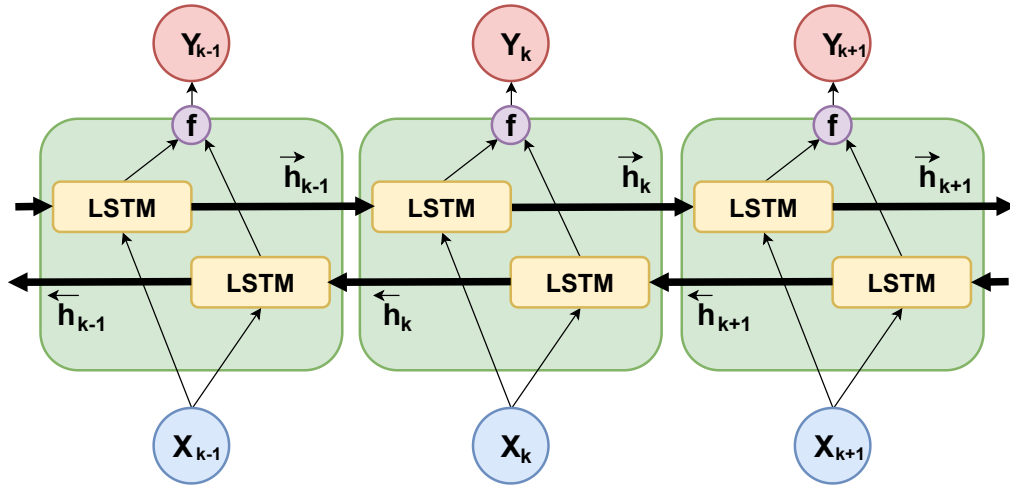


Figure 2.37: BiLSTM structure and function, unfolded in time.

$\vec{h}_k$ represents the hidden states of the forward LSTM layer, and $\overleftarrow{h}_k$ represents the hidden states of the backwards LSTM layer. The output Y_k is a direct result of $f(\vec{h}_k, \overleftarrow{h}_k)$, where $f()$ is a function used to combine the two hidden states. The value of $\vec{h}_k$, in the forward layer, will be a function of the previous iteration hidden state $\vec{h}_{k-1}$, previous iteration input vector X_{k-1} and current iteration input vector X_k , while the value of $\overleftarrow{h}_k$, in the backwards layer, will be a function of the next iteration hidden state $\overleftarrow{h}_{k+1}$, next iteration input vector X_{k+1} and current input vector X_k , so the output value Y_k will depend on information from the time steps $k-1$, k and $k+1$ [65].

In [65], a stacked BiLSTM network was proposed for SOC estimation. The method was tested on multiple drive cycles and at multiple temperatures, managing to obtain a MAE value of 0.54% under the BJDST drive cycle at 45°C.

Gated recurrent unit (GRU) neural networks are a type of Gated RNN, which use only two gates, the update gate and the reset gate. The update gate is used instead of the two gates input gate and forget gate used in the LSTM, fulfilling the same purpose and resulting in a simpler model. The reset gate determines how much of the previous hidden state will be passed as possible candidate for the new hidden state value. Another difference compared to the LSTM, is that the

cell state and the hidden state are now merged into only one state, the hidden state. The LSTM and GRU networks work similarly and are able to produce similar results [24, 52]. Figure 2.38 depicts the structure of a GRU layer.

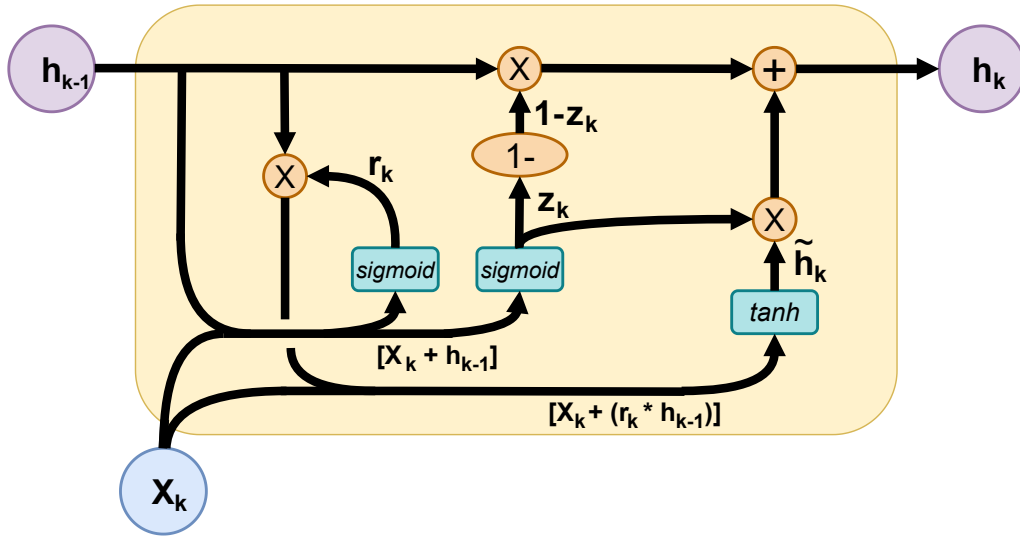


Figure 2.38: GRU layer unit.

In Figure 2.38, r_k and z_k are the associated values of the reset gate and update gate, respectively. The value $1 - z_k$ will be used similarly as in the forget gate in LSTM, where it will determine how much of the previous hidden state will pass to the new hidden state, and the value z_k will be used similarly as in the input gate in LSTM, where it will determine how much of the hidden state candidate value $\tilde{h}_k$ will pass to the new hidden state value [66].

In [67], a GRU network was implemented where SOC is directly estimated through voltage, current and temperature measurements. Trained and tested on different datasets, drive cycles and multiple temperatures, the method managed to obtain MAE of 0.32% under the LA92 drive cycle at 25°C. In [66], another GRU network was implemented on two batteries, a NMC and a LiFePO₄ battery, where SOC is directly estimated through voltage, current and temperature measurements. Tested under a DST profile at multiple temperatures, the method managed to obtain MAE of 0.90% at 40°C on the NMC battery used.

Evaluation and comparison of machine learning methods reviewed

The different methods reviewed are displayed in Table 2.2. In the lowest error achieved column, the lowest MAE or RMSE value obtained is shown, alongside the drive cycle and/or temperature at which that error was achieved. In the testing conditions column, the overall tests that were performed are displayed. In the method characteristics column, some advantages/disadvantages are shown regarding each method in general (FNN, ELM, RBF, NARX, LSTM and BiLSTM, GRU).

Although a general comparison is made between methods, many aspects of their particular implementation are not considered, such as: computational cost, memory usage and number of training repetitions.

In this sense, methods based in FNN and RNN appear to be the most promising machine learning techniques for SOC estimation, due to having the most accurate results when tested under multiple drive cycles and temperatures. These methods are also relatively simple in implementation, since it isn't required any additional method and SOC can be directly estimated based solely on voltage, temperature and current measurements.

Method	Lowest error achieved	Testing conditions	Method Characteristics
FNN [53]	MAE: 0.61% under HWFET	Multiple temperatures Multiple drive cycles	+ Flexibility + Simplicity
FNN [55] + ECM	MAE: 0.3325% under FUDS	Single temperature tests Multiple drive cycles	+ Good accuracy at different temperatures
FNN [56] + UKF	RMSE: 1.4% under FUDS	Multiple temperatures Multiple drive cycles	- Slow training process
ELM [57]	MAE: 0.48% under BJDST at 45°C	Temperatures 25°C and 45°C Multiple drive cycles	+ Good scalability + Good generalization capability + Lower computational cost + Good learning speed - Accuracy very sensitive to number of hidden layer neurons
RBF [58] + AEKF	MAX SOC error: 2%	Single temperature DST and HPPC current profiles	+ Adequate performance with limited information + Good global approximation behaviour
RBF [59] + H ∞	RMSE: 0.7%	Single temperature Constant current charging	- Training algorithm may get stuck on local minimum - Slow training process
SVM [60]	RMSE: 0.4%	Single temperature DST current profile	+ Good performance in high dimension non-linear systems + Accurate and fast estimation - High computational cost
NARX [61]	RMSE: 0.3045% at 0°C on LiFePO ₄	Multiple temperatures Dynamic charge and discharge	+ Effective use of training data + Good accuracy at different temperatures
NARX [62]	MAE: 0.34% under FUDS at 45°C	Multiple temperatures FUDS and US06 drive cycles	- Accuracy very sensitive to number of past inputs and outputs fed as inputs
LSTM [63]	MAE: 0.573%	Multiple temperatures Multiple drive cycles	+ Effective in dealing with long-term data dependencies
LSTM [64]	MAE: 0.84% under FUDS	Single temperature Multiple drive cycles	+ Lower computational cost in online phase - Complex training process
BiLSTM [65]	MAE: 0.54% under BJDST at 45°C	Multiple temperatures Multiple drive cycles	+ Good accuracy at different temperatures
GRU [67]	MAE: 0.32% under LA92 at 25°C	Multiple temperatures Multiple drive cycles	+ Effective in dealing with long-term data dependencies + Good accuracy at different temperatures
GRU [66]	MAE: 0.90% at 40°C on NMC battery	Multiple temperatures DST current profile	- Needs large amount of training data - Needs large storage machine

Table 2.2: Comparison of different machine learning methods reviewed [24, 52].

2.4.2 State-of-Health algorithms

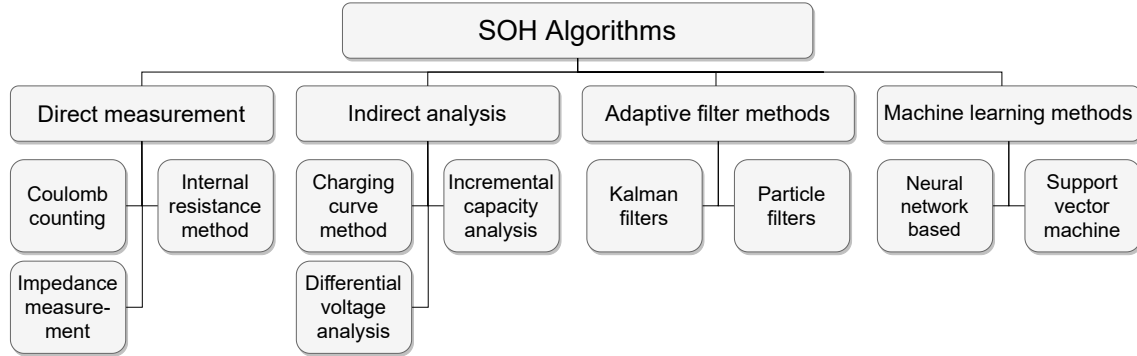


Figure 2.39: SOH methods reviewed in this work.

As displayed in Figure 2.39, all SOH methods reviewed were divided in four groups: Direct measurement methods, Indirect analysis methods, Adaptive filter based methods and Machine learning methods [68, 69, 70].

2.4.2.1 Direct measurement methods

Coulomb Counting method

Through current integration, charge entering and leaving the battery during charge or discharge operations can be measured, as seen in equation 2.8. If a full discharge of the battery is verified (from 100% SOC to 0% SOC), then the maximum available capacity of the battery can be simply calculated, since all the charge the battery can currently hold is known [68].

In [27], SOH is examined by analysing the depth of discharge (DOD) of the battery, defined as the value of the amount of charge released by the battery, divided by the rated max capacity of the battery. DOD is the complement value of SOC, can be defined as:

$$DOD(\%) = 100\% - SOC(\%) \quad (2.17)$$

Great advantage of this method is its very low computational cost. Disadvantages include its inaccuracy when operating for longer periods of time, due to current sensor error which will be integrated and accumulate error over time [70].

Internal resistance measurement

As a battery ages, its capacity will decrease and its internal resistance will increase. The internal resistance value can be associated to this capacity loss, and therefore can be used to estimate the SOH of the battery. The internal resistance can vary due to temperature and current SOC, so these operating conditions must be taken into account when mapping the internal resistance value to the SOH of the battery [70].

The internal resistance of the battery can be measured by applying charge/discharge pulse to the battery, as depicted in Figure 2.40. Measuring the immediate variation in terminal voltage, ΔU_t , and current, Δi_L , by applying ohm's law [68]:

$$R_i = \frac{\Delta U_t}{\Delta i_L} \quad (2.18)$$

Where R_i is the value of internal resistance.

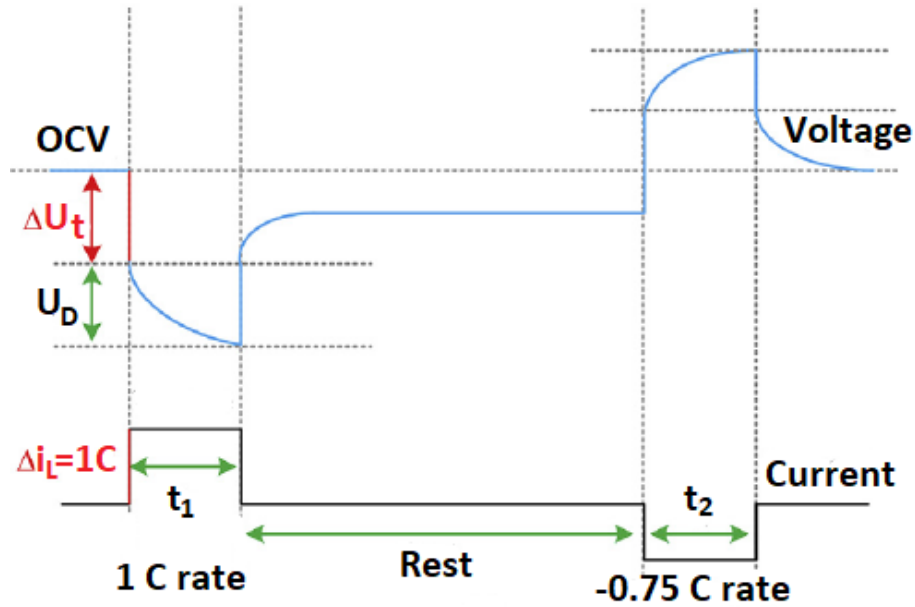


Figure 2.40: Voltage and current waveform when discharge/charge pulse is applied [68].

Impedance measurement

As the battery ages and its capacity decreases, its impedance increases, and can be related to the SOH of the battery. Using the Electrochemical Impedance Spectroscopy (EIS) method, the impedance values of the battery can be obtained, and mapped to parameters of an equivalent model of the battery, to be used for SOH estimation [70]. In [71], the parameters of an equivalent circuit model were extracted from the battery using EIS technique, and then used the relation between internal resistance estimated and the available capacity of the battery, to estimate SOH.

2.4.2.2 Indirect analysis methods

Charging curve method

The curve waveform when a charging pulse is applied to the battery can be used for SOH estimation, because as the battery degrades, its charging curve waveform will change as well [68]. In [72], a SOH estimation technique was implemented, where a constant voltage (CV) step is used as

a health indicator of the battery. In this article, the battery current behaviour during the constant voltage step was approximated by:

$$I(t, C_{loss}) = A(C_{loss})e^{-B(C_{loss})t} + C(C_{loss}) \quad (2.19)$$

Where C_{loss} is the capacity loss of the battery, t is the time value, and A, B, C are parameters relating capacity loss to the resulting current [72].

Incremental Capacity Analysis (ICA)

ICA is an analysis technique which has been used for SOH estimation. In this method, incremental capacity (IC) curves are generated by integrating capacity related to small voltage intervals $\frac{dQ}{dV}$, as seen in Figure 2.41, by charging/discharging a battery with very small current. Through analysis of the peaks obtained in these IC curves, SOH estimation can be made [68]. In [73], the battery aging mechanism was analyzed using an incremental capacity method, where the normalized incremental capacity peak obtained was used to estimate the current capacity of the battery.

Differential Voltage Analysis (DVA)

DVA is another analysis technique which has been used for SOH estimation. In this method, differential voltage (DV) $\frac{dV}{dQ}$ curves are plotted, as seen in Figure 2.42, where through analysis of the peaks obtained, SOH estimation can be made [68]. In [74], a DVA method was used for SOH estimation, where a new method using a center least squares algorithm was used to obtain the DV curve.

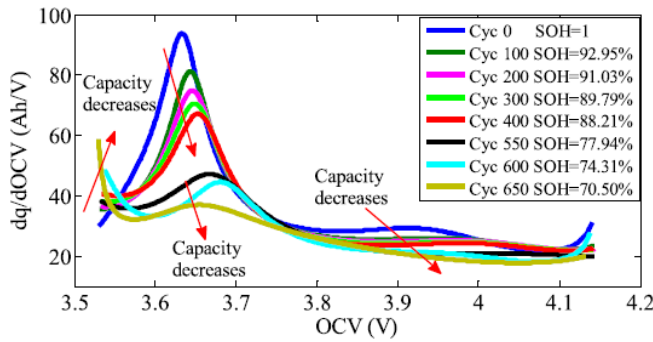


Figure 2.41: ICA curve [73].

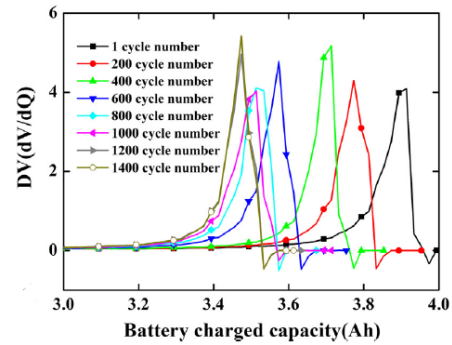


Figure 2.42: DVA curve [74].

2.4.2.3 Adaptive filter based methods

Kalman filters

Kalman filter algorithm and different types have been described in the previous Section 2.4.1.3. For SOH estimation, usually the Kalman filter method is used to estimate battery parameters, such as the internal resistance and capacity, to calculate the SOH [70]. In [75], a Dual-EKF method is

used for SOC and capacity estimation, separately, where the method achieved good accuracy. In [76], the UKF method was used to estimate SOC, internal resistance and capacity of the battery, which reflect the battery SOH.

Particle filters

The particle filter algorithm has been described in the previous Section 2.4.1.3, and has been used in SOH prediction for capacity and internal resistance estimation [69]. In [77], a particle filter method was implemented for SOC and SOH estimation, where the method takes the estimated SOC as input for estimating the SOH of the battery, showing good accuracy.

2.4.2.4 Machine learning methods

Neural network based methods

Neural network algorithms have been described in the previous Section 2.4.1.4. Here, the principle is the same, except the network is trained with relevant data for SOH estimation, and the network will provide an estimation for SOH. Many methods have been implemented which used neural networks for SOH estimation, where accurate results were achieved in different operating conditions [69].

In [78], a Long Short Term Memory (LSTM) network was used for SOH estimation, where time varying values of voltage and current are fed as inputs to the network, sampled during a short period. The method managed to achieve average error of 2.46% in SOH estimation.

In [79], a FNN was used for SOH estimation, where the network received as inputs the terminal voltage of the cells and the rate of change of the terminal voltage $\frac{dV}{dt}$, and outputted the battery SOH. The method was tested on different batteries, achieving a MAE of less than 1%.

In [61], a Dynamically Driven Recurrent Neural network (DDRNN) was used for SOC and SOH estimation, where past values of SOH were fed as inputs to the network, along with temperature, current, voltage, and past values of current and voltage, as seen in 2.35. Good results were obtained when tested under dynamic charge and discharge profiles.

In [80], a Sparse Bayesian Predictive Modeling (SBPM) algorithm was used to learn the relation between the capacity of the battery and a voltage sequence sample entropy. The SBPM uses the idea of radial basis functions, as described in Section 2.4.1.4, in its design. The schematic for obtaining the battery SOH model is shown in Figure 2.43. The method showed accurate results in capacity estimation of the battery.

Support vector machine (SVM)

SVM method has been described in the previous Section 2.4.1.4. For SOH estimation, the method is used for capacity and/or internal resistance estimation of the battery. In [81], a SVM based

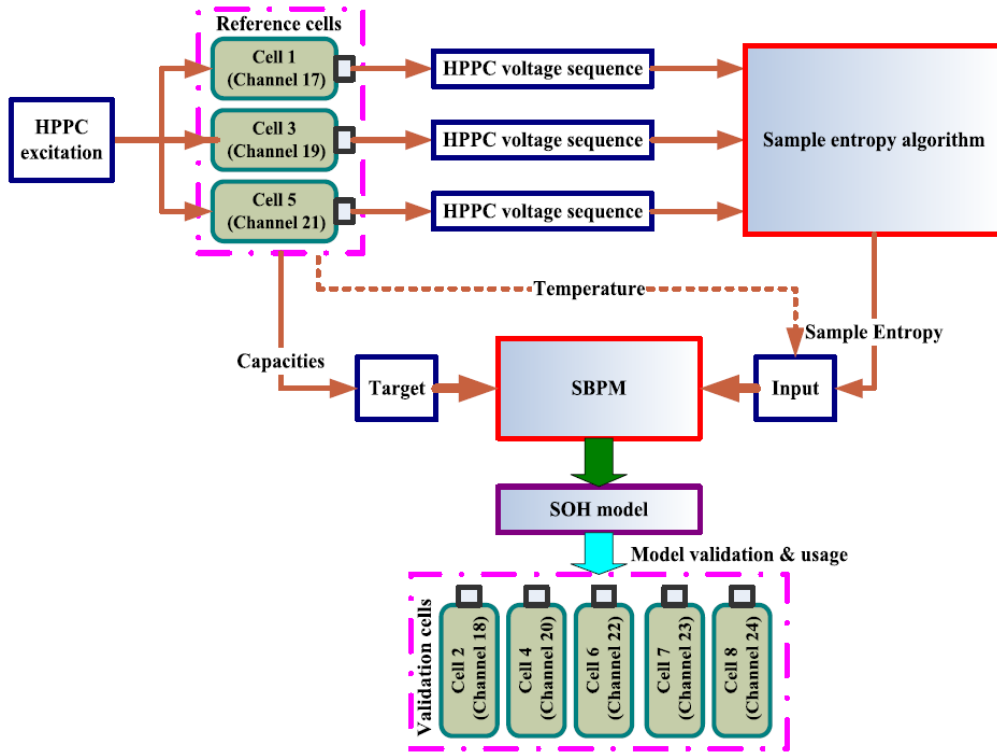


Figure 2.43: SBPM model establishment schematic, image from [80].

battery model was developed to estimate capacity and internal resistance of a battery for SOH estimation, using voltage, temperature and current measurements. The method demonstrated a good accuracy in SOH estimation when tested under different temperatures and operating conditions.

2.4.3 Power prediction

Power prediction is of vital importance in EVs BMS, since knowledge about the available charge and discharge power helps the energy management system (EMS) of the vehicle to make informed control decisions, to prevent damage to the battery and sudden power drops during vehicle acceleration [10].

In this sense, two equations can be used to describe available discharge power capability and charge power capability in a single cell [10]:

$$P_{discharge}(t) = [V_{min} * (V_{OCV}(t) - V_{min}) / R_i] \quad (2.20)$$

$$P_{charge}(t) = [V_{max} * (V_{max} - V_{OCV}(t)) / R_i] \quad (2.21)$$

Where V_{min} is the rated minimum voltage allowed for the cell, V_{max} is the rated maximum voltage allowed for the cell, V_{OCV} is the open circuit voltage of the cell, dependant on SOC and other operating conditions like current and temperature, t is the time value, and R_i is the internal

resistance of the cell varying with temperature, SOC and SOH. We can also see, based on these equations, that the power fade verified over a battery lifetime is mainly due to an increase in its internal resistance [10].

In this section, some power prediction methods were reviewed and divided in two groups: Characteristic maps and Model based methods [10, 19, 82, 83].

Characteristic maps

These methods use known dependencies between power capability and other battery states and parameters, such as SOC, temperature, battery voltage, and others, to build characteristic maps that relate these battery parameters and states to the power capability of the battery. These characteristic maps are built by collecting experimental data from the battery, using methods such as the hybrid pulse power characterization (HPPC), or performing electrochemical impedance spectroscopy (EIS) tests. However, due to battery aging, these initially obtained characteristic maps cannot depict SOP accurately throughout the battery use, therefore adaptive characteristic maps need to be used for online BMS implementations. In methods that use adaptive characteristic maps, usually the difference between estimated power and the measured current battery power is calculated, then if a significant error is verified, the reference points in the characteristic maps are adjusted. Main advantage of this method is its relatively simple implementation. Disadvantages include accentuated loss of accuracy under dynamic loads, and a need for large storage device to implement this method, since the characteristic maps need to be stored in the BMS and can occupy a considerable amount of storage space [10, 82].

In [84], a method for estimating the available battery power is presented, where the battery SOC, temperature and aging state data is used for power prediction. Polynomial functions are used to map the relationship between battery power and battery SOC, temperature and aging state of the battery.

Model based methods

In model based methods for power prediction, the general procedure is to use an algorithm for online parameter estimation of a battery model, such as an adaptive filter algorithm or a least-square-based method, then use the parameters obtained for power prediction. Model based methods have been the preferred choice for SOP estimation, due to their reduced complexity and lesser storage needed when compared to characteristic maps methods. However, to remain accurate throughout the battery lifetime, online parameterization methods must be used to adjust battery parameters to different SOC, temperature and battery age [10, 82, 83].

In [85], SOP was estimated using a Kalman filter (KF), where the KF was used to estimate the parameters of an equivalent circuit model. Then, the estimated resistances of the equivalent circuit were used for SOP estimation. The method was tested under different current profiles to verify its functionality. In [86], an extended Kalman filter (EKF) method was used for parameter estimation

of a 1 RC pair equivalent circuit model, similar to the nRC model presented in Figure 2.26, then these parameters were used for SOP estimation, as depicted in Figure 2.44:

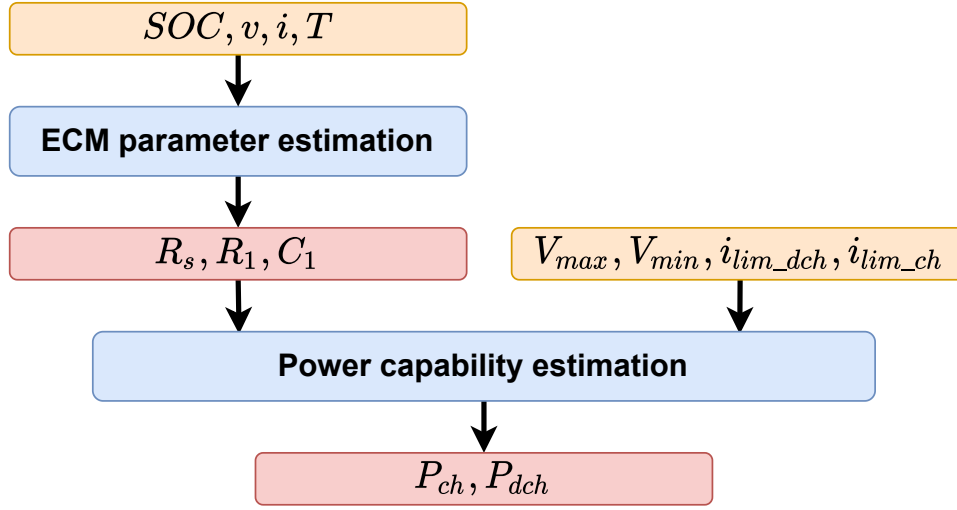


Figure 2.44: SOP estimation block schematic, adapted from [86].

Where v , i , t are voltage, current and temperature measurements, respectively; R_s is the series resistance of the model; R_1 , C_1 are the resistance and capacitance values of the model's RC pair; V_{max} , V_{min} are the battery's maximum and minimum voltage limits; i_{lim_dch} , i_{lim_ch} are the defined current limits for discharge and charge operations, respectively. Full presentation of the calculations used to estimate charge and discharge power capability are provided in the article.

2.5 Online cell parameterization strategies

In this section, only methods suitable for online parameter estimation are reviewed. As mentioned, online parameter estimation is essential to maintain accurate state estimation throughout the battery lifetime, since these parameters vary due to battery aging and operating conditions [11, 83].

Online parameter estimation techniques have been described for electrochemical battery models ([87, 88]). However, due to the high complexity of the model and complex calculations required, these methods are generally not suitable to be embedded in low cost microcontrollers in online BMS [83].

EIS techniques have been implemented for online impedance estimation ([89]). However, the need for additional circuitry to generate diagnosis signals for the battery makes this method more expensive, and therefore not as suitable for online implementation as others.

Some evolutionary techniques have been used for parameter estimation, such as the particle swarm optimisation ([22]) and genetic algorithms ([90]). These techniques can achieve better parameter estimation accuracy than other online methods, but due to excess computational effort and memory consumption, these methods are generally not suitable for practical implementation in embedded BMS [91].

The reviewed methods for parameter estimation were divided in two groups: Kalman filters and Least-squares techniques [11, 83, 91].

Kalman filters

KF operation and characteristics have been described in the previous Section 2.4.1.3. To combine battery state estimation with parameter estimation, two approaches are often used: Joint KF based methods ([92]), and Dual-KF based methods ([40]).

Joint methods use a single filter, where the state variables of the model considered include state estimation variables (such as SOC, OCV, and others) and also include the equivalent circuit model parameters used, leading to a higher dimension state model. Using a higher dimension state model leads to larger matrix operations, such as the matrix inversions required in KF, increasing computational effort of the method [83, 91].

Dual-KF methods use two filters, one for the state estimation variables, and the other for model parameter estimation. The two filters operate in parallel, where each filter uses the state variables of the other filter to update his own state variables, and vice-versa, as seen in Figure 2.32. The state-space model for an EKF state filter can be designed as [18]:

$$\begin{aligned} x_{k+1} &= f(x_k, u_k, \theta_k) + w_k \\ y_k &= h(x_k, u_k, \theta_k) + v_k \\ w_k &\sim N(0, Q_k^x) \\ v_k &\sim N(0, R_k^y) \end{aligned} \quad (2.22)$$

Where x_k is the model state vector, θ_k is the model parameters vector, u_k is the exterior input vector, y_k is the output vector, w_k is the process noise and v_k the measurement noise, both zero-mean processes of covariance Q_k^x and R_k^y , respectively. $f(x_k, u_k, \theta_k)$ is a non-linear function that relates state estimated at iteration k to the next state estimated at the next iteration $k+1$. $h(x_k, u_k, \theta_k)$ relates the updated states estimated to the verified measurements.

For the parameters filter, if assumed that the model parameters vary slowly through time, the parameters filter can be configured to adaptively estimate the model parameters, where the state-space model for the parameter estimation EKF is:

$$\begin{aligned} \theta_{k+1} &= \theta_k + r_k \\ d_k &= h(x_k, u_k, \theta_k) + e_k \\ r_k &\sim N(0, Q_k^\theta) \\ e_k &\sim N(0, R_k^\theta) \end{aligned} \quad (2.23)$$

Where the dynamic changes to θ_k are due to the simulated white noise r_k with covariance Q_k^θ to adapt the parameters over time. Output d_k is given as a result of the estimated θ_k , combined with white noise process e_k with covariance R_k^θ , representing sensor noise and modeling uncertainties [18].

Despite using two filters, the Dual-KF approach has less computational cost than the Joint-KF approach, since it uses lower order matrices on its calculations [83].

Least-squares techniques

Recursive least-squares techniques have been widely used for model parameter estimation ([93, 94, 95, 96] in many different forms with various levels of accuracy and complexity. In [93], a general RLS algorithm with an optimal forgetting factor for parameter estimation is presented, based on a nRC equivalent circuit model, as seen in Figure 2.26:

$$U_L(k) = (1 - \sum_{i=1}^n c_i)U_{oc}(k) + c_1U_L(k-1) + c_2U_L(k-2) + \dots + c_nU_L(k-n) + \quad (2.24)$$

$$+ c_{(n+1)}I_L(k) + c_{(n+2)}I_L(k-1) + c_{(n+3)}I_L(k-2) + \dots + c_{(2n+1)}I_L(k-n)$$

$$U_L(k) = \Phi_n^T(k) * \theta_n(k) \quad (2.25)$$

$$\Phi_n(k) = [1 \quad U_L(k-1) \quad \dots \quad U_L(k-n) \quad I_L(k) \quad I_L(k-1) \quad \dots \quad I_L(k-n)]^T \quad (2.26)$$

$$\theta_n(k) = [(1 - \sum_{i=1}^n c_i)U_{oc}(k) \quad c_1 \quad c_2 \quad \dots \quad c_{2n+1}]^T \quad (2.27)$$

$$y_k = \Phi_n^T(k) * \tilde{\theta}_n(k-1) \quad (2.28)$$

$$e_k = U_L(k) - \Phi_n^T(k) * \tilde{\theta}_n(k-1) \quad (2.29)$$

$$K(k) = \frac{P(k-1)\Phi_n(k)}{\lambda + \Phi_n^T(k)P(k-1)\Phi_n(k)} \quad (2.30)$$

$$P(k) = \frac{P(k-1) - K(k)\Phi_n^T(k)P(k-1)}{\lambda} \quad (2.31)$$

$$\tilde{\theta}_n(k) = \tilde{\theta}_n(k-1) + K(k)e(k) \quad (2.32)$$

Where $U_L(k)$ is the voltage at the terminals of a battery model with n RC pairs, represented in a discrete form after using a bilinear transformation method described in the article; $U_{oc}(k)$ is the OCV of the cell; $I_L(k)$ is the current flowing through the battery; $\Phi_n(k)$ is the sampled input vector for a n RC pair model; $\theta_n(k)$ is the parameters vector for a n RC pair model; y_k is the voltage output of the algorithm using current input vector and the estimated parameters of the previous iteration; e_k is the error between the measured terminal voltage and the voltage output of the algorithm; $K(k)$ is the gain of the algorithm; $P(k)$ is the covariance matrix; λ represents the forgetting factor, which is vital to obtain small error in parameter estimation, usually with a value between [0.95, 1]. The diagram for the algorithm is presented in Figure 2.45. From these estimated parameters, the model parameters can then be directly calculated [93].

The advantage of this method over the Kalman filter methods comes in its significantly lower computational demand and lower memory usage. Disadvantage of this method is that it may face convergence problems if the battery model doesn't replicate battery behaviour accurately, making it a less robust technique than the Kalman filter methods [83, 91].

Non-recursive least-squares techniques have also been applied in battery parameter estimation ([97, 98]). The idea behind non-recursive estimation is to collect and store measurements of battery voltage and current over a period of time, then execute an iterative least-squares optimization method to fit the equivalent battery model to the collected data, determining the model parameters. These methods can be used in cases where the battery model is highly non-linear or fractional. These methods are also capable of having higher accuracy and stability than recursive methods. However, due to high computational effort and memory storage required, these techniques are not very suitable for online implementation in embedded BMS [83, 91].

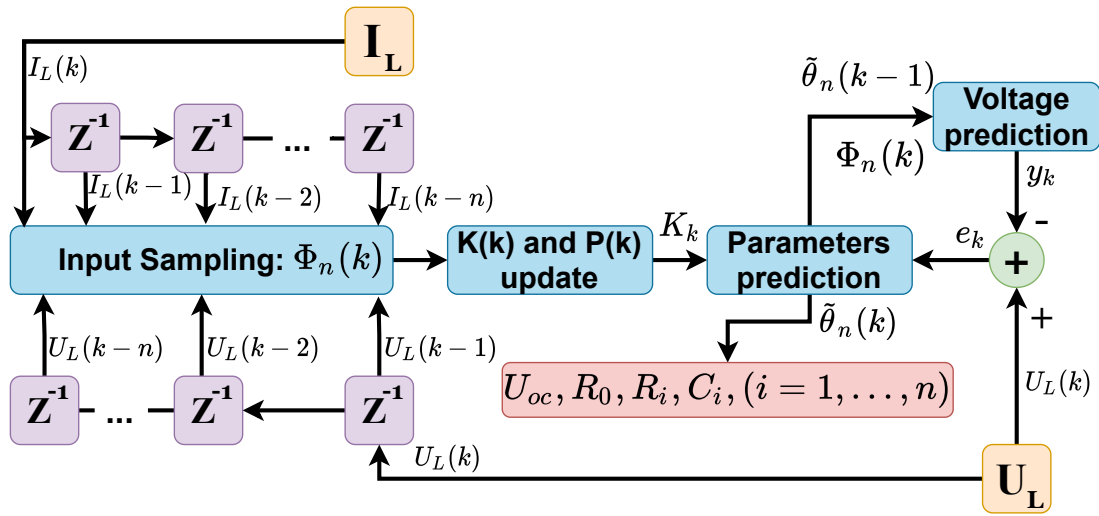


Figure 2.45: RLS diagram, adapted from [93].

Chapter 3

Algorithms and Implementation

3.1 Feedforward Neural Networks

As mentioned in the previous Subsection 2.4.1.4, FNNs have the capacity to learn and model the dynamics of any system, making it a very powerful tool in non-linear system modeling.

3.1.1 Forward propagation

In FNNs, the process of taking a set of inputs and generating an output is commonly referred as the forward propagation. As the name indicates, information in the network is passed sequentially from the input layer neurons to the output neurons, where each neuron in a previous layer is connected to neurons in the next layer through a set of weights, which define the influence that the output of a neuron in the previous layer has over the output of a neuron in the next layer. A general view of a neuron is presented in 3.1:

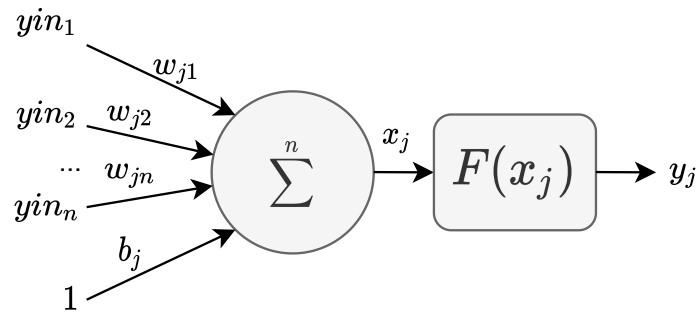


Figure 3.1: Neuron model.

Where $yin_1, yin_2, \dots, yin_n$ represent the inputs to the neuron, in a FNN these are either the inputs to the network or the outputs of the neurons in the previous layer; $w_{j1}, w_{j2}, \dots, w_{jn}$ represent the set of weights associated to the inputs, representing the connections between neuron j to the previous layer neurons; b_j represents the bias of the neuron j ; and $F(x_j)$ the activation function of the neuron [99]. The outputs x_j and y_j of each neuron are then given by:

$$x_j = \sum_i (w_{ji} y_{in_i}) + b_j \quad (3.1)$$

$$y_j = F(x_j) \quad (3.2)$$

As seen, the algorithm only requires adding and multiplying operations to generate an output, making the network simple and cost effective in terms of calculations for online implementation in embedded applications. However, depending on the size of the network, many parameters (weights, biases) may be required in memory space to accurately model complex systems, limiting its application in some areas.

A general and simple method to calculate the forward propagation from all neurons, from a layer to the next, is to perform the calculations in matrix form. Considering two layers with n and m number of neurons, respectively, the vector Y containing all outputs of the second layer can be given as:

$$X = W \times Y_{in} + B \quad (3.3)$$

$$Y = F(X) \quad (3.4)$$

Where X is a $m \times 1$ vector, containing all outputs x_j ; W is a $m \times n$ matrix, containing all weights connecting both layers; Y_{in} is a $n \times 1$ vector, containing all outputs of the previous layer; B is a $m \times 1$ vector, containing the bias value for each neuron; $F(X)$ represents the application of the activation function to each element in X ; And Y is a $m \times 1$ vector containing all outputs of the layer [99].

3.1.2 Gradient descent - Supervised Network Learning

The most popular algorithms for neural network learning are the gradient descent algorithms. Gradient descent is essentially an iterative optimization algorithm, where the purpose is to minimize a function by altering a variable that directly influences it. The general procedure in this algorithm, is to minimize a function $y = f(x)$, by adjusting x a small amount ϵ in the opposite direction of its derivative $f'(x)$. For an ϵ small enough, $f(x - \epsilon f'(x)) \leq f(x)$. If $f'(x) = 0$, then a local minimum has been reached, and for all points in the vicinity of x , $f(x) < f(x \pm \epsilon)$ [100].

In the context of neural network learning, the purpose is to minimize a cost function, also commonly referred as "loss function", based on the error of the network's prediction against the desired value, by adjusting the value of the weights and bias that connect the neurons and form the network. In this sense, to apply the gradient descent for network learning, the derivatives of the cost function in order to the weights and bias need to be calculated, in order to adjust these weights and bias of the network for minimization of the cost function. This vector of partial derivatives is called the gradient of the cost function. Minimizing the cost function, means minimizing the

error of the network's prediction over the set of training samples, where each training sample is composed of a set of inputs, and the desired output of the network for these inputs [100]. A common function used for the loss function E is the instantaneous value of the sum of quadratic errors:

$$E = \frac{1}{2} \sum_{j=1}^{n_j} (\bar{y}_j - y_j)^2 \quad (3.5)$$

Where $\bar{y}_j$ is the desired network output for a set of inputs, y_j is the generated network output for a set of inputs, and n_j is the number of network outputs.

The weights and bias of the network are then adjusted as:

$$\Delta w = -\frac{\alpha}{m} \sum_n^m \frac{\partial E_n}{\partial w} \quad (3.6)$$

$$\Delta b = -\frac{\alpha}{m} \sum_n^m \frac{\partial E_n}{\partial b} \quad (3.7)$$

$$w^+ = w + \Delta w \quad (3.8)$$

$$b^+ = b + \Delta b \quad (3.9)$$

Where α is the defined learning rate, m is the total number of training samples, $\frac{\partial E_n}{\partial w}$ is the gradient of the cost function in order to the weight w at training sample n , $\frac{\partial E_n}{\partial b}$ is the gradient of the cost function in order to the bias b at training sample n , w^+ and b^+ are the new adjusted weights and bias. The learning rate α is a very important parameter for the algorithm, usually chosen by trial and error, to obtain the best results in training [100].

To evaluate the performance of the models over the training or validation dataset, three functions based on the verified error are commonly used: Mean Squared Error (MSE), Root Mean Squared Error (RMSE) and Median Absolute Error (MAE):

$$MSE = \frac{1}{m} \sum_{j=1}^m (\bar{y}_j - y_j)^2 \quad (3.10)$$

$$RMSE = \sqrt{MSE} \quad (3.11)$$

$$MAE = \frac{1}{m} \sum_{j=1}^m |\bar{y}_j - y_j| \quad (3.12)$$

Chain rule - gradient calculation

The gradients of the cost function, in order to the weights and bias can be easily calculated by applying the chain rule of calculus. Considering a neuron j in the output layer and a neuron i in the hidden layer, the gradients of the weights that connect the hidden layer to the output layer can be calculated by [54, 100]:

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial y_j} \times \frac{\partial y_j}{\partial x_j} \times \frac{\partial x_j}{\partial w_{ji}} \quad (3.13)$$

$\frac{\partial E}{\partial y_j}$, $\frac{\partial y_j}{\partial x_j}$, $\frac{\partial x_j}{\partial w_{ji}}$ can then be replaced by:

$$\frac{\partial E}{\partial y_j} = -(\bar{y}_j - y_j), \text{ with } E = \frac{1}{2} \sum_{j=1}^{n_j} (\bar{y}_j - y_j)^2 \quad (3.14)$$

$$\frac{\partial y_j}{\partial x_j} = F'(x_j), \text{ with } y_j = F(x_j) \quad (3.15)$$

$$\frac{\partial x_j}{\partial w_{ji}} = y_{in_i}, \text{ with } x_j = \sum_i (w_{ji} * y_{in_i}) + b_j \quad (3.16)$$

Replacing these values in the original equation 3.13 we get:

$$\frac{\partial E}{\partial w_{ji}} = -(\bar{y}_j - y_j) F'(x_j) y_{in_i} \quad (3.17)$$

Considering a neuron i in a hidden layer that is directly connected to the output layer, and a neuron k connected to the hidden layer i , the gradients of the weights that connect the input or hidden layer k to the hidden layer i can be calculated by [54, 100]:

$$\frac{\partial E}{\partial w_{ik}} = \frac{\partial E}{\partial y_i} \times \frac{\partial y_i}{\partial x_i} \times \frac{\partial x_i}{\partial w_{ik}} \quad (3.18)$$

In this case where the neuron i is a neuron in the hidden layer, its output y_i will directly influence the output of all output layer neurons j , therefore the term $\frac{\partial E}{\partial y_i}$ is calculated as:

$$\frac{\partial E}{\partial y_i} = \sum_j^{n_j} \frac{\partial E}{\partial y_j} \times \frac{\partial y_j}{\partial x_j} \times \frac{\partial x_j}{\partial y_i} \quad (3.19)$$

Where n_j is the number of neurons in the output layer, and $\frac{\partial E}{\partial y_j}$ and $\frac{\partial y_j}{\partial x_j}$ are given by equations 3.14 and 3.15, respectively. The remaining term $\frac{\partial x_j}{\partial y_i}$ is given by:

$$\frac{\partial x_j}{\partial y_i} = w_{ji}, \text{ with } x_j = \sum_i (w_{ji} y_{in_i}) + b_j \quad (3.20)$$

In this case, $y_{in_i} = y_i$. The remaining terms $\frac{\partial y_i}{\partial x_i}$ and $\frac{\partial x_i}{\partial w_{ik}}$ are given by equations 3.15 and 3.16, respectively [54, 100]. Replacing these terms in the original equation 3.18, the resulting value for $\frac{\partial E}{\partial w_{ik}}$ becomes:

$$\frac{\partial E}{\partial w_{ik}} = \sum_j^{n_j} (-(\bar{y}_j - y_j) F'(x_j) w_{ji}) \times F'(x_i) y_{in_k} \quad (3.21)$$

The process for determining the gradient for the bias values is very similar to the one presented for the weights values, except now the term $\frac{\partial x_j}{\partial w_{ji}}$ in 3.13 is replaced by $\frac{\partial x_j}{\partial b_j}$:

$$\frac{\partial x_j}{\partial b_j} = 1, \text{ with } x_j = \sum_i (w_{ji} * y_{in_i}) + b_j \quad (3.22)$$

Network learning algorithms

The algorithm described so far is named the "batch gradient descent" algorithm, where the update to the parameters is done after going through all training samples n_t and averaging the gradient values for each iteration, as seen in equations 3.6 - 3.9. The following is a general description of this algorithm [99]:

1. **Parameter initialization:** Initialize parameters of the network θ (weights and bias) with low, random values. Initialize learning rate α .
2. **Training loop:** While *stopping condition* not reached:

Present all training samples to the network - n_t total number.

Calculate gradient parameters: $g_\theta = \frac{1}{n_t} \sum_n^{n_t} \frac{\partial E_n}{\partial \theta}$

Update parameters: $\theta^+ = \theta - \alpha g_\theta$

The disadvantage of this algorithm, is that generally very large datasets are needed to obtain good generalization. Updating parameters after running through all training samples can take too long, and make the process of network learning very slow in general. As a solution for these problems, network learning is generally done using the Stochastic Gradient Descent (SGD) algorithms and its variants [100].

Stochastic Gradient Descent (SGD)

The main difference between SGD and regular gradient descent algorithms, is that the gradient is averaged over a smaller set of training samples n_b . The algorithm essentially makes a "prediction" of the total gradient, using a smaller set of training samples. The parameters are therefore updated at a much higher frequency compared to the default gradient descent algorithm, using an approximation of the gradient calculated over smaller set of samples. This alteration makes the method capable of converging much faster over training datasets with a very high number of training samples. However, when randomly sampling a small amount n_b training samples, the gradient

calculated over this small set may not be an accurate representation of the gradient over the whole training samples, therefore training becomes much more unstable compared to default gradient descent [99]. The following is a general description of this algorithm:

1. **Parameter initialization:** Initialize parameters of the network θ (weights and bias) with low, random values. Initialize learning rate α .
2. **Training loop:** While *stopping condition* not reached:

Randomly sample a mini-batch of n_b training samples and present it to the network.

Calculate gradient parameters for the mini-batch: $g_\theta = \frac{1}{n_b} \sum_n \frac{\partial E_n}{\partial \theta}$

Update parameters: $\theta^+ = \theta - \alpha g_\theta$

In the SGD algorithms and its variants, it is customary to decrease the learning rate throughout the training iterations, since better results are achieved this way. A simple way to achieve this is to decrease the learning rate proportionally to the iteration step k , until n iterations are achieved:

$$\alpha_k = \left(1 - \frac{k}{n}\right)\alpha_0 + \frac{k}{n}\alpha_n \quad (3.23)$$

Where α_k is learning rate at iteration k , α_0 is the initial learning rate, and α_n is the final value of the learning rate. After n iterations are achieved, the learning rate $\alpha = \alpha_n$, until the end of the training procedure [100].

Stochastic Gradient Descent with Momentum (SGDM)

As a method for accelerating training procedure, SGDM introduces a momentum component η over the original SGD algorithm. This momentum component is used to help maintain the gradient direction over the training samples, stabilizing the training procedure and acting as a filter against the noise introduced over estimating the gradient using a small amount of training samples n_b [99]. The following is a general description of this algorithm:

1. **Parameter initialization:** Initialize parameters of the network θ (weights and bias) with low, random values. Initialize learning rate α . Initialize momentum value η , typical values include $\eta = 0.5$, $\eta = 0.9$ or $\eta = 0.99$.
2. **Training loop:** While *stopping condition* not reached:

Randomly sample a mini-batch of n_b training samples and present it to the network.

Calculate gradient parameters for the mini-batch: $g_\theta = \frac{1}{n_b} \sum_n \frac{\partial E_n}{\partial \theta}$

Update parameters: $\theta_k^+ = \theta_k - \alpha g_\theta + \eta \Delta \theta_{k-1}$

As seen, the parameters update rule is changed. Now, at each update, a term $\eta \Delta \theta_{k-1}$ is added, where $\Delta \theta_{k-1}$ is the value of the change of the parameters in the previous iteration $k - 1$ [99].

Root Mean Squared Propagation (RMSProp)

The learning rate α is consistently the most difficult parameter to set for neural network learning, due to its impact on model performance and no clear definition or method on how to set it other than perhaps trial and error, by visualizing which value yields the fastest and best training results. As a solution, adaptive learning rates algorithms were proposed for network learning. These algorithms adjust the resulting learning rate for each parameter individually, and can lead to much faster network training compared to SGD and SGDM [100].

One of the most popular algorithms in this category is the RMSProp algorithm. In this method, the initial learning rate is divided by the square root of an exponentially decaying moving average of the squared gradient of the parameters v_θ , with a decay value β , resulting in an adaptive learning rate for each parameter. The following is a general description of this algorithm:

1. **Parameter initialization:** Initialize parameters of the network θ (weights and bias) with low, random values. Initialize learning rate α . Initialize decay factor β , typical values include $\beta = 0.9$, $\beta = 0.99$ or $\beta = 0.999$. Initialize ε with a very small value to help stabilize division by very small numbers, typical value $\varepsilon = 10^{-6}$. Initialize moving average accumulator for the squared gradient values $v_\theta = 0$.
2. **Training loop:** While *stopping condition* not reached:

Randomly sample a mini-batch of n_b training samples and present it to the network.

Calculate gradient parameters for the mini-batch: $g_\theta = \frac{1}{n_b} \sum_n \frac{\partial E_n}{\partial \theta}$

Accumulate moving average squared gradient of parameters: $v_\theta^+ = \beta v_\theta + (1 - \beta) g_\theta \cdot g_\theta$

Update parameters: $\theta^+ = \theta - \frac{\alpha}{\sqrt{v_\theta^+ + \varepsilon}} \cdot g_\theta$

Where $\frac{\alpha}{\sqrt{v_\theta^+ + \varepsilon}}$ is the adjusted learning rate for each parameter, applied element wise [100].

Adaptive Moment Estimation (ADAM)

The ADAM algorithm extends the RMSProp algorithm, with the addition of a term for the exponentially decaying moving average of the gradient of the parameters m_θ , and a method for bias correction of the m_θ and v_θ , to compensate for their initial value $m_\theta = 0$ and $v_\theta = 0$ [101]. The following is a general description for this algorithm:

1. **Parameter initialization:** Initialize parameters of the network θ (weights and bias) with low, random values. Initialize learning rate α , typical value $\alpha = 0.001$. Initialize decay factors β_1 and β_2 , typical values include $\beta_1 = 0.9$ and $\beta_2 = 0.999$. Initialize ε with a very small

value to help stabilize division by very small numbers, typical value $\varepsilon = 10^{-8}$. Initialize moving average accumulator for the gradient values and squared gradient values, $m_\theta = 0$, $v_\theta = 0$. Initialize time step $k = 1$.

2. Training loop: While *stopping condition* not reached:

Randomly sample a mini-batch of n_b training samples and present it to the network.

Calculate gradient parameters for the mini-batch: $g_\theta = \frac{1}{n_b} \sum_n \frac{\partial E_n}{\partial \theta}$

Accumulate moving average gradient of parameters: $m_\theta^+ = \beta_1 m_\theta + (1 - \beta_1) g_\theta$

Accumulate moving average squared gradient of parameters: $v_\theta^+ = \beta_2 v_\theta + (1 - \beta_2) g_\theta \cdot g_\theta$

Compensate bias for moving average gradient: $\hat{m}_\theta = \frac{m_\theta}{1 - \beta_1^k}$

Compensate bias for moving average squared gradient: $\hat{v}_\theta = \frac{v_\theta}{1 - \beta_2^k}$

Update parameters: $\theta^+ = \theta - \frac{\alpha \hat{m}_\theta}{\sqrt{\hat{v}_\theta} + \varepsilon} \cdot g_\theta$

Update time step variable: $k^+ = k + 1$

The adaptive learning rate term $\frac{\alpha \hat{m}_\theta}{\sqrt{\hat{v}_\theta} + \varepsilon}$ now uses the corrected values of m_θ and v_θ to update the parameters of the network [101].

Stopping condition

A typical problem when training a network is deciding when to stop the training procedure. The problem, is that when training a large model with enough capacity to overfit the desired task, eventually the model will continue decreasing the training error, while increasing the validation error, steadily losing its generalization capacity and resulting in a worse performing model at each further iteration of the training procedure [99].

Common solutions for stopping the training include defining a maximum number of epochs for the training procedure, and/or defining an accuracy value threshold for the validation dataset, based on the performance functions presented in 3.10, 3.11 or 3.12.

An interesting method to follow when defining the stopping condition is the early stopping algorithm, presented in [100]. The idea in this algorithm, is to stop the training procedure if for a defined number of iterations n_p , the absolute validation error does not decrease. The following is a general description of the algorithm:

1. **Define:** Number of iterations between validations n_v . Number of steps worse validation results are observed n_p . Initialize parameters θ . Initialize auxiliary values:

$$i = 0$$

$$j = 0$$

$$v = \infty$$

$$\theta^* = \theta$$

$$i^* = i$$

2. **Validation loop:** While $j < n_p$:

Run training procedure for n_v iterations and update θ

$$i = i + n_v$$

$$v' = \text{ValidationError}(\theta)$$

if $v' < v$:

$$j = 0$$

$$\theta^* = \theta$$

$$i^* = i$$

$$v = v'$$

else:

$$j = j + 1$$

The resulting best parameters are stored in θ^* and the best number of training iterations is stored in i^* . This is a popular method of defining stopping condition for the training procedure, due to its effectiveness combined with its simplicity. Still, the best values for n_v and n_p remain to be found for the particular task [100].

3.1.3 Activation functions

For feedforward neural networks, the choice of activation functions for the hidden layers is still a very active area of investigation, where no clear guidelines are yet defined for which function to use, depending on the designated task [100].

As a general choice, the Rectified Linear Unit (ReLU) function is a very good option, due to its simplicity and good performance at various different tasks, making it the most popular choice since its introduction. The ReLU function and its derivative are defined as:

$$ReLU(x) = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases} \quad (3.24)$$

$$ReLU'(x) = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases} \quad (3.25)$$

The problem when using ReLU activation function, is that the calculated derivative will be zero for training examples where its activation output is zero, meaning that the network won't learn for these examples. As a solution, other versions of ReLUs have also been used, such as the leaky ReLU unit. In this variation, the output for when $x < 0$ is now $f(x < 0) = s \times x$, meaning that $f'(x < 0) = s \neq 0$. The leaky ReLU function is defined as:

$$leaky - ReLU(x) = \begin{cases} s \times x & x < 0 \\ x & x \geq 0 \end{cases} \quad (3.26)$$

$$leaky - ReLU'(x) = \begin{cases} s & x < 0 \\ 1 & x \geq 0 \end{cases} \quad (3.27)$$

Where s is usually a small value, typically $s = 0.01$ [100].

Before the introduction of the ReLU and its variants, the most popular choices for activation units in feedforward neural networks were the logistic sigmoid and the hyperbolic tangent functions [99, 100]. The logistic sigmoid function and its derivative are defined as:

$$sig(x) = \frac{1}{1 + e^{-x}} \quad (3.28)$$

$$sig'(x) = sig(x)(1 - sig(x)) \quad (3.29)$$

The hyperbolic tangent function is commonly implemented as:

$$tanh(x) = \frac{2}{1 + e^{-2x}} - 1 \quad (3.30)$$

$$tanh'(x) = 1 - tanh(x)^2 \quad (3.31)$$

3.2 State-of-Charge estimation

The traditional method for estimating SOC in a battery cell has been the *Coulomb Counting* method, which consists in integrating the discharging or charging current on the battery pack,

in order to calculate the electric charge that enters or leaves the battery, as described in Section 2.4.1.1. Due to the possible accumulated error caused by integrating the current over time [26], other alternatives for SOC estimation were presented, such as the more common model-based methods [40, 43, 102], described in Sections 2.4.1.2 and 2.4.1.3, and machine learning approaches, described in Section 2.4.1.4.

Multiple machine learning algorithms have been proposed for SOC estimation in these past years. These methods can provide an advantage over model-based methods, since a deeper knowledge of the battery's internal behaviour is not required, reducing the amount of prior study and testing of the cell's behaviour.

With the arrival of the machine learning embedded systems era, power and energy efficiency requirements are gradually no longer an obstacle for automotive applications, but the determination of optimal network architecture is still a bottleneck.

All of the data-driven methods reviewed in Section 2.4.1.4 are capable of accurately estimating SOC. However, in many of the cases reviewed the models developed could be considered too complex or memory consuming to be realistically implemented in low-cost BMS for EVs. These limitations may prevent its widespread use in current BMS solutions. Therefore, in this section, the focus was in presenting a simple feedforward neural network model that could accurately estimate SOC, while being suitable for implementation in typical BMS present in EVs.

3.2.1 Proposed Model

A general view of the proposed model is presented in Fig. 3.2. All details and components of the tested models are explained in this section.

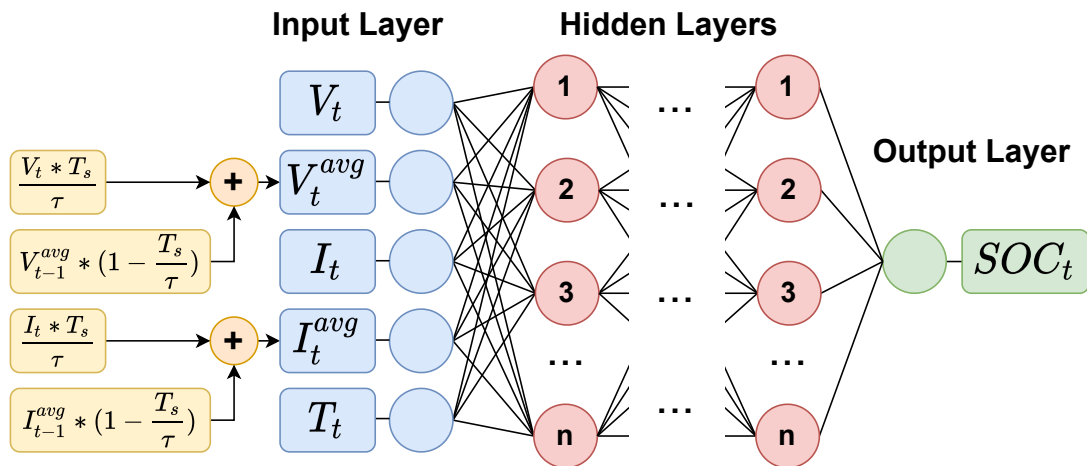


Figure 3.2: FNN general model used

Model structure

Inputs/Outputs: In the proposed model, solely five inputs are used for SOC estimation:

- Terminal voltage at the cell - V_t
- Current flowing through the cell - I_t
- Temperature of the cell - T_t
- Exponential moving average of value τ of the voltage measurements - V_t^{avg} :

$$V_t^{avg} = V_{t-1}^{avg} * (1 - \frac{T_s}{\tau}) + \frac{V_t * T_s}{\tau} \quad (3.32)$$

- Exponential moving average of value τ of the current measurements - I_t^{avg} :

$$I_t^{avg} = I_{t-1}^{avg} * (1 - \frac{T_s}{\tau}) + \frac{I_t * T_s}{\tau} \quad (3.33)$$

Where T_s represents the sampling interval, in seconds. Including the T_s term allows the same model to remain accurate for battery data sampled at a different rate than the data used for training the model. After multiple experiments, it was set $\tau = 250$ seconds, as this was the value that provided the best results.

Providing a moving average of past voltage and current measurements allows the model to relate past values of voltage and current to estimate SOC, adding a memory component to the FNN model, and greatly reducing SOC estimation error when compared to only using voltage, current and temperature measurements.

In [53, 103], similar inputs are used compared to the proposed model. However, the method used for calculating the moving average of past voltage and current measurements required that those previous values be stored in memory for the moving average calculation. The proposed method in this paper of calculating the moving average of previous measurements requires only one value in storage for each moving average at all times, while still providing very accurate results.

Number neurons and layers: To determine an optimal model that provided accurate results while having the least number of parameters, it was decided to train models with only one or two hidden layers, with varying number of neurons per hidden layer. For the one layer models, 7 models were trained with the number of neurons set to 10, 20, 40, 60, 80, 100, and 120. For the two layer models, 5 models were trained with the number of neurons per layer set to 5, 10, 15, 20, and 25. The number of required parameters and achieved results for each model are presented in the next chapter, on Table 4.1.

Activation functions: For the one hidden layer models, the activation function in the hidden layer was the computationally efficient ReLU, described in Section 3.1.3. For the two hidden layer models, the activation function in the first layer was the hyperbolic tangent sigmoid transfer function, and the activation function in the second layer was the ReLU, as in the one layer model.

Two different activation functions were implemented for the hidden layers of the two hidden layer models, due to being the setup that yielded the best estimation results after multiple experiments.

A linear activation function was implemented in the output layer of all models, where the output of the output layer neuron is the same as its input.

3.3 State-of-Health estimation

There are essentially two ways of estimating SOH in a battery cell: estimating the internal resistance of the cell; or estimating the actual total capacity of the cell and comparing it to that of an original cell.

In this sense, multiple methods have been used for SOH estimation, as reviewed in Section 2.4.2. In the machine learning methods reviewed, a common theme was extracting features from the battery's charging profiles, to estimate the SOH of a cell.

During the battery's operation in EVs, there are three distinct phases of operation: discharging, rest, and charging. During the discharge phase, very dynamic current profiles may be applied to the battery, which may complicate the measurement and estimation of the battery's internal characteristics. During the rest stage, no current is applied to the battery, therefore little to no changes can be verified by external measurements. During the charging phase however, at first, constant current is applied to the battery for charging, while in a second phase, current value is decreased through time to maintain a constant voltage level at its terminals, resulting in a much more controllable and "peaceful" environment for parameter and feature extraction.

As illustrated in Figure 3.3, as the battery's cell ages, its voltage waveform during charging changes as well. As can be seen in the graph, the voltage waveform during the charge process, after being discharged 1 time (cycle 1), is clearly different than after being discharged 80 times (cycle 80), or after being discharged 160 times (cycle 160). The differences can be seen in the slope of the voltage curves and in the average voltage values verified.

Since SOH is a cell parameter that varies slowly through the battery's lifetime, estimating the cell's SOH through the analysis of its voltage waveform during charging process is feasible. However, in the methods reviewed, the feature extraction and input preprocessing phase could be complicated to implement for online operation. Therefore, the focus in this section was to investigate and present a simple model based on a FNN for SOH estimation, which uses a very simple method for online input preprocessing, based on the voltage curves during charging operation.

3.3.1 Proposed Model

The proposed model is very similar to the one used for the SOC estimation, depicted in Figure 3.2, with added inputs.

Model structure

Inputs/Outputs: In the proposed model, 7 inputs are used for SOH:

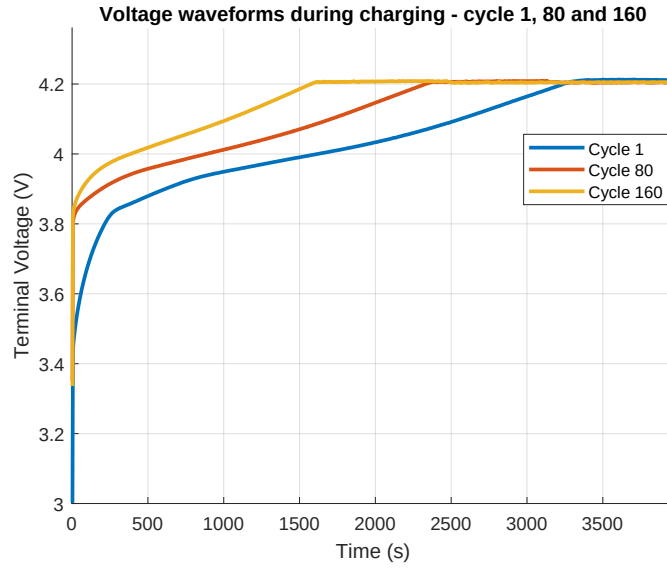


Figure 3.3: Voltage waveforms during charging procedure.

- Terminal voltage at the cell - V_t
- Current flowing through the cell - I_t
- Temperature of the cell - T_t
- Exponential moving average of value τ of the voltage measurements - V_t^{avg} :

$$V_t^{avg} = V_{t-1}^{avg} * \left(1 - \frac{T_s}{\tau}\right) + \frac{V_t * T_s}{\tau} \quad (3.34)$$

- Exponential moving average of value τ of the current measurements - I_t^{avg} :

$$I_t^{avg} = I_{t-1}^{avg} * \left(1 - \frac{T_s}{\tau}\right) + \frac{I_t * T_s}{\tau} \quad (3.35)$$

- Terminal voltage derivative - $V_t' = V_t - V_{t-1}$:
- Exponential moving average of value τ of the terminal voltage derivative - $V_t'^{(avg)}$:

$$V_t'^{(avg)} = V_{t-1}'^{(avg)} * \left(1 - \frac{T_s}{\tau}\right) + \frac{V_t' * T_s}{\tau} \quad (3.36)$$

Where T_s represents the sampling interval, in seconds. After multiple experiments, it was set $\tau = 750$ seconds, as this was the value that provided the best results.

The moving average values in this case are accumulated only from the beginning of the charging profile, adding a memory component of the measured voltage values during the constant current phase of the charging procedure. Therefore, when the constant current phase of the charging

procedure is near its end, these values will be able to reflect necessary characteristics of the voltage waveform to determine the current SOH value of the cell.

Ideally, after presenting the charging profile to the battery, only one SOH estimation is made. Therefore, it is required to define a voltage setpoint, for which when the measured voltage value is higher than this defined voltage setpoint, the inputs are fed to the FNN model, and a SOH estimation is made and stored in memory.

In this case, the SOH output of the model is given as a direct percentage value, relating the actual total capacity value of the cell to the total capacity of a brand new cell, as described in the previous chapter in equation 2.2.

Number neurons and layers: To determine an optimal model that provided accurate results while having the least number of parameters, it was decided to train models with only one hidden layer, with varying number of neurons per hidden layer. In this sense, 6 models were trained with the number of neurons in the hidden layer set to 10, 20, 30, 40, 50 and 60. The number of required parameters and achieved results for each model are presented in the next chapter on Tables 4.3, 4.4, 4.5 and 4.6.

Activation functions: For the hidden layer of all models trained in this section, the ReLU activation function was used, as described in Section 3.1.3.

A linear activation function was implemented in the output layer of all models, where the output of the output layer neuron is the same as its input.

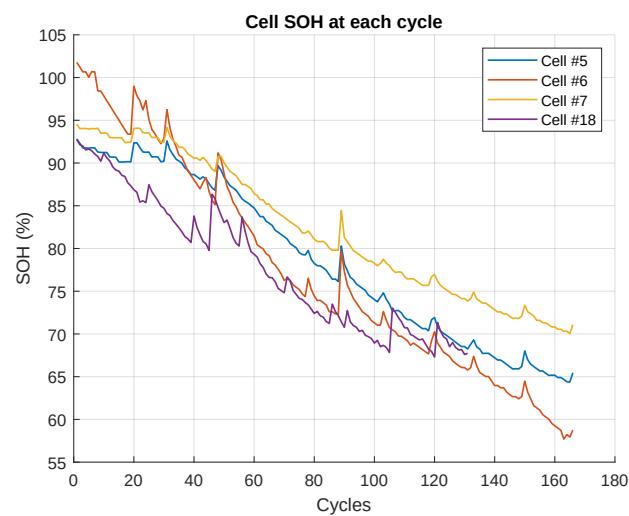


Figure 3.4: Battery SOH measured at each cycle.

Training and validation dataset

The training and validation dataset was taken from the NASA Prognostics Center of Excellence Data Repository [104]. The repository contains data associated to the repeated charging and discharging of four 18650 lithium-ion cells, for cell degradation study purposes. The four cells, labeled #5, #6, #7 and #18, are all charged using the CC-CV method, and are discharged continuously until a minimum voltage setpoint is reached. This cycle is repeated 166 times, where significant battery ageing is verified for the last cycles. Figure 3.4 depicts the SOH observed at each cycle for each cell tested, while Table 3.1 shows the details of the charging and discharging cycles applied to the cells.

Cell	Charge current (A)	Charge cut-off current (mA)	Discharge current (A)	Upper voltage limit (V)	Lower voltage limit (V)	Temperature	Initial capacity (Ah)	Rated capacity (Ah)
#5	1.5	20	2	4.2	2.7	Room temperature	1.86	2
#6	1.5	20	2	4.2	2.5	Room temperature	2.04	2
#7	1.5	20	2	4.2	2.2	Room temperature	1.89	2
#18	1.5	20	2	4.2	2.5	Room temperature	1.86	2

Table 3.1: NASA Battery dataset description.

To train and validate the model, since data from 4 distinct cells are available, for training each model one of the cells was chosen for validation, while remaining cells were used for training the model. Therefore, four different training and validation data were used for training the models: cell #5 is chosen for validation, data in training uses cell #6, #7 and #18; cell #6 is chosen for validation, data in training uses cell #5, #7 and #18; cell #7 is chosen for validation, data in training uses cell #5, #6 and #18; and finally cell #18 is chosen for validation, data in training uses cell #5, #6 and #7. Results achieved with each of training and validation datasets are provided in the next chapter in Section 4.2.

Training methods

Training the models was performed in MATLAB environment, using custom developed training functions. In this case, all networks were trained using the SGDM algorithm, described in Section 3.1.2.

To ensure fairness in model comparison, each model was trained 5 times using the SGDM algorithm. Each training repetition lasted for 1000 epochs. Cost function used for training the networks was the instantaneous value of the sum of the quadratic errors, as depicted in equation 3.5.

3.4 Cell model parameter estimation

As mentioned in this paper, an online method for estimating the battery's parameters is very important, due to the variation of the cell's characteristics depending on the current SOC, temperature and current flowing through the battery. A common way to describe these battery's characteristics, is to model the cell using an equivalent circuit model, as described in Section 2.3. This way, each element in the equivalent circuit corresponds to a battery parameter to be estimated.

Many methods have been used for cell parameter estimation, as reviewed in Section 2.5. The most common methods reviewed use a Kalman Filter system to estimate the parameters of a nRC pair circuit, as depicted in Figure 2.26, due to the model providing a good combination of accuracy and simplicity for implementation.

Another popular alternative for model parameters estimation is the usage of a Recursive Least Squares with Forgetting Factor (RLS-FF) algorithm. This method provides an advantage over more common Kalman Filter algorithms in lower computational cost and not requiring any specific knowledge or study of the cell's internal characteristics, making it a completely data-driven algorithm.

Therefore, in this paper a study on the application of a RLS-FF algorithm to a 1RC pair cell model is presented, to showcase the method's accuracy when emulating the cell's voltage behaviour.

3.4.1 Proposed Algorithm

The proposed algorithm is taken from [94], where the parameters of a 1RC pair cell model, depicted in Figure 3.5, were estimated using a RLS-FF algorithm.

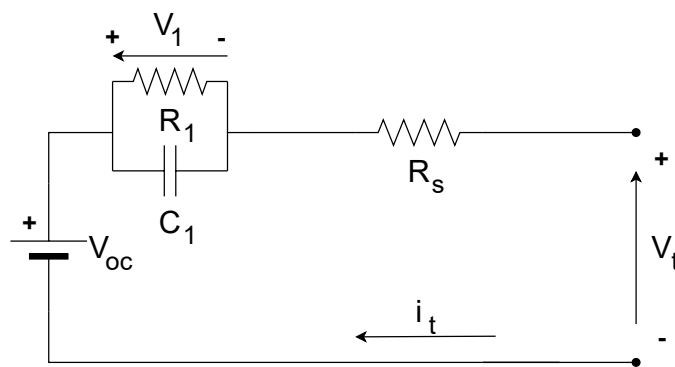


Figure 3.5: 1RC pair model used.

The RLS-FF algorithm has been previously explained in equations 2.28 - 2.32. To apply the RLS-FF algorithm for cell parameter estimation, it is necessary define the values for the measurement vector ϕ_n and the parameter vector θ_n .

In this sense, the voltage state equation of the cell model can be described in its discrete form:

$$V_1(k+1) = e^{-\frac{T_s}{\tau}} V_1(k) + (1 - e^{-\frac{T_s}{\tau}}) R_1 i_t(k) \quad (3.37)$$

$$V_t(k) = V_{oc}(k) - V_1(k) - i_t(k) R_s \quad (3.38)$$

Where V_1 is the voltage across the RC pair; T_s is the sampling time, assumed constant; $\tau = R_1 \times C_1$ is the time constant of the RC pair; i_t is the current flowing out of the cell; V_t is the cell terminal voltage; and V_{oc} is the cell's open circuit voltage. Combining equations 3.37 and 3.38:

$$V_t(k+1) = V_{oc}(k+1) - (e^{-\frac{T_s}{\tau}} V_1(k) + (1 - e^{-\frac{T_s}{\tau}}) R_1 i_t(k)) - i_t(k+1) R_s \quad (3.39)$$

Replacing $E_t = V_t - V_{oc}$, the difference between terminal voltage and open circuit voltage, we get:

$$E_t = V_t - V_{oc} \quad (3.40)$$

$$E_t(k+1) = -(e^{-\frac{T_s}{\tau}} V_1(k) + (1 - e^{-\frac{T_s}{\tau}}) R_1 i_t(k)) - i_t(k+1) R_s \quad (3.41)$$

Reorganizing equation 3.41, we get:

$$E_t(k+1) = e^{-\frac{T_s}{\tau}} E_t(k) - R_s i_t(k+1) + (e^{-\frac{T_s}{\tau}} R_s - (1 - e^{-\frac{T_s}{\tau}}) R_1) i_t(k) \quad (3.42)$$

$$E_t(k+1) = p_1 E_t(k) + p_2 i_t(k+1) + p_3 i_t(k) \quad (3.43)$$

Where p_1 , p_2 and p_3 are the elements of the parameters vector θ_n :

$$p_1 = e^{-\frac{T_s}{\tau}} \quad (3.44)$$

$$p_2 = -R_s \quad (3.45)$$

$$p_3 = e^{-\frac{T_s}{\tau}} R_s - (1 - e^{-\frac{T_s}{\tau}}) R_1 \quad (3.46)$$

Defining the prediction goal variable y as;

$$y(k) = E_t(k) \quad (3.47)$$

$$y(k) = V_t(k) - V_{oc}(k) \quad (3.48)$$

The measurement vector ϕ_n , parameter vector θ_n , and prediction of the y value $\hat{y}$ can be defined as:

$$\phi_n(k) = [E_t(k-1), i_t(k), i_t(k-1)]^T \quad (3.49)$$

$$\theta_n(k) = [p_1, p_2, p_3]^T \quad (3.50)$$

$$\hat{y}(k) = \phi_n(k)^T \theta_n(k) \quad (3.51)$$

Essentially, using the current parameters vector and after updating the measurement vector, the model will predict the next value of $y(k) = E_t(k)$. Then, the error between the observed y and estimated $\hat{y}$ $e(k) = y - \hat{y}$ is used to update the parameters for the next iteration.

From equations 3.44, 3.45 and 3.46 the electric model parameters can then be deduced as:

$$R_s = -p_2 \quad (3.52)$$

$$R_1 = \frac{p_1 p_2 + p_3}{p_1 - 1} \quad (3.53)$$

$$C_1 = \frac{(1 - p_1)T_s}{(p_1 p_2 + p_3) \ln(p_1)} \quad (3.54)$$

Worth noting that the value for V_{oc} during estimation is not a directly measurable quantity of the cell, therefore when using this algorithm, it is assumed that the value for V_{oc} is estimated elsewhere. In the context provided in this paper, V_{oc} is provided by mapping the value of SOC, produced by the SOC estimation model described in Section 3.2, to its V_{oc} equivalent value, through the intrinsic relationship between SOC-OCV, as described in Section 2.4.1.1.

Chapter 4

Results and Discussion

4.1 State-of-Charge estimation results

4.1.1 Model simulation results

Training and validation dataset

The training and validation dataset was taken from [105], where multiple driving and charging cycles at various temperatures were performed on a new 3Ah LG HG2 Li-ion cell. The authors of the dataset provided a script to generate a similar deep FNN model that they presented in a related work in [103], alongside the same training data and validation data used in that paper to validate the model.

Training methods

The training of the FNNs were performed in MATLAB environment, using custom developed training functions. Four different methods of network weight adjustment were implemented: SGD, SGDM, RMSProp, and ADAM. In Section 3.1.2, a detailed description is given on how to implement these training algorithms.

To ensure fairness in model comparison, each model was trained 3 times with each of the mentioned training methods (SGD, SGDM, RMSProp, ADAM), leading to 12 training repetitions in total for each model. Each training repetition lasted for 100 epochs. Cost function used for training the networks was the instantaneous value of the sum of the quadratic errors, as illustrated in equation 3.5.

Base model

As a reference to compare our proposed models, we ran the script available in [105] to generate the same model used in [103]. This will be referred as the Base Model in this paper. The proposed models in Section 3.2.1 were trained using the same training and validation dataset. However, the moving average of the voltage and current measurements was calculated as specified in Section 3.2.1.

Performance metrics

To evaluate our models, we used four distinct performance metrics: Root Mean Squared Error (RMSE), Median Absolute Error (MAE), maximum estimation error verified (MAX), and number of parameters in the model.

Validation results

	Neurons	RMSE	MAE	MAX	Parameters
One hidden layer models	10	1.384%	0.9177%	8.033%	71
	20	1.284%	0.8423%	11.10%	141
	40	1.205%	0.8360%	10.31%	281
	60	1.142%	0.7686%	8.683%	421
	80	1.126%	0.6866%	21.28%	561
	100	1.096%	0.6992%	10.11%	701
	120	1.097%	0.6639%	13.61%	841
Two hidden layer models	5-5	1.381%	0.9785%	8.282%	66
	10-10	1.251%	0.7957%	6.988%	181
	15-15	1.182%	0.8339%	7.707%	346
	20-20	1.214%	0.8042%	12.08%	561
	25-25	1.127%	0.7440%	6.799%	826
	Base Model	1.374%	0.9306%	12.93%	3466

Table 4.1: Minimum total SOC validation error achieved.

Table 4.1 shows minimum validation error achieved over the total validation data, while Table 4.2 shows validation error achieved at different temperatures for the best performing models.

Model	Temperature	RMSE	MAE	MAX error
Base Model	-10°C	1.405%	1.019%	12.93%
	0°C	1.373%	0.9353%	6.986%
	10°C	1.537%	1.011%	6.204%
	25°C	1.174%	0.7793%	8.401%
25-25 neurons two layer model	-10°C	0.9692%	0.6886%	6.741%
	0°C	1.147%	0.7327%	6.799%
	10°C	1.378%	0.8682%	5.846%
	25°C	0.9565%	0.6842%	4.447%
100 neurons one layer model	-10°C	0.8884%	0.6137%	10.11%
	0°C	1.100%	0.7124%	8.267%
	10°C	1.341%	0.8177%	8.999%
	25°C	0.9879%	0.6477%	8.476%

Table 4.2: SOC validation error at different temperatures.

As seen, using a different averaging method combined with other definitions allowed the proposed FNNs to achieve more accurate results, while requiring significantly less parameters in storage when compared to the Base Model.

Figures 4.1, 4.2 and 4.3 plot the RMSE, MAE, MAX error achieved in validation and the percentage of SOC points for which the SOC prediction error was higher than 4% at each SOC interval, for the chosen models and different training methods used. The x-axis represents the chosen models, as displayed in Table 4.1, and the y-axis the achieved error value. The error achieved by the Base Model in each case is plotted as the blue dashed line, so we can directly compare it to the error achieved by each training method and model. In Figure 4.1 the one layer models are chosen for the comparison, in Figure 4.2 the two layer models are chosen for the comparison, and in Figure 4.3 the best performing models out of the one layer and two layer models are chosen for the comparison.

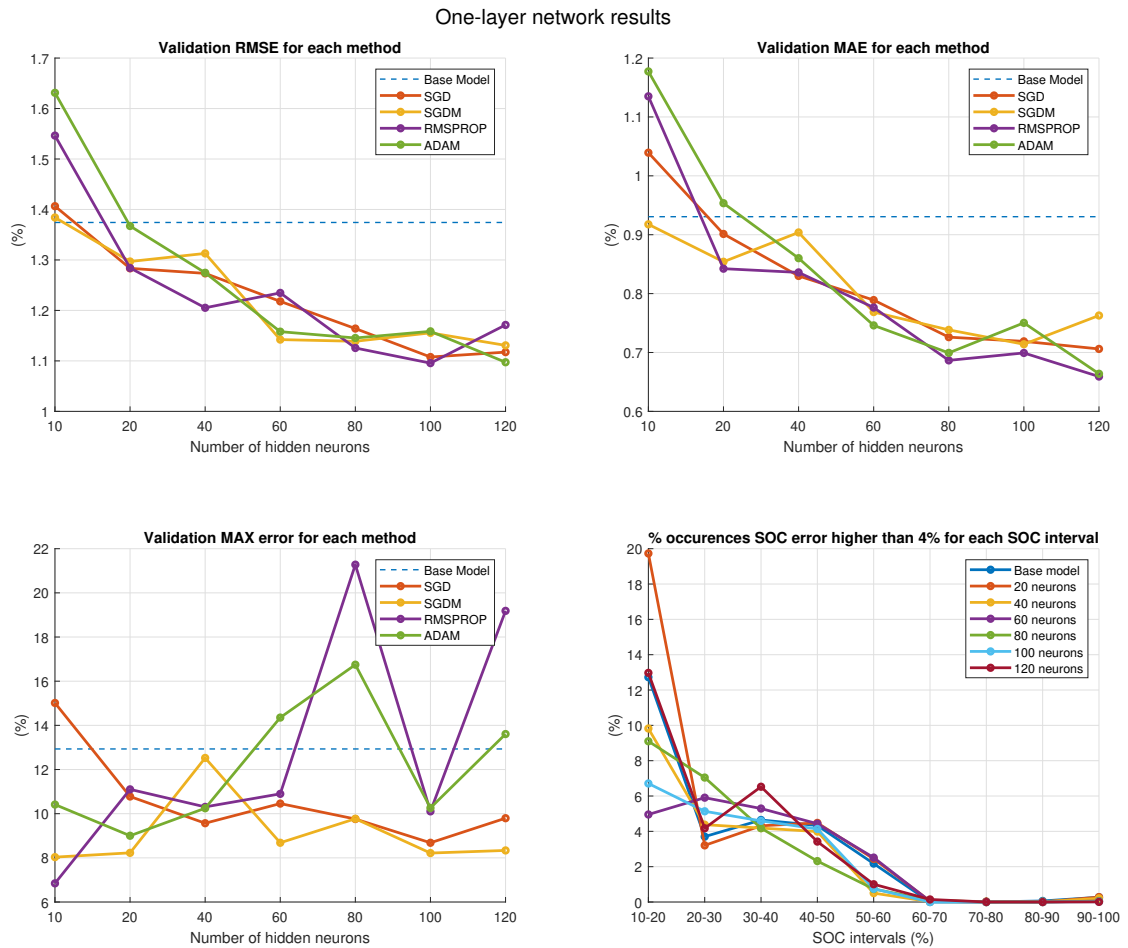


Figure 4.1: Comparison validation results and training methods - one layer models.

Analysing the results obtained for the one layer models in Figure 4.1, it can be seen that a one hidden layer network with 20 hidden layer neurons and 141 parameters in total, managed to

achieve a lower RMSE, MAE and MAX error values than the Base Model for all training methods. This is a very dramatic improvement over the Base Model, since the number of parameters, 141, required in storage is significantly lower when compared to the Base Model's 3466 parameters. In this sense, a general improvement was verified when initially increasing the number of hidden layer neurons of the model. However, after reaching 80 neurons, no further significant improvement was verified when increasing the number of neurons.

In general, the SGD and SGDM methods resulted in less validation MAX error when compared to using the RMSProp and ADAM methods, while the RMSProp and ADAM methods resulted in lower RMSE and MAE values than the SGD and SGDM methods. For all models, it can be observed that the method struggles more in the lower half of SOC intervals (10% - 50%), where the SOC prediction error was higher than 4% much more often when compared to the results in the upper half of SOC intervals (50% - 100%). The best performing one layer model out of all tested was the one hidden layer model with 100 hidden neurons, when trained with RMSprop method. This model managed to achieve RMSE value of 1.096%, a reduction of 20.23% compared to the RMSE value of 1.374% achieved by the Base Model.

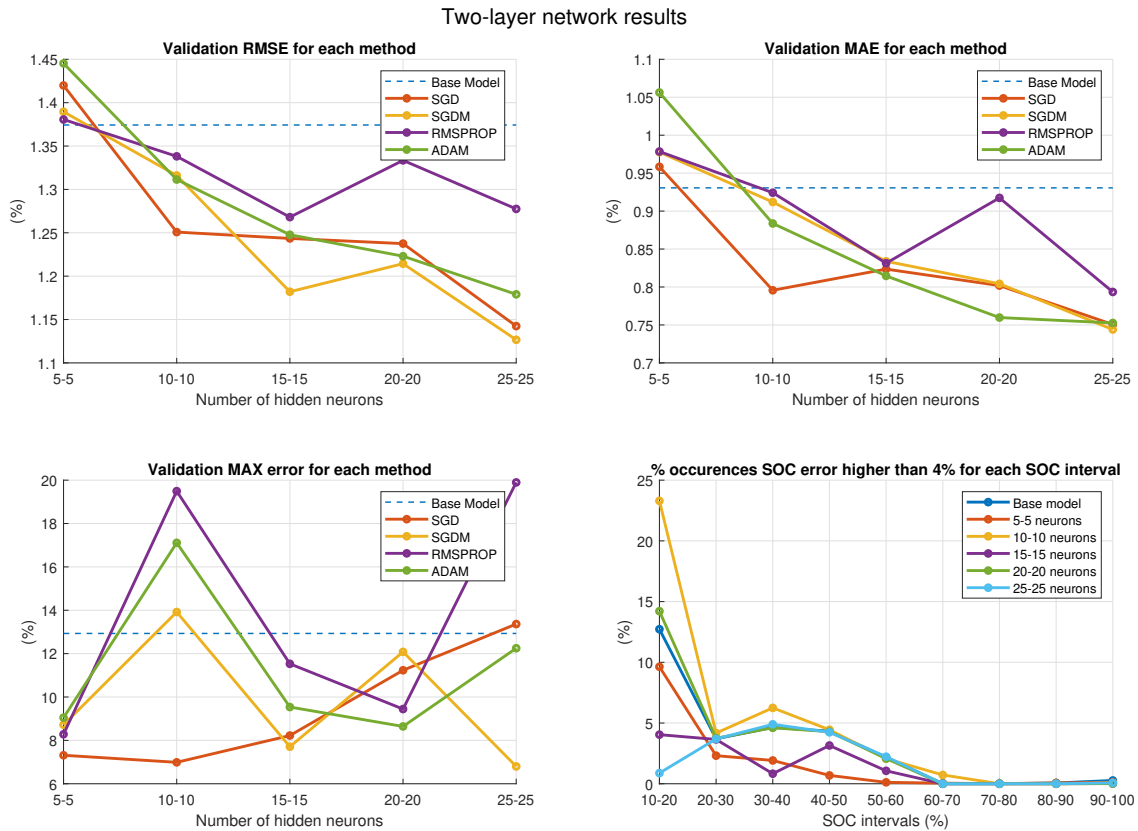


Figure 4.2: Comparison validation results and training methods - two layer models.

Analysing the results achieved for the two layer models in Figure 4.2, it can be seen that a two hidden layer network with as few as 10 neurons per hidden layer managed to achieve lower RMSE and MAE values than the Base Model, for all training method considered. In general, for

this case, the SGD and SGDM methods achieved lesser RMSE and MAE values than the ADAM and RMSProp methods. Similarly to the one hidden layer models results, the models struggle more in the lower half of SOC interval, while remaining very accurate in the upper half of SOC interval. The best performing two layer model out of all tested was the two hidden layer model with 25 hidden neurons per layer, when trained with SGDM method. This model managed to achieve RMSE value of 1.127%, a reduction of 17.98% compared to the RMSE value of 1.374% achieved by the Base Model.

Analysing the results obtained when comparing the best performing models out of the one layer and two layer models in Figure 4.3, it can be seen that the two hidden layer networks, in general, performed worse than the one layer models in terms of RMSE and MAE. However, in general, the two layer models managed to achieve less MAX error when compared to the one layer models. The best performing model out of all tested remains the one hidden layer model with 100 hidden neurons, when trained with RMSprop method.

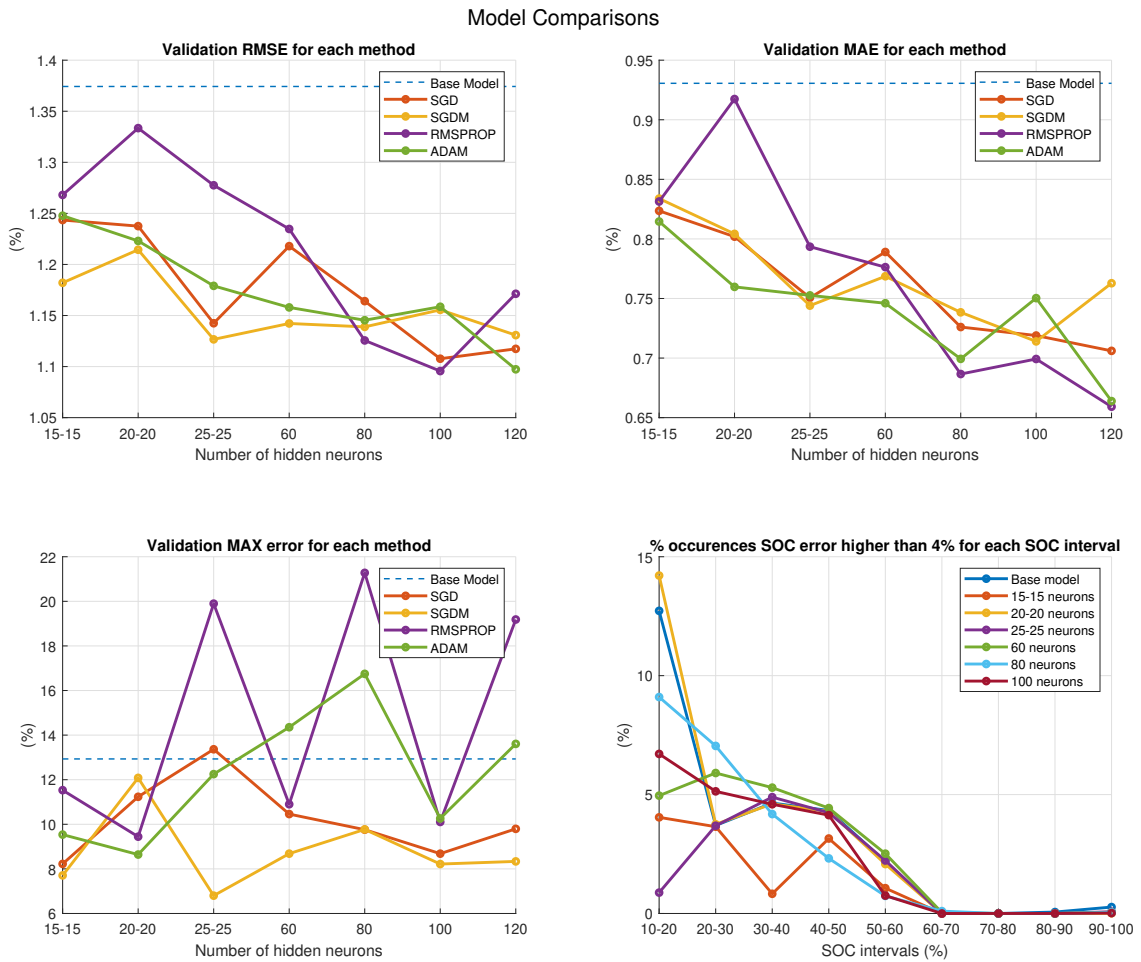


Figure 4.3: Comparison validation results and training methods.

Figure 4.4 plots the validation results at 25°C for the 25-25 neurons two hidden layer model, where the maximum absolute error verified was 4.447%. For the 25-25 neurons two hidden layer

model, at all considered temperatures and drive cycles, an accurate prediction of SOC is verified, with the maximum absolute error never surpassing 6.799%.

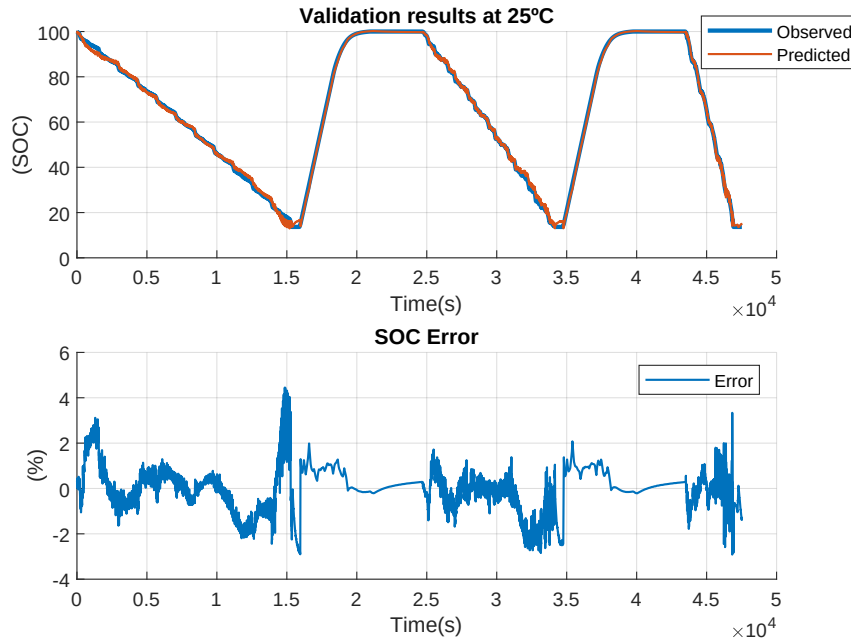


Figure 4.4: Model 25-25 neurons - Validation results at 25°C.

4.1.2 Experimental results

In this section, the SOC estimation results achieved using the battery experimental data collected by Continental Engineering Services are presented. The battery data made available consists primarily of CC-CV charge profiles and constant current discharge profiles, taken at a constant temperature of 23°C.

Training and validation dataset

For training and validating the network, two cycles were used for training the model while one cycle was chosen for validation of the model. The data used was sampled at a period of 1 second, with a single-precision floating-point resolution.

The training cycle 1, presented in Figure 4.5, consists of a 0.8A constant current discharge of the battery, mixed with some short duration current pulses of 10A applied at 100, 90, 80, ..., 20 and 10 % SOC.

The training cycle 2, presented in Figure 4.6, consists of multiple CC-CV charging profiles at a rate of 2A, and multiple constant current discharge profiles of the cell, at a rate of 10A and 0.8A.

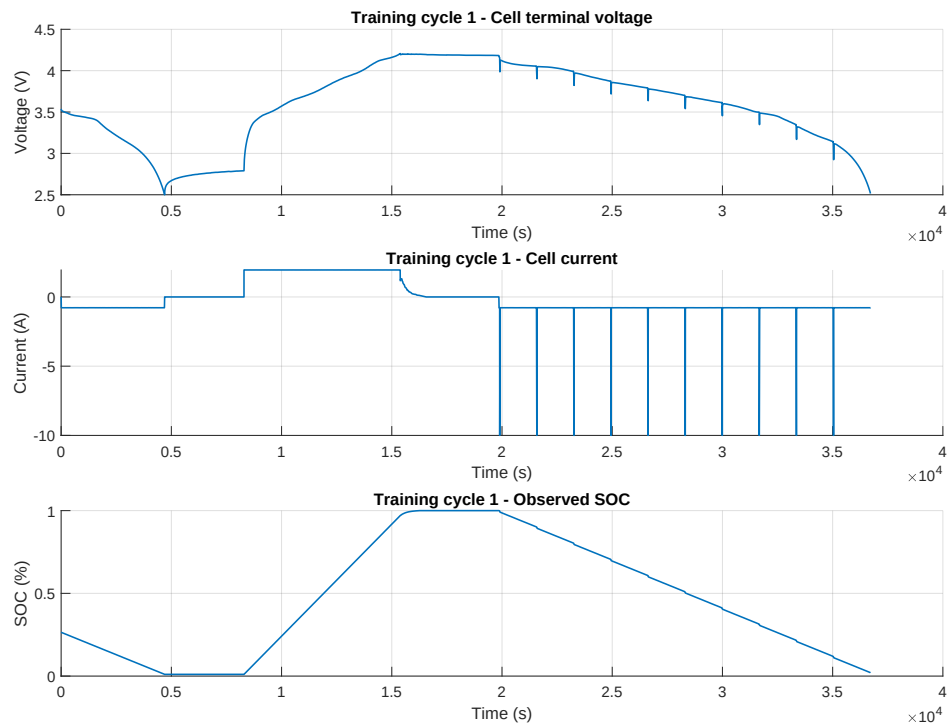


Figure 4.5: Training cycle 1 - Voltage, current and SOC data.

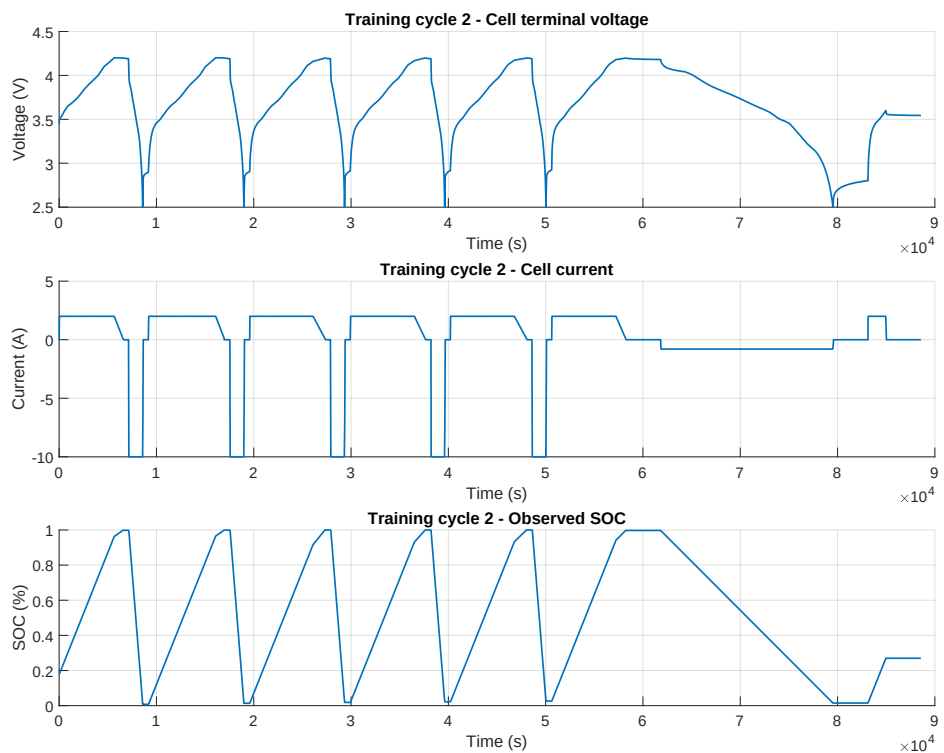


Figure 4.6: Training cycle 2 - Voltage, current and SOC data.

The validation cycle, presented in Figure 4.7, consists of multiple CC-CV charging profiles at a rate of 2A, and multiple constant current discharge profiles of the cell, at a current rate of 10A, 15A, 20A and 25A.

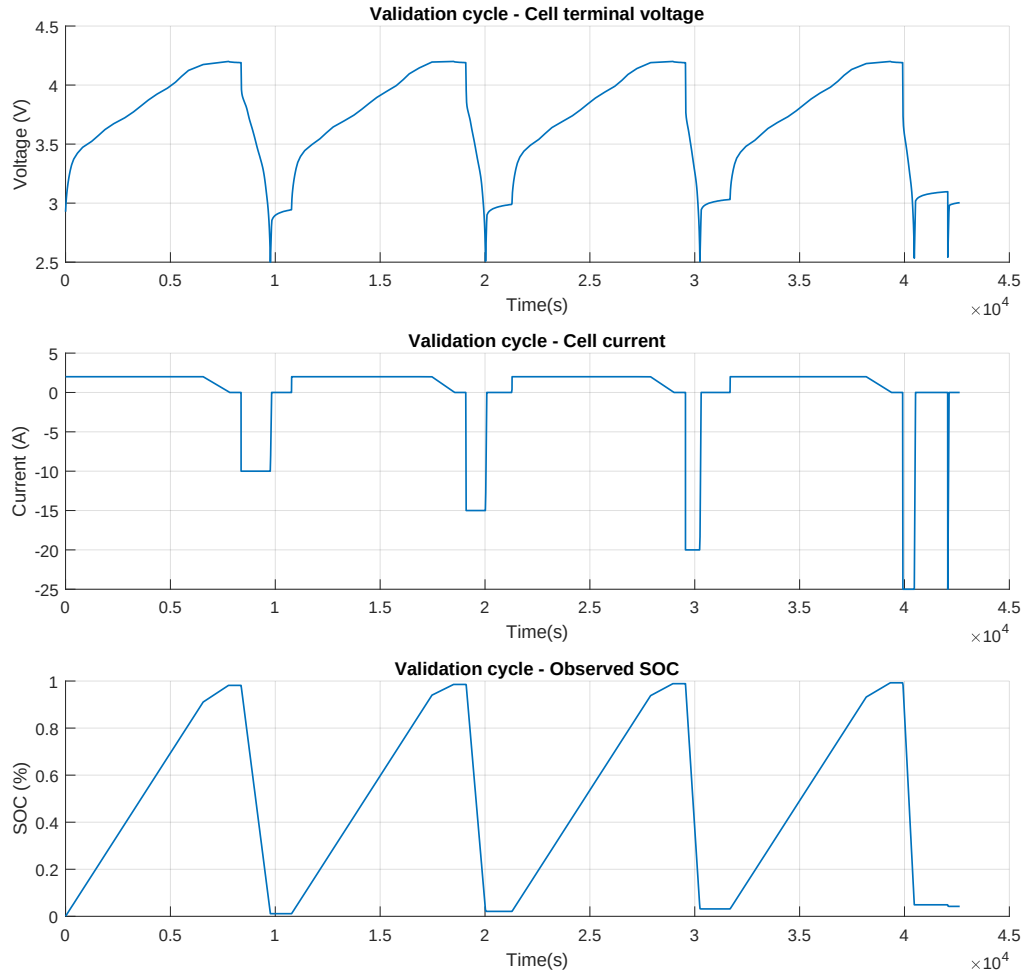


Figure 4.7: Validation cycle - Voltage, current and SOC data.

SOC model used

The general model used for SOC estimation is the same as presented in Section 3.2. In this case, only a 10 hidden layer neuron model was trained and validated using the data made available by Continental Engineering Services.

The model was trained in MATLAB environment using the SGDM algorithm, described in Section 3.1.2. Training of the model was repeated 20 times, with each training repetition running for 100 epochs. The results achieved with the best performing model out of all repetitions are presented next.

To evaluate the accuracy of the model, the same performance metrics (RMSE, MAE, MAX error) presented in the previous Section 4.1.1 are used in this case as well.

Validation results

Figure 4.8 presents the SOC prediction results and SOC error achieved for the validation cycle. As can be seen, despite during training the maximum discharge current presented to the model being only 10A, the model managed to accurately predict SOC for higher discharge current profiles during validation at 15A, 20A and 25A, showcasing the model's generalization capacity and capability of modeling the cell's internal parameters and behaviour. During validation, the model achieved RMSE of 1.467%, MAE of 1.205% and MAX absolute error of 5.55%, demonstrating its effectiveness in modeling the battery's SOC behaviour.

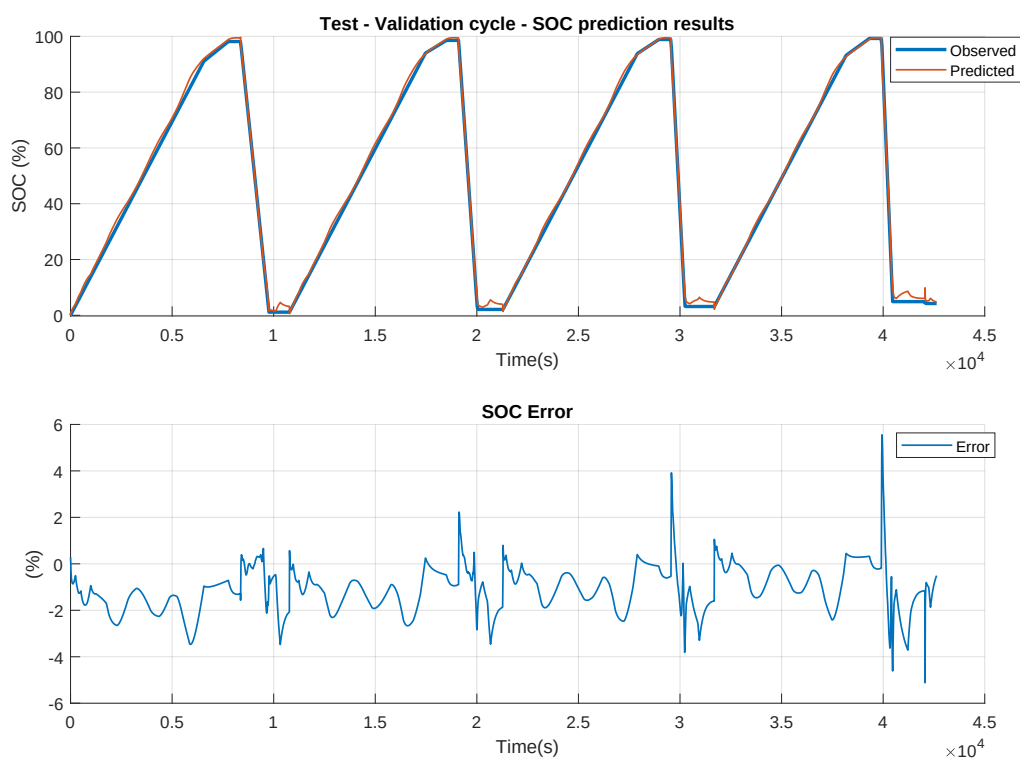


Figure 4.8: SOC prediction results - Validation cycle.

4.2 State-of-Health estimation results

Description of the models trained and dataset used were given in Section 3.3.

Performance metrics

To evaluate our models, we used four distinct performance metrics: Root Mean Squared Error (RMSE), Median Absolute Error (MAE), maximum estimation error verified (MAX), and number of parameters in the model.

Validation results

Table 4.3 shows minimum validation error achieved over the total cell #5 validation data. As seen, low error is verified for each of the models, with the maximum RMSE value of 2.1682% out of all models verified for the 10 neurons model, while the lowest RMSE value of 1.9105% verified belonging to the 50 neurons model. Another important remark about the results achieved, is that when increasing the number of neurons per hidden layer above 30, no significant accuracy improvement was verified on the SOH prediction task.

Neurons	RMSE	MAE	MAX	Parameters
10	2.1682%	1.8392%	3.9288%	91
20	2.0667%	1.7909%	3.5808%	181
30	2.0714%	1.7823%	4.1361%	271
40	1.9313%	1.6755%	3.7288%	361
50	1.9105%	1.6433%	3.5326%	451
60	1.9125%	1.6581%	3.8676%	541

Table 4.3: Minimum total SOH validation error achieved for cell #5.

Figure 4.9 plots the estimation results verified for all trained models alongside the true SOH observed for cell #5. As it can be seen, a relatively accurate SOH estimation is verified for each model. All models behaved in a similar way, where a continuous error is verified for cycles below 20 and for the later cycles above 120. For cycles 20 to 120, accurate results were verified for all models.

Table 4.4 shows minimum validation error achieved over the total cell #6 validation data. In this case, a higher error is verified in general when compared to the results obtained when cell #5 data was chosen for validation, particularly when comparing the MAX error achieved. The worst performing model was the 20 neuron model, with a RMSE value of 3.1042%, while the best performing model in this case was the 40 neuron model. Mixed results were verified when increasing the number of neurons per hidden layer above 20, where a 40 neuron model performed better than the 50 or 60 neuron models.

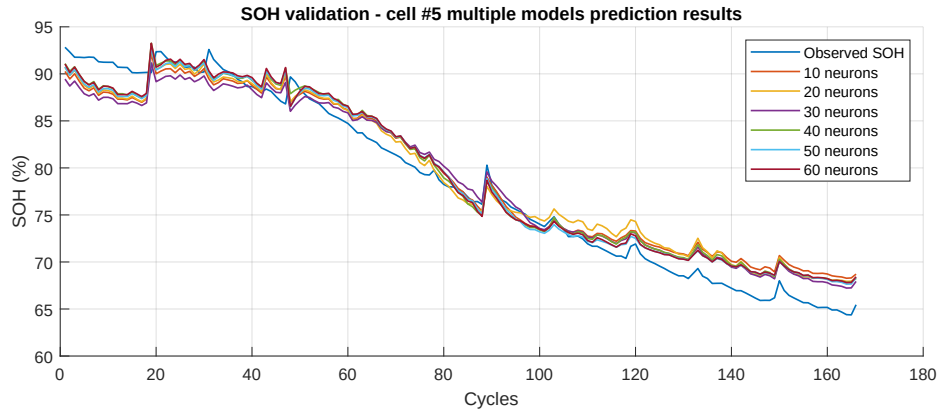


Figure 4.9: SOH results - Cell #5 multiple models prediction results.

Neurons	RMSE	MAE	MAX	Parameters
10	3.1025%	2.5656%	8.2215%	91
20	3.1042%	2.6408%	8.0269%	181
30	2.9015%	2.5226%	7.2040%	271
40	2.6329%	2.2984%	6.8961%	361
50	2.9214%	2.5936%	7.1547%	451
60	2.6991%	2.3688%	6.9354%	541

Table 4.4: Minimum total SOH validation error achieved for cell #6.

Figure 4.10 plots the estimation results verified for all trained models alongside the true SOH observed for cell #6. In this case, all models struggled to accurately predict the SOH for the later cycles above 100.

Table 4.5 shows minimum validation error achieved over the total cell #7 validation data. When using cell #5, #6 and #18 for training the network and cell #7 for validation, very inaccurate results were verified, with the lowest RMSE value of 4.1264% belonging to the 10 neuron model. As it can be seen, using cell #5 as validation and cell #6, #7 and #18 yielded much more accurate results for SOH prediction in contrast to using cell #7 as validation and remaining cells in training.

Neurons	RMSE	MAE	MAX	Parameters
10	4.1264%	3.3973%	8.7802%	91
20	4.2381%	3.4988%	8.9465%	181
30	4.2462%	3.6687%	8.5937%	271
40	4.2137%	3.5221%	8.8679%	361
50	4.1725%	3.7242%	8.0599%	451
60	4.1769%	3.4940%	8.7680%	541

Table 4.5: Minimum total SOH validation error achieved for cell #7.

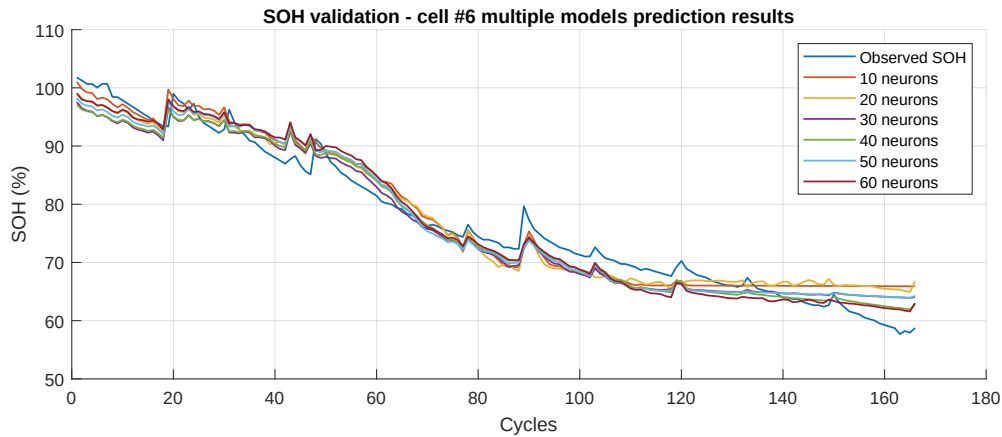


Figure 4.10: SOH results - Cell #6 multiple models prediction results.

Figure 4.11 plots the estimation results verified for all trained models alongside the true SOH observed for cell #7. As was expected after analysing Table 4.5, all models trained failed to accurately predict SOH throughout all cycles.

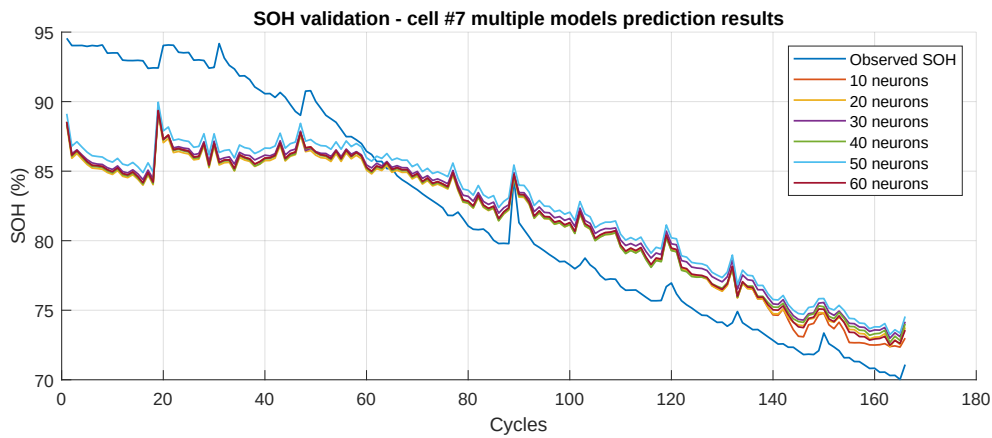


Figure 4.11: SOH results - Cell #7 multiple models prediction results.

Table 4.6 shows minimum validation error achieved over the total cell #18 validation data. Analysing the data, in general the error value achieved in each model was lower than the error achieved when using the cell #6 for validation, with the lowest RMSE value of 2.5319% verified for the 20 neuron model. However, the MAX error for each model was higher in this case compared to the cell #6 case. No significant accuracy improvement was verified when increasing the number of neurons of the models.

Figure 4.12 plots the estimation results verified for all trained models alongside the true SOH observed for cell #18. As can be seen, in general a somewhat accurate prediction of SOH was

Neurons	RMSE	MAE	MAX	Parameters
10	2.6431%	2.1584%	8.8839%	91
20	2.5319%	2.0965%	8.5625%	181
30	2.5950%	2.2111%	8.6221%	271
40	2.5908%	2.0867%	8.7915%	361
50	2.6433%	2.2054%	7.1786%	451
60	2.5783%	2.1213%	8.9176%	541

Table 4.6: Minimum total SOH validation error achieved for cell #18.

made. However, in some cycles a high error is visible in the SOH prediction, justifying the high MAX error verified in Table 4.6 for most models.

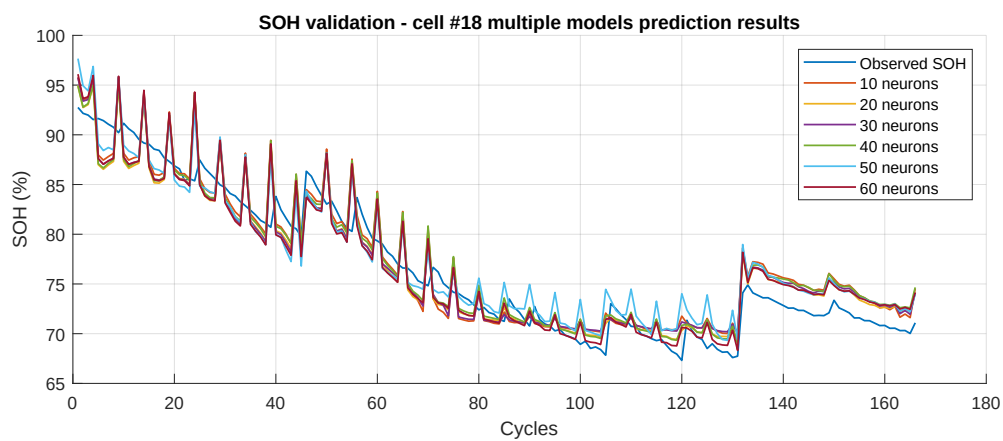


Figure 4.12: SOH results - Cell #18 multiple models prediction results.

As can be seen, the models performed at their best when trained using cell #6, #7 and #18 data and validated using cell #5 data, where a 50 neuron model had a MAX error of 3.5326% when predicting the SOH of the cell for all cycles.

4.3 Cell model parameter estimation results

Validation data

All validation data used for the parameter estimation algorithm was taken from [105], the same dataset used to train and validate the SOC estimation model presented in 3.2. To validate the algorithm, UDDS and US06 cycle data at 25°C, 10°C, 0°C and -10°C from the dataset were used. In these tests, the value of the OCV required for the algorithm is given by a Look-Up-Table (LUT) that relates the value of SOC to OCV. The respective value of SOC is given by the proposed SOC estimation model presented in 3.2.

To validate the algorithm, only the voltage prediction error generated by the model will be analysed. If the battery model has low voltage prediction error, then the parameters estimated by the algorithm are considered valid to represent the parameters of the battery's equivalent model.

To evaluate the error produced by the model, three distinct performance metrics are used: RMSE (Eq. 3.11), MAE (Eq. 3.12) and MAX absolute error achieved during estimation.

Validation results

Table 4.7 showcases the RMSE, MAE and MAX absolute error obtained during estimation. For the UDDS cycle data, a general low error is verified when observing the resultant RMSE and MAE values, in contrast to the MAX error verified which is significantly higher. For the US06 cycle data, a similar behaviour is observed, albeit with higher error in all error components when compared to the UDDS cycle results. This is somewhat expected, since the US06 is a much more dynamic cycle, with higher and more frequent current spikes, than the UDDS cycle. For both cycles, the estimation error increases dramatically as the temperature lowers, revealing some inadequacy to model the battery cell at lower temperatures. However, at ambient temperature when subjected to the UDDS cycle, the method achieved very low prediction error, with a maximum verified error of 31.1217 mV.

Cycle	Temperature	RMSE (mV)	MAE (mV)	MAX error (mv)
UDDS Cycle	-10°C	4.5669	1.9462	78.999
	0°C	3.0658	1.2382	53.3703
	10°C	2.0332	0.8058	41.336
	25°C	1.3121	0.5460	31.1217
US06 Cycle	-10°C	7.7828	3.5501	167.220
	0°C	6.1647	2.7605	97.6445
	10°C	5.8106	2.5786	75.0477
	25°C	4.1752	1.8677	72.149

Table 4.7: RLS-FF Model voltage prediction error for multiple temperatures.

Figure 4.13 plots the prediction results and error for the UDDS cycle.

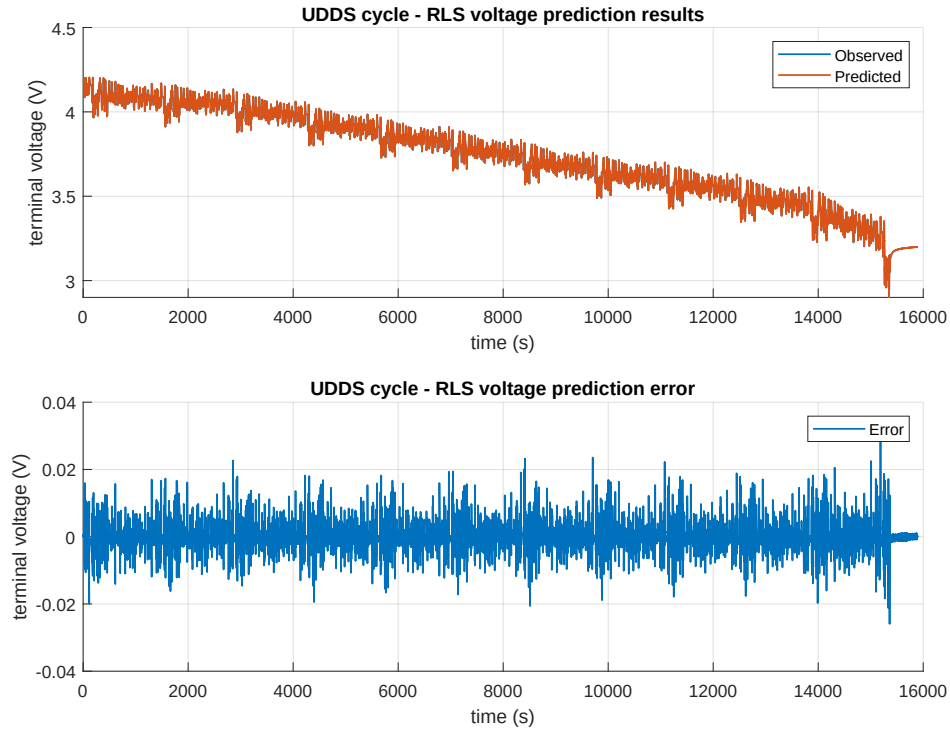


Figure 4.13: Voltage prediction results - UDDS Cycle at 25°C.

Analysing the results, we can confirm that in general a low error is verified, with some periods where the error is considerably higher than in the remaining periods.

Figure 4.14 plots the parameter estimation results of the RLS-FF algorithm during the UDDS cycle.

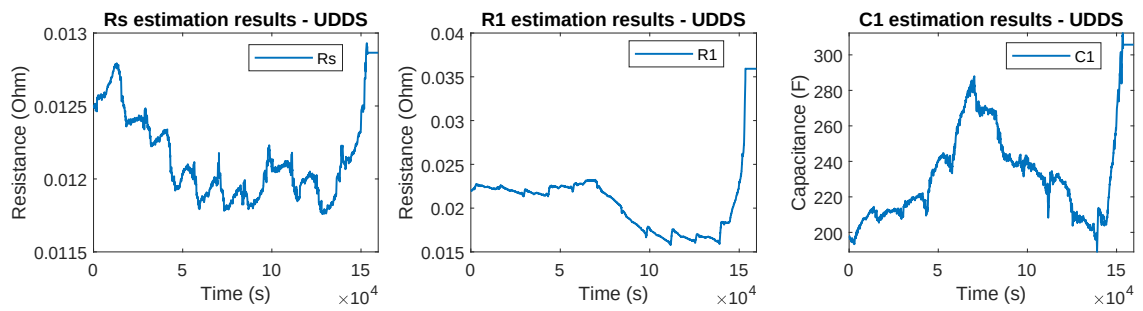


Figure 4.14: Parameter prediction results - UDDS Cycle at 25°C.

The parameters estimated are the series resistance R_s and the RC pair R_1 and C_1 , as described in Section 3.4. In general, the battery's parameters are mainly influenced by the current SOC, SOH, temperature of the cells, and current flowing through the cells. In this case, the SOH of the cell is a constant value, since no significant battery degradation is verified throughout the UDDS cycle. The results seem to make sense, since a general increase in each parameter is verified as the

SOC lowers throughout the UDDS cycle. Quick changes in the parameters values are also justified by the dynamic changes in current applied to the battery throughout the cycle.

Figure 4.15 plots the prediction results and error for the US06 cycle. In this case it can be seen that the prediction is relatively accurate, with some spikes occurring in the verified error, justifying the vast difference between the MAX error value to the RMSE and MAE values achieved on Table 4.7.

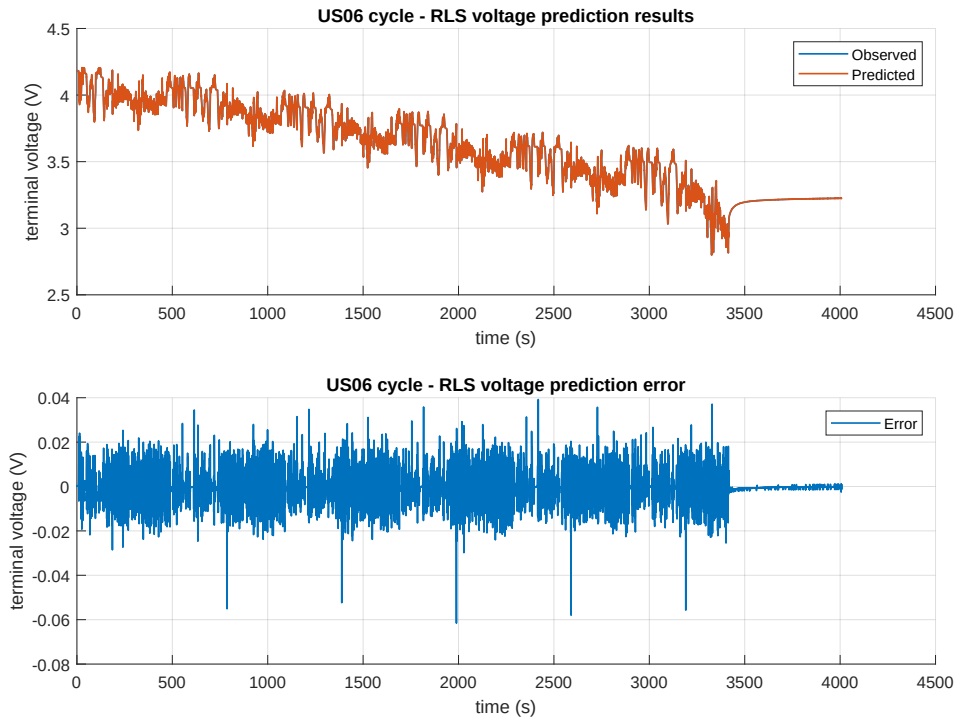


Figure 4.15: Voltage prediction results - US06 Cycle at 25°C.

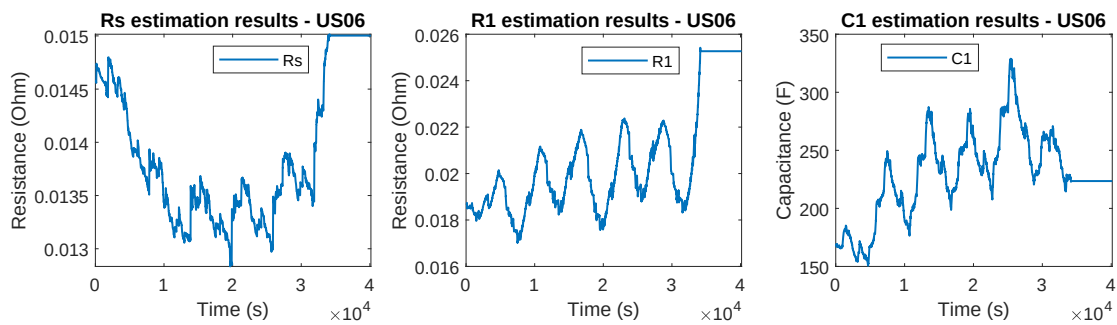


Figure 4.16: Parameter prediction results - US06 Cycle at 25°C.

Figure 4.16 plots the parameter estimation results of the RLS-FF algorithm during the US06

cycle. Similarly to the estimation results during the UDDS cycle, a general increase of the parameters is verified as the SOC lowers throughout the cycle. However, in this case each parameter varies more frequently during the cycle. This is justified by the much more dynamic nature of the current profile applied, since more frequent and higher valued current spikes are verified during the US06 cycle compared to the UDDS cycle.

Chapter 5

HW setup designed

In this chapter, the HW setup designed for testing the BMS is presented. This HW setup was designed to be capable of emulating the voltage and current behaviours of lithium-ion cells for practical testing of the BMS. A general diagram of the system is presented in Figure 5.1.

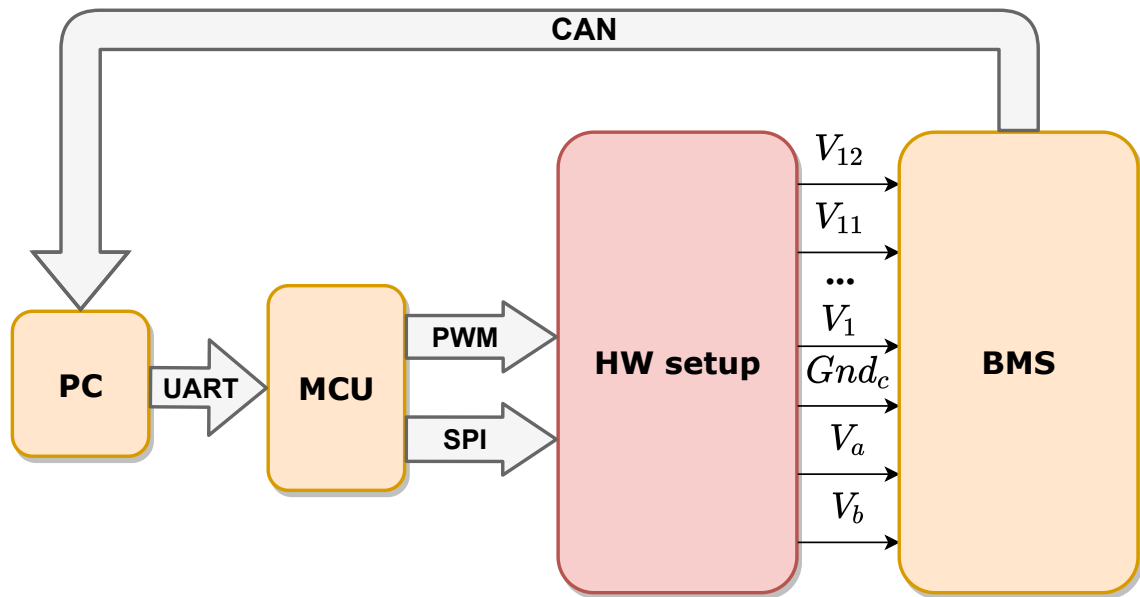


Figure 5.1: HW setup overview.

The plan was to send the voltage and current reference points for each cell from the PC to a Microcontroller Unit (MCU). Then, the MCU would generate PWM signals and communicate via SPI to the HW setup to adjust the voltage values that the setup outputs. The voltage signals outputted by the setup are directly connected to the BMS ADC channels. The BMS then sends relevant data to the PC via CAN interface. This type of setup allows us to generate voltage and current values recorded from tests performed to a real lithium-ion cell, allowing us to replicate a real cell's behaviour for BMS testing.

In this case, for the 48V BMS used by the company, the signals required to emulate are: 12 voltage channels, one for each lithium-ion cell in the pack; a ground channel, connected to the base of the battery pack; and 2 differential voltage channels, supposed to be connected to a shunt resistor in series with the battery pack, to measure the current flowing through the cells.

5.1 Cell voltage emulation

The circuit used to generate voltage value for a cell is presented in Figure 5.2.

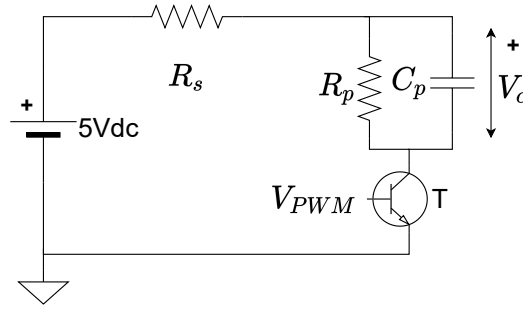


Figure 5.2: Circuit for single cell voltage V_c emulation.

In this circuit, adjusting the duty-cycle D of the V_{PWM} signal coming from the MCU allows control of the cell V_c voltage value at the terminals of the capacitor C_p . To determine the relation between the duty-cycle D and the resulting value for V_c , first it is required to model the change in V_c when the transistor is either on or off. Figures 5.3 and 5.4 display the resulting circuits when the transistor is closed or open, respectively.

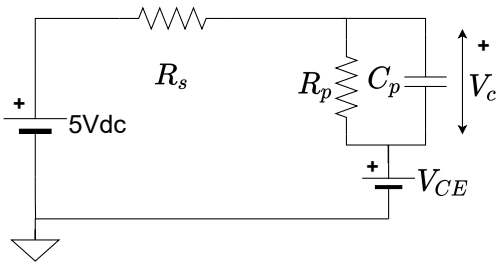


Figure 5.3: Circuit when transistor is closed.

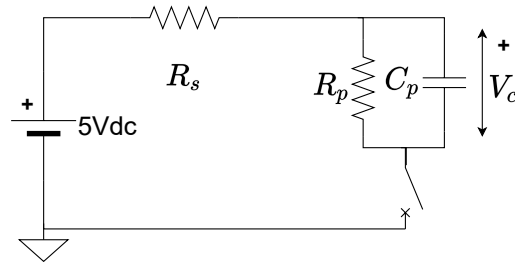


Figure 5.4: Circuit when transistor is open.

Where V_{CE} represents the voltage drop at the transistor when the transistor is closed/conducting. The dynamic equations that represent the behaviour of V_c are given as:

$$\text{Transistor closed: } \frac{dV_c^{closed}}{dt} = \frac{5 - V_c - V_{CE}}{R_s \times C_p} - \frac{V_c}{R_p \times C_p} \quad (5.1)$$

$$\text{Transistor open: } \frac{dV_c^{open}}{dt} = -\frac{V_c}{R_p \times C_p} \quad (5.2)$$

For a small enough period T of the PWM waveform driving the transistor, the variation $\frac{dV_c}{dt}$ can be approximated as:

$$\text{Transistor closed: } \frac{dV_c^{closed}}{dt} \approx \frac{5 - V_c^{avg} - V_{CE}}{R_s \times C_p} - \frac{V_c^{avg}}{R_p \times C_p} \quad (5.3)$$

$$\text{Transistor open: } \frac{dV_c^{open}}{dt} \approx -\frac{V_c^{avg}}{R_p \times C_p} \quad (5.4)$$

Where V_c^{avg} is the averaged value of V_c during the period T . The final resulting value of V_c^{avg} for a defined duty-cycle D is achieved when the sum of the voltage variations of V_c while the transistor is closed or open, ΔV_c^{closed} and ΔV_c^{open} , is equal to 0, $\Delta V_c^{closed} + \Delta V_c^{open} = 0$. Specifying duty-cycle D as the percentage of time during the period T where the transistor is closed, these voltage variations are calculated as:

$$\Delta V_c^{closed} = \frac{dV_c^{closed}}{dt} \times D \times T \quad (5.5)$$

$$\Delta V_c^{open} = \frac{dV_c^{open}}{dt} \times (1 - D) \times T \quad (5.6)$$

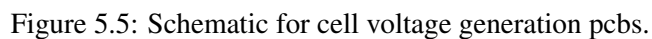
Solving $\Delta V_c^{closed} + \Delta V_c^{open} = 0$ in order to D , the duty-cycle D required to obtain a required V_c^{avg} can then be determined as:

$$D = \frac{V_c^{avg}}{5 - V_c^{avg} - V_{CE}} \times \frac{R_s}{R_p} \quad (5.7)$$

Therefore, after specifying a voltage value for V_c^{avg} , the duty-cycle D required to achieve that voltage value can be calculated.

To emulate all cells connected in series in a 48V battery pack for the BMS, the positive terminal of the V_c voltage of each cell must be connected to the negative terminal of the V_c voltage of the next cell, so that the voltage at the top of the pack can reach values between 30V and 50V relative to the ground of the cells. To achieve this setup, each of the 5Vdc supply voltages for each cell must be isolated from each other, so that each cell can be controlled independently and the differential voltages of each cell can be stacked on top of each other to emulate a battery pack with multiple cells connected in series. Due to the difference in grounds and relatively high voltage difference, the MCU's PWM signal must also be isolated for driving the cell's transistor.

The final schematic for the pcb design is presented in Figure 5.5. The TEL_5-1211 components are 12V-5V isolated converters, and the SSA12 are manual Single Pole Double Throw (SPDT) switches, allowing the ground of each cell to be connected to the positive terminal of the V_c voltage of the previous cell, or to the negative/ground terminal of the previous cell's V_c . Isolation between the MCU's PWM signals and the cells are assured by the use of the high-speed 6N136 optocouplers.



For each pcb, it was decided to place 4 cells, so that the resulting pcb would not be too big. The signals "GND_Out" and "GND_In" are available so that multiple pcbs can be connected in series in order to get more cells connected in series. In this case, since there are 12 voltage channels, only 3 pcbs are required.

The resulting mounted pcb is presented in Figure 5.6. Wires were welded to the inputs/outputs of the pcb due to the ordered connectors not arriving in time.

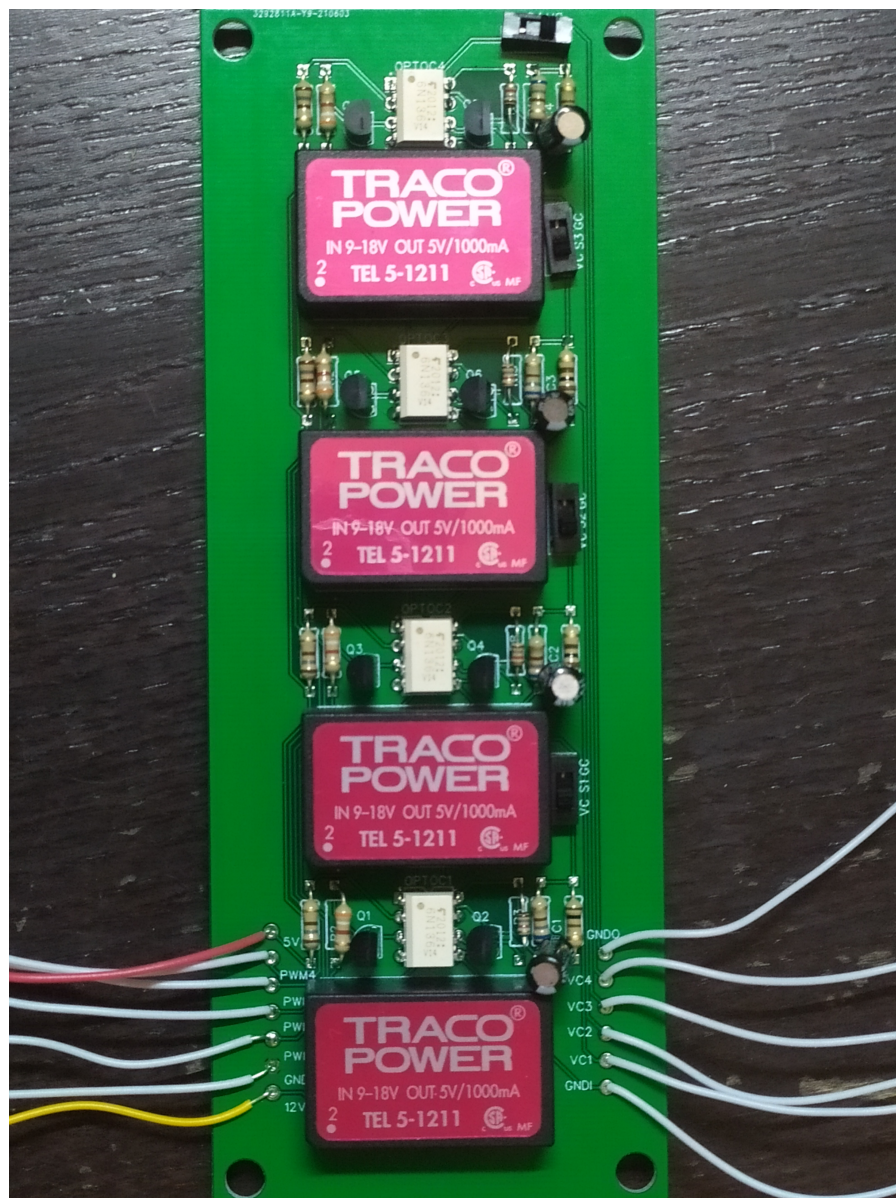


Figure 5.6: PCB for cell voltage generation.

5.1.1 Experimental results

The first test was to set the desired voltage $V_c^{avg} = 4V$. The captured voltage waveform is presented in Figure 5.7.

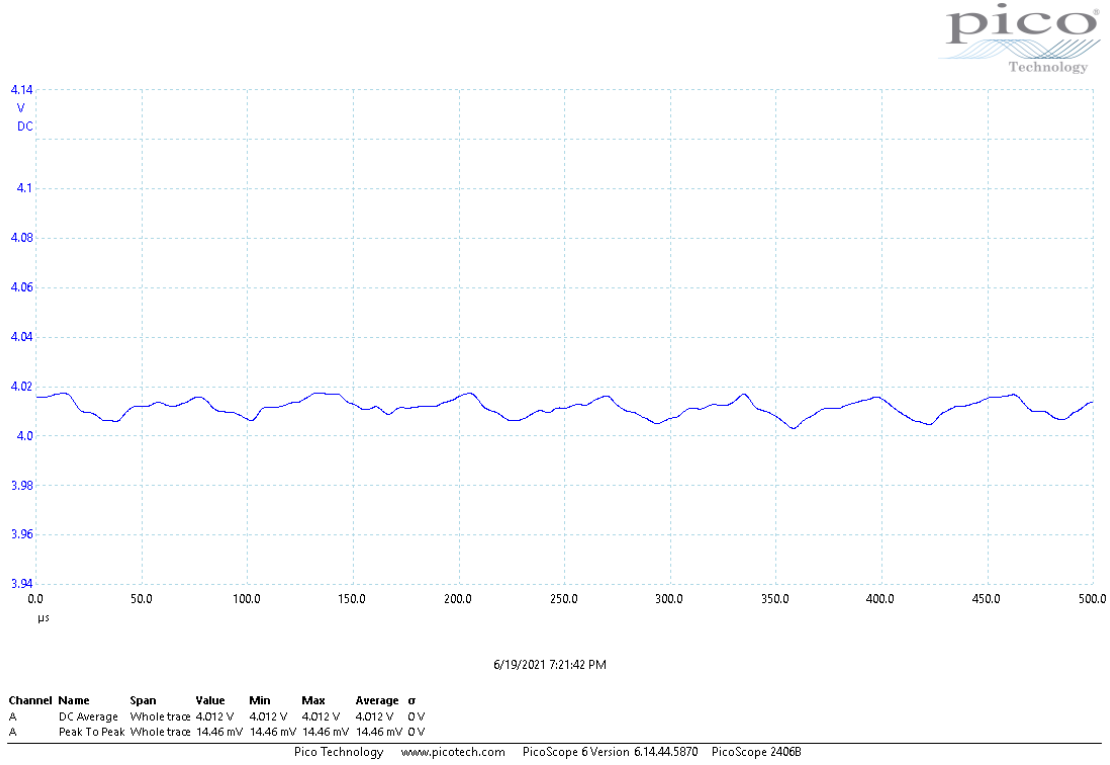


Figure 5.7: Steady 4V cell output.

As can be seen, the measured DC average of the signal is 4.012 Volt, a small error of 12mV compared to the desired output. As the designed system has no feedback implemented, a small steady state error is to be expected. The measured peak-to-peak voltage V_{pk-pk} of the signal is 14.46mV, meaning a small ripple voltage V_r of $\frac{V_{pk-pk}}{2} = 7.23mV$. This value could be further decreased by increasing the frequency of the PWM waveform or increasing the capacity of the capacitor C_p . In this case, the frequency of the PWM was set to 15.625 kHz, and the capacity $C_p = 22\mu F$.

The second test was to measure the voltage output rise time from $V_c^{avg} = 2.5V$ to $V_c^{avg} = 4.2V$. The captured voltage waveform is presented in Figure 5.8.

Analysing the image, after approximately 12ms, the output voltage had already reached its final averaged value of 4.236V, with a small error of 36mV compared to the desired output of 4.2V. The initial averaged voltage of 2.461V also presented a error of 39mV compared to the desired initial output of 2.5V. Keeping in mind that the sample period of the BMS is set at 100ms, $12ms < 100ms$, meaning that the setup has more than enough time to set the voltage output to its final desired value between BMS voltage samples.

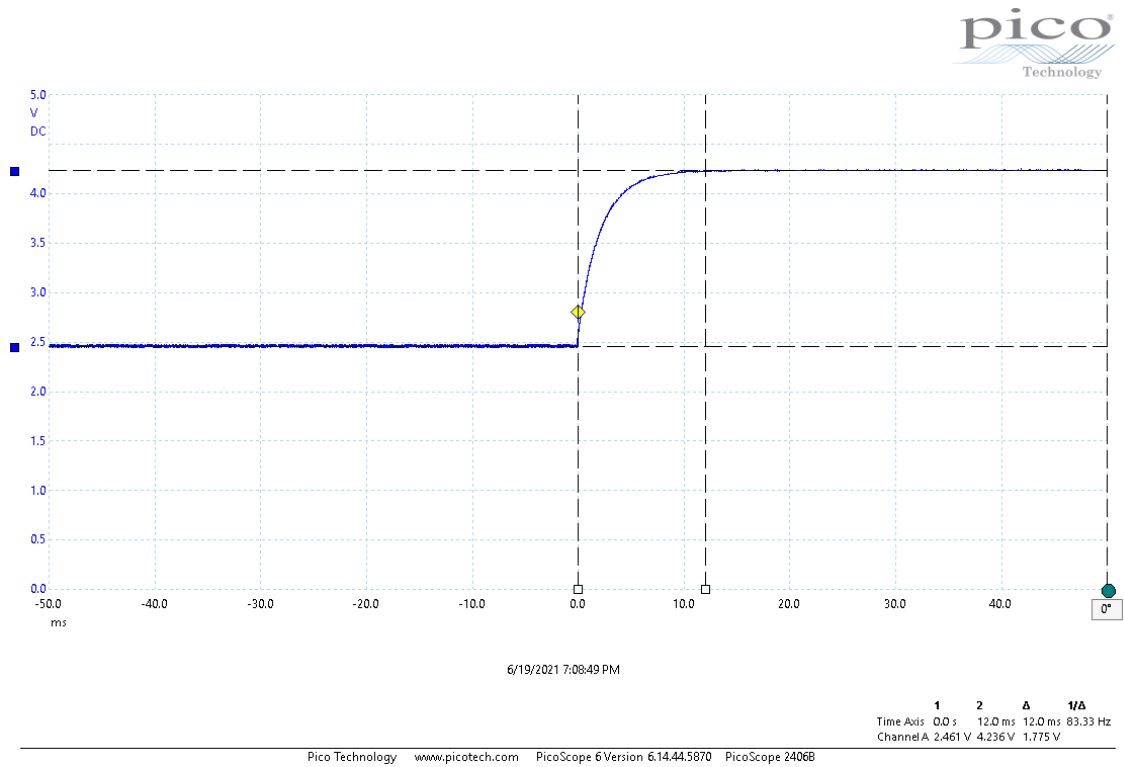


Figure 5.8: Rise from 2.5V to 4.2V cell output.

The third test was to measure the voltage output from 4 stacked cells, from $V_c^{avg} = 2.5V$ to $V_c^{avg} = 4V$. The captured voltage waveforms from the 4 cells is displayed in Figure 5.9.

The blue line, channel A, plots the voltage of the first cell relative to the ground of the cells; the red line, channel B, plots the voltage measured at the second cell relative to the ground of the cells; the green line, channel C, plots the voltage at the third cell; the yellow line, channel D, plots the voltage at the fourth cell relative to the ground of the cells. As can be seen, the circuit is working as planned, as each cell voltage is adding up in series while the differential voltage between cells remains at the desired output value defined, albeit with some small error. The DC average values presented in the image are measured only between the rulers set at 15ms and 25ms, to measure the final averaged value of the cell voltage.

The final test was to emulate a full discharge cycle on the setup. To achieve this, the PC would periodically send the desired voltage values observed from a discharge test performed on a real lithium-ion cell to the MCU, where it would adjust its PWM signals sent to the HW setup in order to generate the output voltage. The results achieved are presented in Figure 5.10.

The discharge cycle ran for approximately 33 minutes, and consisted of a continuous discharge of a lithium-ion cell at a constant rate of 10A, including a relaxation period at the end where no current was applied for a while. From the results achieved, it can be seen that the setup is capable of replicating the voltage behaviour of a real lithium-ion cell.

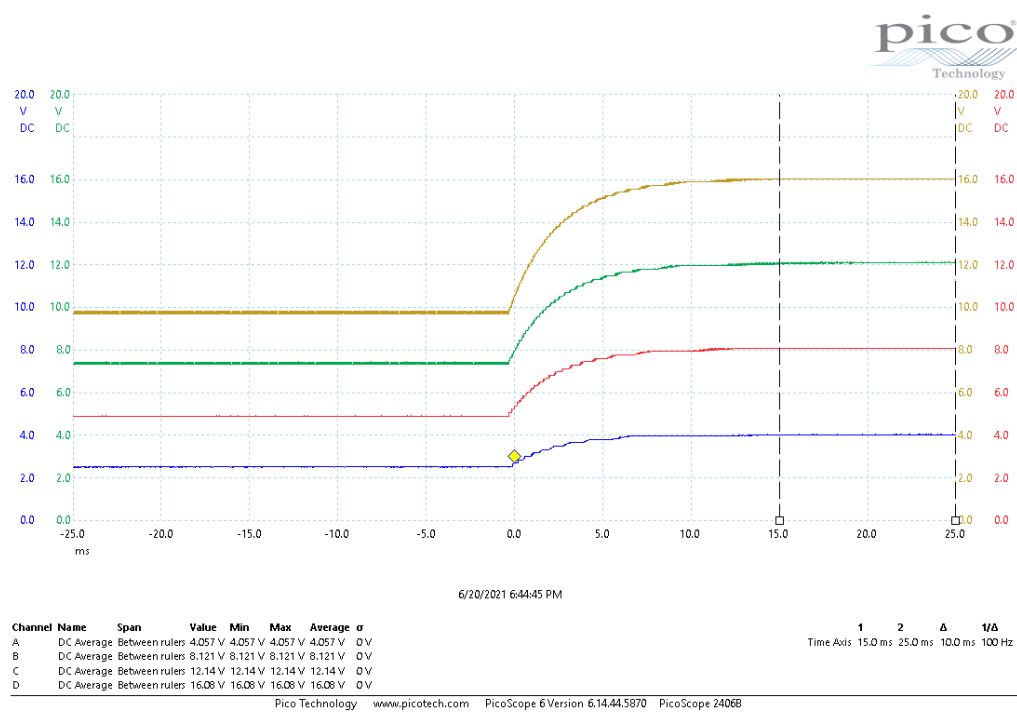


Figure 5.9: Multiple cell voltage output test.

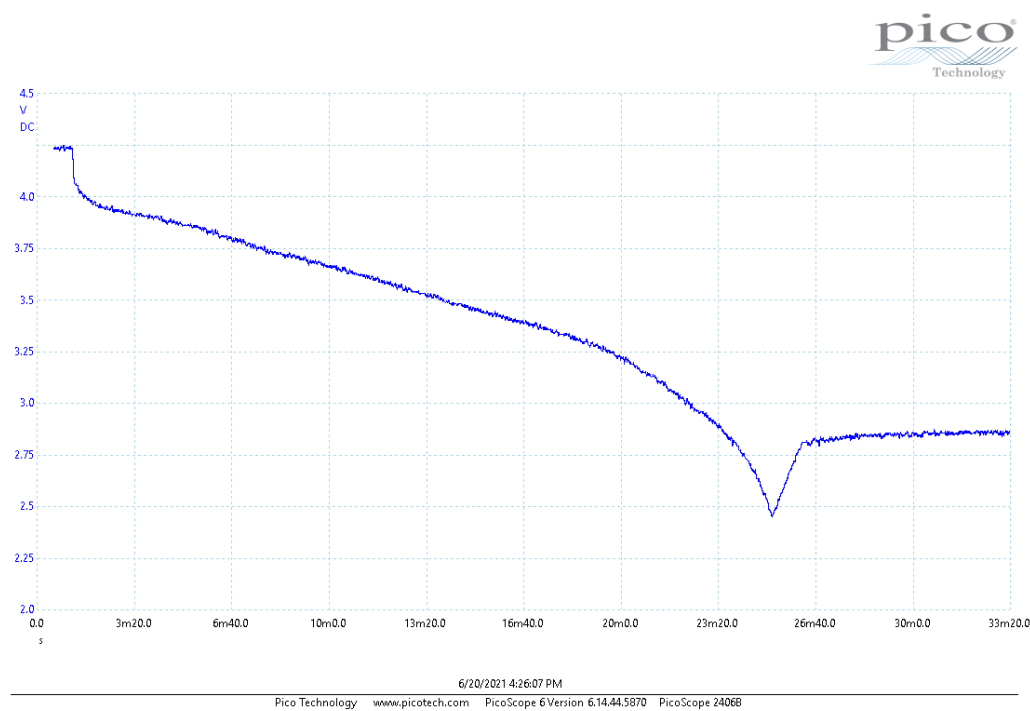


Figure 5.10: Discharge cycle voltage emulated on a HW setup cell.

5.2 Cell current emulation

The circuit used to generate the differential voltage required to emulate the voltage drop at a shunt resistor in series with a battery pack, for current measurement of the BMS, is presented in Figure 5.11.

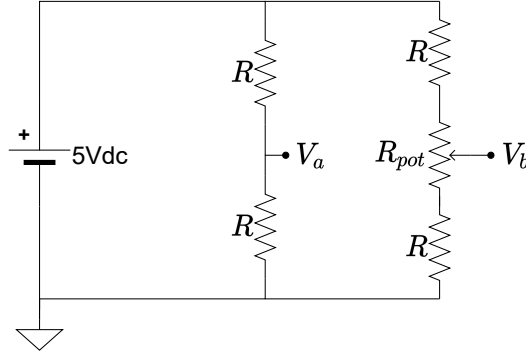


Figure 5.11: Differential voltage generation.

In this circuit, varying the position of the potentiometer allows control of the resulting voltage between points V_b and V_a . This allows the generation of positive and negative voltages between those two points. The voltage V_{b-a} can be given as:

$$V_{b-a} = V_b - V_a = \frac{5 \times (R_{pot}^{W-B} + R)}{R_{pot}^{A-B} + 2R} V - 2.5V \quad (5.8)$$

Where R_{pot}^{W-B} is the resistance from the wiper to the base of the potentiometer, and R_{pot}^{A-B} is the resistance from the top to the base of the potentiometer. To make the circuit controllable by a MCU, a digital potentiometer with SPI interface is used.

Depending on the current value desired for the BMS to measure and the resistance of the shunt resistor used R_{shunt} , the required V_{b-a} is given as:

$$V_{b-a} = R_{shunt} \times I_{desired} \quad (5.9)$$

In this case, to emulate the voltage drop at a shunt resistor connected to a battery pack, the point V_a is connected to the base of the voltage emulation cells, generating the differential V_{b-a} relative to this point for the BMS to measure. Therefore, isolation between the MCU and this section is also required.

The final schematic for the pcb design is presented in Figure 5.12. The TEL_5-1211 is a 12V-5V isolated converter, used also in the voltage emulation pcb in the previous section. The MCP4261-502EP is the digital potentiometer used, with maximum resistance between point A

and B of 5kΩ. The SI8662AB-B-IS1 component is a digital isolator used for isolating the SPI bus between the MCU and the digital potentiometer.

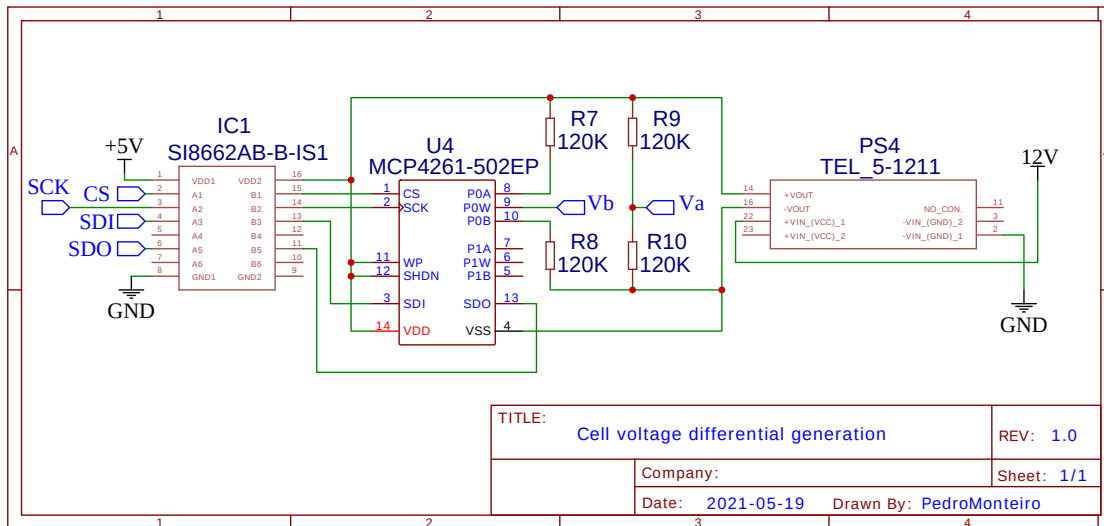


Figure 5.12: Schematic for differential voltage generation pcb.

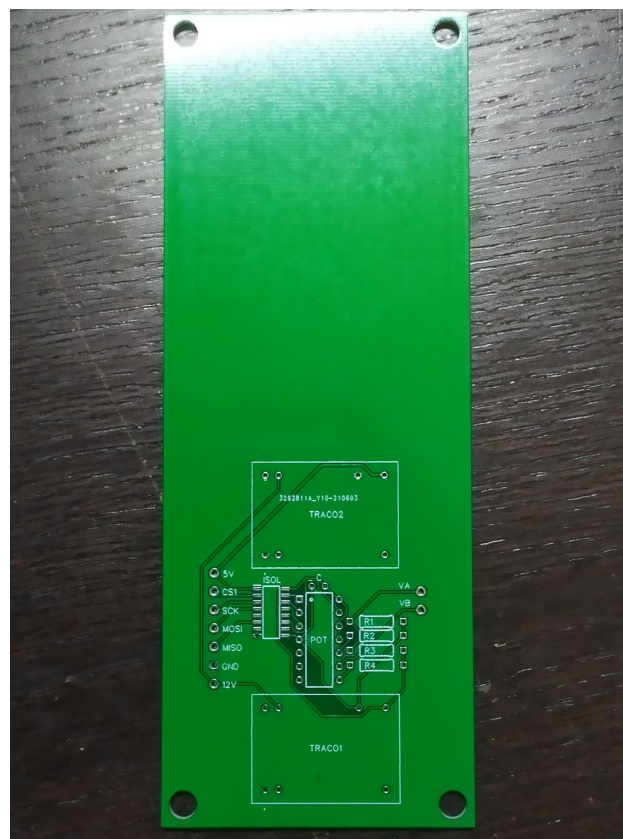


Figure 5.13: PCB for differential voltage generation.

The resulting pcb is displayed in Figure 5.13. The empty space on the pcb was added so that the pcb would be of the same size as the voltage emulation pcbs, so that they could be assembled on top of each other. An additional TEL_5-1211 was added to generate the 5V power supply for the voltage emulation pcbs. Unfortunately, the SI8662AB-B-IS1 digital isolator components did not arrive in time, therefore the pcb was not assembled.

5.2.1 Experimental results

The test performed involved setting the differential voltage from 0V to -20mV. The differential voltage was measured using two channels, channel A connected to V_b , and channel B connected to V_a . The subtraction A-B is then presented on screen for visualization of the differential voltage V_{b-a} . The results achieved are presented in Figure 5.14.

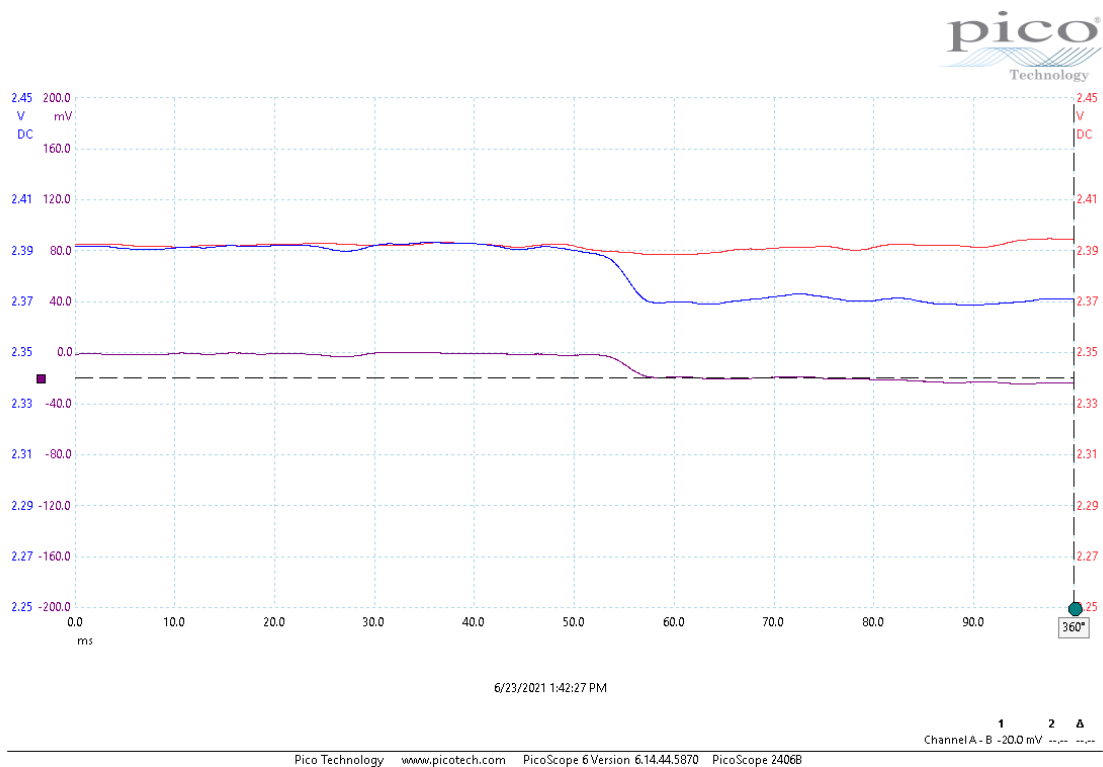


Figure 5.14: Differential voltage generation results.

The blue line corresponds to channel A reading, the red line to the channel B reading, and the resulting purple line corresponds to the difference between channel A and B, the differential voltage V_{b-a} . As can be seen, the voltage V_{b-a} drops from being near 0V to approximately -20mV at the marker set, in under 10ms. However, many offset errors were verified and had to be corrected while using this type of circuit for differential voltage generation, due to the inherent inaccuracy in resistance values from the resistors used and the digital potentiometer used, making

the method not suitable for the required small mV variations required. For future implementation, a more precise method for generating voltage differential should be used.

Chapter 6

Final Remarks and future work

The main contribution presented in this thesis was the development and study of simple and computationally efficient data-driven methods for SOC, SOH, and cell model online parameter estimation for lithium-ion batteries. Another important contribution, was the development of a HW setup capable of emulating the voltage and current behaviour of a lithium-ion 48V battery pack.

For the State-of-Charge estimation, a study was made using an online publicly available dataset [105], where it was demonstrated how FNNs with low number of parameters are more than capable of accurately estimating SOC, when tested with multiple dynamic drive cycles at different temperatures. The method presented in this paper was capable of achieving these results due to the efficient method for online input preprocessing implemented, which added a memory component to the FNN model, greatly improving its accuracy in SOC estimation. To further validate the method presented, the model was trained and validated with experimental data collected by Continental Engineering Solutions. The data made available consisted of multiple constant current charge and discharge cycles applied to a lithium-ion cell. In this case, despite that during the training procedure the model was not presented to currents above 10A, during validation the model was capable of accurately predicting SOC for higher current discharges at 15A, 20A and 25A, showcasing the model's generalization capability and capacity in modeling the battery's behaviour, while being trained with limited data.

For the State-of-Health estimation, a study was made using a similar model to the one used for SOC estimation, with the same method for input preprocessing applied, albeit with different inputs. The purpose was to check whether the same type of inputs used for SOC estimation could also be used for battery SOH estimation, when analysing the voltage data during the charging procedure. Data used in this study was taken from a publicly available dataset [104], where multiple charging and discharging profiles were applied to 4 cells, for battery degradation study purposes. In this case, mixed results were achieved, where for some cells the method was capable of accurately representing SOH, while for others bad results were achieved. In general, further changes and improvements must be made in the model, since in its current state it is not viable for SOH estimation.

For the battery model online parameter estimation, a study was made using an online RLS-FF

algorithm, to check its viability for estimating the battery's parameters when presented with data from dynamic drive cycles at multiple temperatures. Data used was taken from the same dataset as the one used for SOC estimation [105]. The algorithm, when presented with data from a UDDS cycle at a temperature of 25°C, was capable of very accurately predicting the cell's voltage, with a maximum verified error of 31.122mV. However, when presented with data from UDDS at lower temperatures or data from the much more dynamic cycle US06, a significantly higher error is verified. This showcases the algorithm's inability to accurately predict the model's parameters for more dynamic cycles at lower temperatures. Generally at ambient temperatures 25°C and 10°C, the battery model was capable of accurately predicting the cell's voltage.

For the HW setup designed, a setup capable of emulating the cell voltage and current behaviour of 12 cells connected in series, to form a 48V battery pack, was designed and validated. This type of setup can prove very useful for BMS debugging and practical testing, since voltage and current data collected from any cell can be emulated in the setup and connected to the BMS, enabling a real practical test of the BMS without requiring a physical battery pack present. For the cell voltage generation, the setup was capable of producing the desired voltage value for each cell, in a fast and steady manner. As a demonstration of its capabilities, the voltage observed from a lithium-ion cell during an entire discharge cycle was emulated. For the battery current generation, a simple setup was designed to emulate the differential voltage drop of a shunt resistor connected in series with the battery pack, for current measurement of the BMS. Results demonstrating its capability are presented. However, due to the inherent inaccuracy of the resistors and components used, the current generation setup is not accurate enough to emulate the mV variations observed in a real shunt resistor connected to a battery pack. Further improvements must be made in order to accurately emulate the current behaviour of a battery pack.

6.1 Future work

In this section, a list of possible improvements to the work made in this thesis are suggested:

SOC estimation using SOH as input - The inclusion of the current cell's SOH as input for SOC estimation could make the model more capable of accurately predicting the SOC of a cell as it deteriorates. However, for the application of this concept, the battery data used in training and validation must include drive cycles taken at multiple stages of battery deterioration (decreasing SOH).

Accuracy comparison between multiple methods for online cell parameter estimation - In this thesis, only the accuracy achieved by the RLS-FF algorithm for online cell parameter estimation was analysed. For further validation, a comparison on the accuracy achieved in voltage prediction could be made between the RLS-FF algorithm used in this paper and other common methods used, such as the Kalman Filter based methods.

HW setup improvement and further validation - In its current state, the method for current generation is not accurate enough. Therefore, for further use of the HW setup a new setup should be implemented for current generation. Afterwards, it could be interesting to connect the setup to a BMS, and emulate a full drive cycle, to record the SOC prediction results estimated by the BMS.

More dynamic experimental data for SOC estimation - In this thesis, the data used to experimentally validate the SOC estimation model consisted only of constant current charge and discharge profiles. For further experimental validation, the model should be trained and validated using much more dynamic drive cycle data.

SOH model further improvement and validation - The SOH model presented in this thesis was capable of accurately predicting SOH for some cells, while remaining inaccurate for other cells. Training the model with data from more cells could make the model capable of accurately predicting SOH for all cells tested. Testing of different inputs could also help improve results achieved.

References

- [1] Texas Instruments. *16-Cell EV/HEV Integrated Battery Monitor and Protector*, 11 2016. URL: <https://www.ti.com/lit/ds/symlink/bq76pl455a-q1.pdf>.
- [2] Texas Instruments. *3-to-6 Series Cell Lithium-Ion Battery Monitor and Secondary Protection IC for EV and HEV Applications*, 12 2016. URL: <https://www.ti.com/lit/ds/symlink/bq76pl536a-q1.pdf>.
- [3] Linear Technology. *Multicell Battery Monitors*, 2014. URL: <https://www.ti.com/lit/ds/symlink/bq76pl455a-q1.pdf>.
- [4] M. Lelie, T. Braun, M. Knips, H. Nordmann, F. Ringbeck, H. Zappen, and D.U. Sauer. Battery management system hardware concepts: An overview. *Applied Sciences (Switzerland)*, 8(4), 2018. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85044679275&doi=10.3390%2fapp8040534&partnerID=40&md5=2b99fbaac632bd06c6b9c232eed365c0>, doi:10.3390/app8040534.
- [5] M.A. Hannan, M.M. Hoque, A. Hussain, Y. Yusof, and P.J. Ker. State-of-the-art and energy management system of lithium-ion batteries in electric vehicle applications: Issues and recommendations. *IEEE Access*, 6:19362–19378, 2018. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85044308026&doi=10.1109%2fACCESS.2018.2817655&partnerID=40&md5=4e1518de522741d6aeb183d6c1be676d>, doi:10.1109/ACCESS.2018.2817655.
- [6] E. Chemali, M. Preindl, P. Malysz, and A. Emadi. Electrochemical and electrostatic energy storage and management systems for electric drive vehicles: State-of-the-art review and future trends. *IEEE Journal of Emerging and Selected Topics in Power Electronics*, 4(3):1117–1134, 2016. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-84980512047&doi=10.1109%2fJESTPE.2016.2566583&partnerID=40&md5=39d500e6635a8bbe1b7b51efba07c5a6>, doi:10.1109/JESTPE.2016.2566583.
- [7] S. Steinhorst, Z. Shao, S. Chakraborty, M. Kauer, S. Li, M. Lukasiewicz, S. Narayanaswamy, M.U. Rafique, and Q. Wang. Distributed reconfigurable battery system management architectures. In *Proceedings of the Asia and South Pacific Design Automation Conference, ASP-DAC*, volume 25-28-January-2016, pages 429–434, 2016. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-84996605382&doi=10.1109%2fASPDAC.2016.7428049&partnerID=40&md5=665d6038ec26878e241a9af32c7da5c6>, doi:10.1109/ASPDAC.2016.7428049.

- [8] H. Rahimi-Eichi, U. Ojha, F. Baronti, and M. Chow. Battery management system: An overview of its application in the smart grid and electric vehicles. *IEEE Industrial Electronics Magazine*, 7(2):4–16, 2013. doi:[10.1109/MIE.2013.2250351](https://doi.org/10.1109/MIE.2013.2250351).
- [9] L. Lu, X. Han, J. Li, J. Hua, and M. Ouyang. A review on the key issues for lithium-ion battery management in electric vehicles. *Journal of Power Sources*, 226:272–288, 2013. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-84869862215&doi=10.1016%2fj.jpowsour.2012.10.060&partnerID=40&md5=8f074486b4b47223653f0e7ce56eeea3>, doi:[10.1016/j.jpowsour.2012.10.060](https://doi.org/10.1016/j.jpowsour.2012.10.060).
- [10] A. Farmann and D.U. Sauer. A comprehensive review of on-board state-of-available-power prediction techniques for lithium-ion batteries in electric vehicles. *Journal of Power Sources*, 329:123–137, 2016. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-84983394282&doi=10.1016%2fj.jpowsour.2016.08.031&partnerID=40&md5=265bac8182b6fd8011a2306c4d437155>, doi:[10.1016/j.jpowsour.2016.08.031](https://doi.org/10.1016/j.jpowsour.2016.08.031).
- [11] S. Barcellona and L. Piegari. Lithium ion battery models and parameter identification techniques. *Energies*, 10(12), 2017. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85044463245&doi=10.3390%2fen10122007&partnerID=40&md5=af2ca77891a5ca698d089b1f0b1a9ff0>, doi:[10.3390/en10122007](https://doi.org/10.3390/en10122007).
- [12] Z. B. Omariba, L. Zhang, and D. Sun. Review of battery cell balancing methodologies for optimizing battery pack performance in electric vehicles. *IEEE Access*, 7:129335–129352, 2019. doi:[10.1109/ACCESS.2019.2940090](https://doi.org/10.1109/ACCESS.2019.2940090).
- [13] U.K. Das, P. Shrivastava, K.S. Tey, M.Y.I. Bin Idris, S. Mekhilef, E. Jamei, M. Seyedmahmoudian, and A. Stojcevski. Advancement of lithium-ion battery cells voltage equalization techniques: A review. *Renewable and Sustainable Energy Reviews*, 134, 2020. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85090402533&doi=10.1016%2fj.rser.2020.110227&partnerID=40&md5=af562f7decd9c8441dad5b34f3743cbd>, doi:[10.1016/j.rser.2020.110227](https://doi.org/10.1016/j.rser.2020.110227).
- [14] J. Gallardo-Lozano, E. Romero-Cadaval, M.I. Milanés-Montero, and M.A. Guerrero-Martinez. Battery equalization active methods. *Journal of Power Sources*, 246:934–949, 2014. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-84884646980&doi=10.1016%2fj.jpowsour.2013.08.026&partnerID=40&md5=9eeb93a1c10950cfc173d210ed1363e2>, doi:[10.1016/j.jpowsour.2013.08.026](https://doi.org/10.1016/j.jpowsour.2013.08.026).
- [15] J. Meng, G. Luo, M. Ricco, M. Swierczynski, D.-I. Stroe, and R. Teodorescu. Overview of lithium-ion battery modeling methods for state-of-charge estimation in electrical vehicles. *Applied Sciences (Switzerland)*, 8(5), 2018. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85046147834&doi=10.3390%2fapp8050659&partnerID=40&md5=e79a72e776dd522afd55c849e8a23054>, doi:[10.3390/app8050659](https://doi.org/10.3390/app8050659).
- [16] P. Shrivastava, T.K. Soon, M.Y.I.B. Idris, and S. Mekhilef. Overview of model-based online state-of-charge estimation using kalman filter family for lithium-ion batteries. *Renewable*

- and Sustainable Energy Reviews*, 113, 2019. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85067867691&doi=10.1016%2fj.rser.2019.06.040&partnerID=40&md5=be10046a935cdc7f65dd9262910dd1c3>, doi:10.1016/j.rser.2019.06.040.
- [17] A. Fotouhi, D.J. Auger, K. Propp, S. Longo, and M. Wild. A review on electric vehicle battery modelling: From lithium-ion toward lithium-sulphur. *Renewable and Sustainable Energy Reviews*, 56:1008–1021, 2016. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-84955498118&doi=10.1016%2fj.rser.2015.12.009&partnerID=40&md5=1f08e75e36b0495d85960852e997a953>, doi:10.1016/j.rser.2015.12.009.
- [18] S. Nejad, D.T. Gladwin, and D.A. Stone. A systematic review of lumped-parameter equivalent circuit models for real-time estimation of lithium-ion battery states. *Journal of Power Sources*, 316:183–196, 2016. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-84962026513&doi=10.1016%2fj.jpowsour.2016.03.042&partnerID=40&md5=c73469da8927ff8ce2766147842fee32>, doi:10.1016/j.jpowsour.2016.03.042.
- [19] Y. Wang, J. Tian, Z. Sun, L. Wang, R. Xu, M. Li, and Z. Chen. A comprehensive review of battery modeling and state estimation approaches for advanced battery management systems. *Renewable and Sustainable Energy Reviews*, 131, 2020. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85087996211&doi=10.1016%2fj.rser.2020.110015&partnerID=40&md5=44cdb7148557bdf72f3116bbc4712cc2>, doi:10.1016/j.rser.2020.110015.
- [20] H. He, R. Xiong, and J. Fan. Evaluation of lithium-ion battery equivalent circuit models for state of charge estimation by an experimental approach. *Energies*, 4(4):582–598, 2011. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-79955655304&doi=10.3390%2fen4040582&partnerID=40&md5=e68cd76bc073dc3b699f4beb0bddb11f>, doi:10.3390/en4040582.
- [21] H. He, R. Xiong, H. Guo, and S. Li. Comparison study on the battery models used for the energy management of batteries in electric vehicles. *Energy Conversion and Management*, 64:113–121, 2012. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-84866489445&doi=10.1016%2fj.enconman.2012.04.014&partnerID=40&md5=2a7780c46124855204112f8c631c71e9>, doi:10.1016/j.enconman.2012.04.014.
- [22] X. Hu, S. Li, and H. Peng. A comparative study of equivalent circuit models for li-ion batteries. *Journal of Power Sources*, 198:359–367, 2012. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-80755140658&doi=10.1016%2fj.jpowsour.2011.10.013&partnerID=40&md5=b6c20a36687f2873e992958e457f4c30>, doi:10.1016/j.jpowsour.2011.10.013.
- [23] U. Westerhoff, K. Kurbach, F. Lienesch, and M. Kurrat. Analysis of lithium-ion battery models based on electrochemical impedance spectroscopy. *Energy Technology*, 4(12):1620–1630, 2016. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85006220538&doi=10.1002%2fente>.

- 201600154&partnerID=40&md5=54b2d72a289d8463893c64ba803a22cb,
doi:10.1002/ente.201600154.
- [24] C. Vidal, P. Malysz, P. Kollmeyer, and A. Emadi. Machine learning applied to electrified vehicle battery state of charge and state of health estimation: State-of-the-art. *IEEE Access*, 8:52796–52814, 2020. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85082517342&doi=10.1109%2fACCESS.2020.2980961&partnerID=40&md5=9ea2adbfcd9e004f81c7cf4fd85d085a,doi:10.1109/ACCESS.2020.2980961>.
- [25] M.A. Hannan, M.S.H. Lipu, A. Hussain, and A. Mohamed. A review of lithium-ion battery state of charge estimation and management system in electric vehicle applications: Challenges and recommendations. *Renewable and Sustainable Energy Reviews*, 78:834–854, 2017. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85019063470&doi=10.1016%2fj.rser.2017.05.001&partnerID=40&md5=7c6224228d436672eb55b7510c39874c,doi:10.1016/j.rser.2017.05.001>.
- [26] R. Zhang, B. Xia, B. Li, L. Cao, Y. Lai, W. Zheng, H. Wang, and W. Wang. State of the art of lithium-ion battery soc estimation for electrical vehicles. *Energies*, 11(7), 2018. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85051170653&doi=10.3390%2fen11071820&partnerID=40&md5=f4f601ed4a1153ab9b78587a4fb4cf21,doi:10.3390/en11071820>.
- [27] K.S. Ng, C.-S. Moo, Y.-P. Chen, and Y.-C. Hsieh. Enhanced coulomb counting method for estimating state-of-charge and state-of-health of lithium-ion batteries. *Applied Energy*, 86(9):1506–1511, 2009. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-63449140302&doi=10.1016%2fj.apenergy.2008.11.021&partnerID=40&md5=399d415a5a21cd7a38f362005bebbbc2,doi:10.1016/j.apenergy.2008.11.021>.
- [28] Sergio Mendoza, Ji Liu, Partha Mishra, and Hosam Fathy. On the relative contributions of bias and noise to lithium-ion battery state of charge estimation errors. *Journal of Energy Storage*, 11:86–92, 06 2017. doi:10.1016/j.est.2017.01.006.
- [29] Ines Baccouche, Sabeur Jemmali, Bilal Manai, Noshin Omar, and Najoua ESSOUKRI BEN AMARA. Improved ocv model of a li-ion nmc battery for online soc estimation using the extended kalman filter. *Energies*, 10:764, 05 2017. doi:10.3390/en10060764.
- [30] Xiaosong Hu, Fengchun Sun, and Yuan Zou. Estimation of state of charge of a lithium-ion battery pack for electric vehicles using an adaptive luenberger observer. *Energies*, 3(9):1586–1603, 2010. URL: <https://www.mdpi.com/1996-1073/3/9/1586,doi:10.3390/en3091586>.
- [31] B. Wang, Z. Liu, S.E. Li, S.J. Moura, and H. Peng. State-of-charge estimation for lithium-ion batteries based on a nonlinear fractional model. *IEEE Transactions on Control Systems Technology*, 25(1):3–11, 2017. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-84966393410&doi=10.1109%2fTCST.2016.2557221&partnerID=40&md5=3ad279846a76cf3e11540e698fe6e406,doi:10.1109/TCST.2016.2557221>.

- [32] J. Xu, C.C. Mi, B. Cao, J. Deng, Z. Chen, and S. Li. The state of charge estimation of lithium-ion batteries based on a proportional-integral observer. *IEEE Transactions on Vehicular Technology*, 63(4):1614–1621, 2014. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-84903637330&doi=10.1109%2fTVT.2013.2287375&partnerID=40&md5=19386c27693728500f9374f69d0f0873,doi:10.1109/TVT.2013.2287375>.
- [33] I.-S. Kim. The novel state of charge estimation method for lithium battery using sliding mode observer. *Journal of Power Sources*, 163(1 SPEC. ISS.):584–590, 2006. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-33947581222&doi=10.1016%2fj.jpowsour.2006.09.006&partnerID=40&md5=817ad3478c66d94ba2746aa1ad39a279,doi:10.1016/j.jpowsour.2006.09.006>.
- [34] Xiaopeng Chen, Weixiang Shen, Zhenwei Cao, and Ajay Kapoor. A novel approach for state of charge estimation based on adaptive switching gain sliding mode observer in electric vehicles. *Journal of Power Sources*, 246:667 – 678, 2014. URL: <http://www.sciencedirect.com/science/article/pii/S0378775313013797,doi:https://doi.org/10.1016/j.jpowsour.2013.08.039>.
- [35] B. Xia, C. Chen, Y. Tian, W. Sun, Z. Xu, and W. Zheng. A novel method for state of charge estimation of lithium-ion batteries using a nonlinear observer. *Journal of Power Sources*, 270:359–366, 2014. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-84905863952&doi=10.1016%2fj.jpowsour.2014.07.103&partnerID=40&md5=d1ab3bc71f878577b48a156fb64ea0a2,doi:10.1016/j.jpowsour.2014.07.103>.
- [36] W. Li, L. Liang, W. Liu, and X. Wu. State of charge estimation of lithium-ion batteries using a discrete-time nonlinear observer. *IEEE Transactions on Industrial Electronics*, 64(11):8557–8565, 2017. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85036462625&doi=10.1109%2fTIE.2017.2703685&partnerID=40&md5=468630fa5cfd6103f611ee3a82dfcf06,doi:10.1109/TIE.2017.2703685>.
- [37] Gregory L. Plett. Extended kalman filtering for battery management systems of lipb-based hev battery packs: Part 1. background. *Journal of Power Sources*, 134(2):252 – 261, 2004. URL: <http://www.sciencedirect.com/science/article/pii/S0378775304003593,doi:https://doi.org/10.1016/j.jpowsour.2004.02.031>.
- [38] M. Urbain, S. Raël, B. Davat, and P. Desprez. State estimation of a lithium-ion battery through kalman filter. In *PESC Record - IEEE Annual Power Electronics Specialists Conference*, pages 2804–2810, 2007. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-48349146509&doi=10.1109%2fPESC.2007.4342463&partnerID=40&md5=7b68014ede2ad3a7fc3dcbb50c55058b,doi:10.1109/PESC.2007.4342463>.
- [39] M. W. Yatsui and H. Bai. Kalman filter based state-of-charge estimation for lithium-ion batteries in hybrid electric vehicles using pulse charging. In *2011 IEEE Vehicle Power and Propulsion Conference*, pages 1–5, 2011. doi:10.1109/VPPC.2011.6042988.

- [40] M. Mastali, J. Vazquez-Arenas, R. Fraser, M. Fowler, S. Afshar, and M. Stevens. Battery state of the charge estimation using kalman filtering. *Journal of Power Sources*, 239:294 – 307, 2013. URL: <http://www.sciencedirect.com/science/article/pii/S0378775313005259>, doi:<https://doi.org/10.1016/j.jpowsour.2013.03.131>.
- [41] Z. Zhu, J. Sun, and D. Liu. Online state of charge ekf estimation for lifepo4 battery management systems. In *2012 International Symposium on Intelligent Signal Processing and Communications Systems*, pages 609–614, 2012. doi:[10.1109/ISPACS.2012.6473562](https://doi.org/10.1109/ISPACS.2012.6473562).
- [42] Rui Xiong, Xianzhi Gong, Chunting Chris Mi, and Fengchun Sun. A robust state-of-charge estimator for multiple types of lithium-ion batteries using adaptive extended kalman filter. *Journal of Power Sources*, 243:805 – 816, 2013. URL: <http://www.sciencedirect.com/science/article/pii/S0378775313010835>, doi:<https://doi.org/10.1016/j.jpowsour.2013.06.076>.
- [43] R. Xiong, H. He, F. Sun, and K. Zhao. Evaluation on state of charge estimation of batteries with adaptive extended kalman filter by experiment approach. *IEEE Transactions on Vehicular Technology*, 62(1):108–117, 2013. doi:[10.1109/TVT.2012.2222684](https://doi.org/10.1109/TVT.2012.2222684).
- [44] Wei He, Nicholas Williard, Chaochao Chen, and Michael Pecht. State of charge estimation for electric vehicle batteries using unscented kalman filtering. *Microelectronics Reliability*, 53(6):840 – 847, 2013. URL: <http://www.sciencedirect.com/science/article/pii/S0026271412005185>, doi:<https://doi.org/10.1016/j.microrel.2012.11.010>.
- [45] Jiani Du, Zhitao Liu, and Youyi Wang. State of charge estimation for li-ion battery based on model from extreme learning machine. *Control Engineering Practice*, 26:11 – 19, 2014. URL: <http://www.sciencedirect.com/science/article/pii/S0967066113002530>, doi:<https://doi.org/10.1016/j.conengprac.2013.12.014>.
- [46] Rudolph Merwe, Arnaud Doucet, Nando Freitas, and Eric Wan. The unscented particle filter. *NIPS*, 13, 01 2001.
- [47] M. Gao, Y. Liu, and Z. He. Battery state of charge online estimation based on particle filter. In *2011 4th International Congress on Image and Signal Processing*, volume 4, pages 2233–2236, 2011. doi:[10.1109/CISP.2011.6100603](https://doi.org/10.1109/CISP.2011.6100603).
- [48] Bizhong Xia, Zhen Sun, Ruifeng Zhang, and Zizhou Lao. A cubature particle filter algorithm to estimate the state of the charge of lithium-ion batteries based on a second-order equivalent circuit model. *Energies*, 10(4), 2017. URL: <https://www.mdpi.com/1996-1073/10/4/457>, doi:[10.3390/en10040457](https://doi.org/10.3390/en10040457).
- [49] Yao He, XingTao Liu, ChenBin Zhang, and ZongHai Chen. A new model for state-of-charge (soc) estimation for high-power li-ion batteries. *Applied Energy*, 101:808 – 814, 2013. Sustainable Development of Energy, Water and Environment Systems. URL: <http://www.sciencedirect.com/science/article/pii/S0306261912006058>, doi:<https://doi.org/10.1016/j.apenergy.2012.08.031>.
- [50] Mohammad Charkhgard and Mohammad Haddad Zarif. Design of adaptive h filter for implementing on state-of-charge estimation based on battery

- state-of-charge-varying modelling. *IET Power Electronics*, 8(10):1825–1833, 2015. URL: <https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/iet-pel.2014.0523>, arXiv:<https://ietresearch.onlinelibrary.wiley.com/doi/pdf/10.1049/iet-pel.2014.0523>, doi:<https://doi.org/10.1049/iet-pel.2014.0523>.
- [51] R. Xiong, Q. Yu, L.Y. Wang, and C. Lin. A novel method to obtain the open circuit voltage for the state of charge of lithium ion batteries in electric vehicles by using h infinity filter. *Applied Energy*, 207:346–353, 2017. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85019732146&doi=10.1016%2fj.apenergy.2017.05.136&partnerID=40&md5=6fffd591e136ddf944cd736da633df5c3>, doi:10.1016/j.apenergy.2017.05.136.
- [52] M.S. Hossain Lipu, M.A. Hannan, A. Hussain, A. Ayob, M.H.M. Saad, T.F. Karim, and D.N.T. How. Data-driven state of charge estimation of lithium-ion batteries: Algorithms, implementation factors, limitations and future trends. *Journal of Cleaner Production*, 277, 2020. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85091229921&doi=10.1016%2fj.jclepro.2020.124110&partnerID=40&md5=7a8872cfddb2c7ef86d5c0611963d9c1>, doi:10.1016/j.jclepro.2020.124110.
- [53] E. Chemali, P.J. Kollmeyer, M. Preindl, and A. Emadi. State-of-charge estimation of li-ion batteries using deep neural networks: A machine learning approach. *Journal of Power Sources*, 400:242–255, 2018. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85051653123&doi=10.1016%2fj.jpowsour.2018.06.104&partnerID=40&md5=4a459ef11d5c013a66deb570bfa9bdaf>, doi:10.1016/j.jpowsour.2018.06.104.
- [54] C. Vidal, M. Hausmann, D. Barroso, P.M. Shamsabadi, A. Biswas, E. Chemali, R. Ahmed, and A. Emadi. Hybrid energy storage system state-of-charge estimation using artificial neural network for micro-hybrid applications. In *2018 IEEE Transportation and Electrification Conference and Expo, ITEC 2018*, pages 868–873, 2018. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85053818087&doi=10.1109%2fITEC.2018.8450251&partnerID=40&md5=6c5b420fa6d6dfc9d3608db3cbb9c251>, doi:10.1109/ITEC.2018.8450251.
- [55] X. Dang, L. Yan, H. Jiang, X. Wu, and H. Sun. Open-circuit voltage-based state of charge estimation of lithium-ion power battery by combining controlled auto-regressive and moving average modeling with feedforward-feedback compensation method. *International Journal of Electrical Power and Energy Systems*, 90:27–36, 2017. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85011906831&doi=10.1016%2fj.ijepes.2017.01.013&partnerID=40&md5=4a1d89f5964e039e5d920dbc7730e040>, doi:10.1016/j.ijepes.2017.01.013.
- [56] W. He, N. Williard, C. Chen, and M. Pecht. State of charge estimation for li-ion batteries using neural network modeling and unscented kalman filter-based error cancellation. *International Journal of Electrical Power and Energy Systems*, 62:783–791, 2014. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-84903199498&doi=10.1016%2fj.ijepes>.

- 2014.04.059&partnerID=40&md5=906caf409766d47e0eba3a0d7191010e, doi:10.1016/j.ijepes.2014.04.059.
- [57] M.S. Hossain Lipu, M.A. Hannan, A. Hussain, M.H. Saad, A. Ayob, and M.N. Uddin. Extreme learning machine model for state-of-charge estimation of lithium-ion battery using gravitational search algorithm. *IEEE Transactions on Industry Applications*, 55(4):4225–4234, 2019. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85067849198&doi=10.1109%2fTIA.2019.2902532&partnerID=40&md5=1b1eceae17d81276b2de47e9fe3601b>, doi:10.1109/TIA.2019.2902532.
- [58] F. Sun, R. Xiong, and H. He. A systematic state-of-charge estimation framework for multi-cell battery pack in electric vehicles using bias correction technique. *Applied Energy*, 162:1399–1409, 2016. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-84923352487&doi=10.1016%2fj.apenergy.2014.12.021&partnerID=40&md5=0a624dfc0a1875b3ffd7cbf2a6fec160>, doi:10.1016/j.apenergy.2014.12.021.
- [59] A. Alfi, M. Charkhgard, and M.H. Zarif. Hybrid state of charge estimation for lithium-ion batteries: Design and implementation. *IET Power Electronics*, 7(11):2758–2764, 2014. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-84909586237&doi=10.1049%2fiet-pel.2013.0746&partnerID=40&md5=d7ae26d11c3e5b9ecdc150e89b5fa2d2>, doi:10.1049/iet-pel.2013.0746.
- [60] J.C. Alvarez Anton, P.J. Garcia Nieto, C. Blanco Viejo, and J.A. Vilan Vilan. Support vector machines used to estimate the battery state of charge. *IEEE Transactions on Power Electronics*, 28(12):5919–5926, 2013. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-84879110729&doi=10.1109%2fTPEL.2013.2243918&partnerID=40&md5=bf7fde489e7503b594e38b64b9cdded0>, doi:10.1109/TPEL.2013.2243918.
- [61] H. Chaoui and C.C. Ibe-Ekeocha. State of charge and state of health estimation for lithium batteries using recurrent neural networks. *IEEE Transactions on Vehicular Technology*, 66(10):8773–8783, 2017. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85021794128&doi=10.1109%2fTVT.2017.2715333&partnerID=40&md5=b251a73f4c93385f0f817dbba1d8b193>, doi:10.1109/TVT.2017.2715333.
- [62] M.S.H. Lipu, M.A. Hannan, A. Hussain, M.H.M. Saad, A. Ayob, and F. Blaabjerg. State of charge estimation for lithium-ion battery using recurrent narx neural network model based lighting search algorithm. *IEEE Access*, 6:28150–28161, 2018. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85046997280&doi=10.1109%2fACCESS.2018.2837156&partnerID=40&md5=6767fb7d7cf8552eb5db7bb1ff5429bb>, doi:10.1109/ACCESS.2018.2837156.
- [63] E. Chemali, P.J. Kollmeyer, M. Preindl, R. Ahmed, and A. Emadi. Long short-term memory networks for accurate state-of-charge estimation of li-ion batteries. *IEEE Transactions on Industrial Electronics*, 65(8):6730–6739, 2018. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85040029828&doi=10.1109%2fTIE.2017.2787586&partnerID=40&md5=fd068f87249a14408b74a08c83c5b5f3>, doi:10.1109/TIE.2017.2787586.

- [64] F. Yang, X. Song, F. Xu, and K.-L. Tsui. State-of-charge estimation of lithium-ion batteries via long short-term memory network. *IEEE Access*, 7:53792–53799, 2019. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85065417270&doi=10.1109%2fACCESS.2019.2912803&partnerID=40&md5=731a7ff9e54de329e5b568a33ac3cba1>, doi:10.1109/ACCESS.2019.2912803.
- [65] C. Bian, H. He, and S. Yang. Stacked bidirectional long short-term memory networks for state-of-charge estimation of lithium-ion batteries. *Energy*, 191, 2020. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85075852125&doi=10.1016%2fj.energy.2019.116538&partnerID=40&md5=1abc139f72410f03b07f5d3581c552f8>, doi:10.1016/j.energy.2019.116538.
- [66] F. Yang, W. Li, C. Li, and Q. Miao. State-of-charge estimation of lithium-ion batteries based on gated recurrent neural network. *Energy*, 175:66–75, 2019. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85063348865&doi=10.1016%2fj.energy.2019.03.059&partnerID=40&md5=6ae073cc039b6324133ff650e8aef683>, doi:10.1016/j.energy.2019.03.059.
- [67] C. Li, F. Xiao, and Y. Fan. An approach to state of charge estimation of lithium-ion batteries based on recurrent neural networks with gated recurrent unit. *Energies*, 12(9), 2019. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85066276137&doi=10.3390%2fen12091592&partnerID=40&md5=6aede2e5e8c271984749576b023a112b>, doi:10.3390/en12091592.
- [68] R. Xiong, L. Li, and J. Tian. Towards a smarter battery management system: A critical review on battery state of health monitoring methods. *Journal of Power Sources*, 405:18–29, 2018. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85054653853&doi=10.1016%2fj.jpowsour.2018.10.019&partnerID=40&md5=f68b2a8535972d120c2b91625d5b6968>, doi:10.1016/j.jpowsour.2018.10.019.
- [69] M.S.H. Lipu, M.A. Hannan, A. Hussain, M.M. Hoque, P.J. Ker, M.H.M. Saad, and A. Ayob. A review of state of health and remaining useful life estimation methods for lithium-ion battery in electric vehicles: Challenges and recommendations. *Journal of Cleaner Production*, 205:115–133, 2018. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85054827445&doi=10.1016%2fj.jclepro.2018.09.065&partnerID=40&md5=e98283595bf7fe3b234e6478136b974c>, doi:10.1016/j.jclepro.2018.09.065.
- [70] M. Bercibar, I. Gandiaga, I. Villarreal, N. Omar, J. Van Mierlo, and P. Van Den Bossche. Critical review of state of health estimation methods of lithium-ion batteries for real applications. *Renewable and Sustainable Energy Reviews*, 56:572–587, 2016. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-84950108579&doi=10.1016%2fj.rser.2015.11.042&partnerID=40&md5=a6ed215017c050e966cd7b197c183875>, doi:10.1016/j.rser.2015.11.042.

- [71] Matteo Galeotti, Lucio Cinà, Corrado Giammanco, Stefano Cordiner, and Aldo Di Carlo. Performance analysis and soh (state of health) evaluation of lithium polymer batteries through electrochemical impedance spectroscopy. *Energy*, 89:678 – 686, 2015. URL: <http://www.sciencedirect.com/science/article/pii/S0360544215007756>, doi:<https://doi.org/10.1016/j.energy.2015.05.148>.
- [72] Akram Eddahech, Olivier Briat, and Jean-Michel Vinassa. Determination of lithium-ion battery state-of-health based on constant-voltage charge phase. *Journal of Power Sources*, 258:218 – 227, 2014. URL: <http://www.sciencedirect.com/science/article/pii/S0378775314002031>, doi:<https://doi.org/10.1016/j.jpowsour.2014.02.020>.
- [73] J. Tian, R. Xiong, and Q. Yu. Fractional-order model-based incremental capacity analysis for degradation state recognition of lithium-ion batteries. *IEEE Transactions on Industrial Electronics*, 66(2):1576–1584, 2019. doi:[10.1109/TIE.2018.2798606](https://doi.org/10.1109/TIE.2018.2798606).
- [74] 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. *Applied Energy*, 168:465 – 472, 2016. URL: <http://www.sciencedirect.com/science/article/pii/S0306261916301131>, doi:<https://doi.org/10.1016/j.apenergy.2016.01.125>.
- [75] Seongjun Lee, Jonghoon Kim, Jaemoon Lee, and B.H. Cho. State-of-charge and capacity estimation of lithium-ion battery using a new open-circuit voltage versus state-of-charge. *Journal of Power Sources*, 185(2):1367 – 1373, 2008. URL: <http://www.sciencedirect.com/science/article/pii/S0378775308017965>, doi:<https://doi.org/10.1016/j.jpowsour.2008.08.103>.
- [76] F. Zhang, G. Liu, and L. Fang. Battery state estimation using unscented kalman filter. In *2009 IEEE International Conference on Robotics and Automation*, pages 1863–1868, 2009. doi:[10.1109/ROBOT.2009.5152745](https://doi.org/10.1109/ROBOT.2009.5152745).
- [77] Simon Schwunk, Nils Armbruster, Sebastian Straub, Johannes Kehl, and Matthias Vetter. Particle filter for state of charge and state of health estimation for lithium-iron phosphate batteries. *Journal of Power Sources*, 239:705 – 710, 2013. URL: <http://www.sciencedirect.com/science/article/pii/S037877531201614X>, doi:<https://doi.org/10.1016/j.jpowsour.2012.10.058>.
- [78] G.-W. You, S. Park, and D. Oh. Diagnosis of electric vehicle batteries using recurrent neural networks. *IEEE Transactions on Industrial Electronics*, 64(6):4885–4893, 2017. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85028720175&doi=10.1109%2fTIE.2017.2674593&partnerID=40&md5=10c153198d69ece50588cdebf44af8ee>, doi:[10.1109/TIE.2017.2674593](https://doi.org/10.1109/TIE.2017.2674593).
- [79] J. Wu, Y. Wang, X. Zhang, and Z. Chen. A novel state of health estimation method of li-ion battery using group method of data handling. *Journal of Power Sources*, 327:457–464, 2016. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-84979641117&doi=10.1016%2fj.jpowsour.2016.07.065&partnerID=40&md5=146697f1b656aa1c46d2814a2dfd73f4>, doi:[10.1016/j.jpowsour.2016.07.065](https://doi.org/10.1016/j.jpowsour.2016.07.065).

- [80] X. Hu, J. Jiang, D. Cao, and B. Egardt. Battery health prognosis for electric vehicles using sample entropy and sparse bayesian predictive modeling. *IEEE Transactions on Industrial Electronics*, 63(4):2645–2656, 2016. doi:10.1109/TIE.2015.2461523.
- [81] V. Klass, M. Behm, and G. Lindbergh. A support vector machine-based state-of-health estimation method for lithium-ion batteries under electric vehicle operation. *Journal of Power Sources*, 270:262–272, 2014. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-84905707183&doi=10.1016%2fj.jpowsour.2014.07.116&partnerID=40&md5=c4120fcd393edf21a3710ede5b4d13b6>, doi:10.1016/j.jpowsour.2014.07.116.
- [82] X. Hu, F. Feng, K. Liu, L. Zhang, J. Xie, and B. Liu. State estimation for advanced battery management: Key challenges and future trends. *Renewable and Sustainable Energy Reviews*, 114, 2019. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85070541160&doi=10.1016%2fj.rser.2019.109334&partnerID=40&md5=ea8c7b148d3dc1ab750550e4645a0471>, doi:10.1016/j.rser.2019.109334.
- [83] W. Waag, C. Fleischer, and D.U. Sauer. Critical review of the methods for monitoring of lithium-ion batteries in electric and hybrid vehicles. *Journal of Power Sources*, 258:321–339, 2014. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-84896102664&doi=10.1016%2fj.jpowsour.2014.02.064&partnerID=40&md5=e85ab1cce56321ba5e3727550f52b2c9>, doi:10.1016/j.jpowsour.2014.02.064.
- [84] D. Kim and D. Jung. Method of estimating maximum output of battery for hybrid electric vehicle, April 2009. URL: <https://www.freepatentsonline.com/7518375.html>.
- [85] L. W. Juang, P. J. Kollmeyer, T. M. Jahns, and R. D. Lorenz. Implementation of online battery state-of-power and state-of-function estimation in electric vehicle applications. In *2012 IEEE Energy Conversion Congress and Exposition (ECCE)*, pages 1819–1826, 2012. doi:10.1109/ECCE.2012.6342591.
- [86] R. D. Anderson, Y. Zhao, X. Wang, X. G. Yang, and Y. Li. Real time battery power capability estimation. In *2012 American Control Conference (ACC)*, pages 592–597, 2012. doi:10.1109/ACC.2012.6314892.
- [87] Alexander P. Schmidt, Matthias Bitzer, Árpád W. Imre, and Lino Guzzella. Model-based distinction and quantification of capacity loss and rate capability fade in li-ion batteries. *Journal of Power Sources*, 195(22):7634 – 7638, 2010. URL: <http://www.sciencedirect.com/science/article/pii/S0378775310009948>, doi:<https://doi.org/10.1016/j.jpowsour.2010.06.011>.
- [88] M. F. Samadi, S. M. M. Alavi, and M. Saif. Online state and parameter estimation of the li-ion battery in a bayesian framework. In *2013 American Control Conference*, pages 4693–4698, 2013. doi:10.1109/ACC.2013.6580563.
- [89] Keith S. Champlin and Kevin Bertness. Fundamentally new approach to battery performance analysis using dfra/dfis technology. In *INTELEC, International Telecommunications Energy Conference (Proceedings)*, pages 348–355, 2000. URL:

- <https://www.scopus.com/inward/record.uri?eid=2-s2.0-0034547415&partnerID=40&md5=297d49a3e8a53c175b4f9348ede5633a>.
- [90] P. Kumar and P. Bauer. Parameter extraction of battery models using multiobjective optimization genetic algorithms. In *Proceedings of EPE-PEMC 2010 - 14th International Power Electronics and Motion Control Conference*, pages T9106–T9110, 2010. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-78650366465&doi=10.1109%2fEPEPEMC.2010.5606653&partnerID=40&md5=81423f45a33cc1e0b2cc25d7fe4ab997,doi:10.1109/EPEPEMC.2010.5606653>.
- [91] S Nejad. *Adaptive Techniques for Estimation and Online Monitoring of Battery Energy Storage Devices*. PhD thesis, University of Sheffield, November 2016. URL: <http://etheses.whiterose.ac.uk/17681/>.
- [92] Hongwen He, Rui Xiong, and Hongqiang Guo. Online estimation of model parameters and state-of-charge of lifepo4 batteries in electric vehicles. *Applied Energy*, 89(1):413 – 420, 2012. Special issue on Thermal Energy Management in the Process Industries. URL: <http://www.sciencedirect.com/science/article/pii/S0306261911005010>, doi:<https://doi.org/10.1016/j.apenergy.2011.08.005>.
- [93] Hongwen He, Xiaowei Zhang, Rui Xiong, Yongli Xu, and Hongqiang Guo. Online model-based estimation of state-of-charge and open-circuit voltage of lithium-ion batteries in electric vehicles. *Energy*, 39(1):310 – 318, 2012. Sustainable Energy and Environmental Protection 2010. URL: <http://www.sciencedirect.com/science/article/pii/S030654421200014X>, doi:<https://doi.org/10.1016/j.energy.2012.01.009>.
- [94] Rui Xiong, Fengchun Sun, Xianzhi Gong, and Chenchen Gao. A data-driven based adaptive state of charge estimator of lithium-ion polymer battery used in electric vehicles. *Applied Energy*, 113:1421 – 1433, 2014. URL: <http://www.sciencedirect.com/science/article/pii/S0306261913007435>, doi:<https://doi.org/10.1016/j.apenergy.2013.09.006>.
- [95] M. A. Roscher, O. S. Bohlen, and D. U. Sauer. Reliable state estimation of multicell lithium-ion battery systems. *IEEE Transactions on Energy Conversion*, 26(3):737–743, 2011. doi:10.1109/TEC.2011.2155657.
- [96] G.L. Plett. Recursive approximate weighted total least squares estimation of battery cell total capacity. *Journal of Power Sources*, 196(4):2319–2331, 2011. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-78649499149&doi=10.1016%2fj.jpowsour.2010.09.048&partnerID=40&md5=a132f24fb496c6320fa924d8fc739230,doi:10.1016/j.jpowsour.2010.09.048>.
- [97] T. Kim and W. Qiao. A hybrid battery model capable of capturing dynamic circuit characteristics and nonlinear capacity effects. *IEEE Transactions on Energy Conversion*, 26(4):1172–1180, 2011. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-82155192197&doi=10.1109%2fTEC.2011.2167014&partnerID=40&md5=9348dbc4134b5f0a51ad5156f911e179,doi:10.1109/TEC.2011.2167014>.

- [98] S. Yuan, H. Wu, X. Zhang, and C. Yin. Online estimation of electrochemical impedance spectra for lithium-ion batteries via discrete fractional order model. In *2013 9th IEEE Vehicle Power and Propulsion Conference, IEEE VPPC 2013*, pages 224–229, 2013. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-84893293650&doi=10.1109%2fVPPC.2013.6671694&partnerID=40&md5=cc36e45fe7d7371ae71a0d8abc36aac3>, doi:10.1109/VPPC.2013.6671694.
- [99] Martin T. Hagan, Howard B. Demuth, Mark H. Beale, and Orlando De Jesús. *Neural Network Design (2nd Edition)*. Martin Hagan, 2014. <https://hagan.okstate.edu/nnd.html>.
- [100] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [101] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. [arXiv:1412.6980](https://arxiv.org/abs/1412.6980).
- [102] Gregory L. Plett. Extended kalman filtering for battery management systems of lipb-based hev battery packs: Part 3. state and parameter estimation. *Journal of Power Sources*, 134(2):277 – 292, 2004. URL: <http://www.sciencedirect.com/science/article/pii/S0378775304003611>, doi:<https://doi.org/10.1016/j.jpowsour.2004.02.033>.
- [103] Carlos Vidal, Phillip Kollmeyer, Mina Naguib, Pawel Malysz, Oliver Gross, and Ali Emadi. Robust xev battery state-of-charge estimator design using a feedforward deep neural network. *SAE International Journal of Advances and Current Practices in Mobility*, 2(5):2872–2880, apr 2020.
- [104] B. Saha and K. Goebel. "battery data set", 2007. URL: <http://ti.arc.nasa.gov/project/prognostic-data-repository>.
- [105] Phillip Kollmeyer, Carlos Vidal, Mina Naguib, and Michael Skells. Lg 18650hg2 li-ion battery data and example deep neural network xev soc estimator script, 2020. URL: [doi:10.17632/cp3473x7xv.3](https://doi.org/10.17632/cp3473x7xv.3).

Appendix A

Developed training functions

In this appendix chapter, the developed MATLAB functions used for training the models presented in Section 3.2 and Section 3.3 are displayed. These functions were developed based on the algorithms presented in Section 3.1.

Figures A.1 and A.2 display the code of the function used for calculating the outputs of all neurons during the forward propagation of the FNN network.

Figures A.3 and A.4 display the code of the function used for calculating the gradients of all weights and biases for a given training sample.

Figures A.5, A.6 and A.7 display the code of the functions used for updating the parameters (weights, biases) and training the model, based on the SGD algorithm.

Figures A.8, A.9 and A.10 display the code of the functions used for updating the parameters (weights, biases) and training the model, based on the SGDM algorithm.

Figures A.11, A.12 and A.13 display the code of the functions used for updating the parameters (weights, biases) and training the model, based on the RMSPROP algorithm.

Figures A.14, A.15 and A.16 display the code of the functions used for updating the parameters (weights, biases) and training the model, based on the ADAM algorithm.

```

function [x,y,yout] =
    ForwardProp(structure,activation,weights,biases,input)
%FORWARDPROP Forward Propagation.

layers = size(structure);

nw = 1;
nb = 1;
ny = 1;
nl = 1;
nout = 1;

nneurons = 0;

for i=structure
    nneurons = nneurons + i;
end

y = zeros(1,nneurons);
yout = zeros(1,structure(1,layers(1,2)));
x = zeros(1,nneurons);

for i=1:structure(1,1)
    y(1,ny) = input(1,i);
    x(1,ny) = y(1,ny);
    ny = ny + 1;
end

for i=structure(1,2:layers(1,2))
    for j=1:i
        for k=1:structure(1,nl)
            y(1,ny)=y(1,ny) + y(1,(ny-(structure(1,nl)-(k-1)+(j-1))))*weights(1,nw);
            nw = nw + 1;
        end
        y(1,ny) = y(1,ny) + biases(1,nb);
        x(1,ny) = y(1,ny);
        act = activation(1,nl);
        if (act < 0)
            y(1,ny)=clippedRelu(y(1,ny), -act);
        elseif (act > 0 && act < 1)
            y(1,ny)=leakyRelu(y(1,ny), act);
        elseif (act == 1)
            y(1,ny)=Relu(y(1,ny));
        elseif (act == 2)
            y(1,ny)=Sigmoid(y(1,ny));
        elseif (act == 3)
            y(1,ny)=Tansig(y(1,ny));
        end
        if (nl==(layers(1,2)-1))
            yout(1,nout)=y(1,ny);
            nout = nout + 1;
        end
    end
end

```

Figure A.1: Forward propagation function - page (1/2).

```
        end
        nb = nb + 1;
        ny = ny + 1;
    end
    nl = nl + 1;
end
end
```

Published with MATLAB® R2019a

```

function [gweights,gbiases] =
    BackProp(structure,activation,weights,biases,x,y,out)
%BACKPROP Calculates gradients for a training sample.

layers = size(structure);

nw = size(weights,2);
nb = size(biases,2);
ny = size(y,2);
nd = ny;
nl = size(structure,2);
nls = nl;

delta = zeros(1, ny);
gweights = zeros(1, nw);
gbiases = zeros(1, nb);

ny = ny - structure(1,nls);
for i=1:structure(1,nls)
    delta(1,nd) = -(out(1,(structure(1,nls)-(i-1))) - y(1,nd));
    act = activation(1,nls-1);
    if (act < 0)
        delta(1,nd)=delta(1,nd)*deltacappedRelu(x(1,nd),-act);
    elseif (act > 0 && act < 1)
        delta(1,nd)=delta(1,nd)*deltaLeakyRelu(x(1,nd),act);
    elseif (act == 1)
        delta(1,nd)=delta(1,nd)*deltaRelu(x(1,nd));
    elseif (act == 2)
        delta(1,nd)=delta(1,nd)*deltaSigmoid(x(1,nd));
    elseif (act == 3)
        delta(1,nd)=delta(1,nd)*deltaTansig(x(1,nd));
    end
    for j=1:structure(1,(nls-1))
        gweights(1,nw)=delta(1,nd)*y(1,ny);
        nw = nw - 1;
        ny = ny - 1;
    end
    gbiases(1,nb)=delta(1,nd);
    nb = nb - 1;
    nd = nd - 1;
    ny = ny + structure(1,(nls-1));
end
nls = nls - 1;
for i=1:(nl-2)
    ny = ny - structure(1,nls);
    for j=1:structure(1,nls)
        for k=1:structure(1,nls+1)
            delta(1,nd)=delta(1,nd)+delta(1,(nd+k+(j-1)))*weights(1,nw
+(j-1)*structure(1,nls-1)+k*structure(1,nls)-(j-1));
        end
        act = activation(1,nls-1);
        if (act < 0)

```

Figure A.3: Backward propagation function - page (1/2).

```

        delta(1,nd)=delta(1,nd)*deltaclippedRelu(x(1,nd),-act);
elseif (act > 0 && act < 1)
    delta(1,nd)=delta(1,nd)*deltaleakyRelu(x(1,nd),act);
elseif (act == 1)
    delta(1,nd)=delta(1,nd)*deltaRelu(x(1,nd));
elseif (act == 2)
    delta(1,nd)=delta(1,nd)*deltaSigmoid(x(1,nd));
elseif (act == 3)
    delta(1,nd)=delta(1,nd)*deltaTansig(x(1,nd));
end
for k=1:structure(1,(nls-1))
    gweights(1,nw)=delta(1,nd)*y(1,ny);
    nw = nw - 1;
    ny = ny - 1;
end
gbiases(1,nb)=delta(1,nd);
nb = nb - 1;
nd = nd - 1;
ny = ny + structure(1,(nls-1));
end
nls = nls - 1;
end
end

```

Published with MATLAB® R2019a

```
function [weights, biases] = TrainSGD(weights, biases, gweights,  
    gbiases, learn_rate)  
%TRAINSGD Alters weights with SGD optimization  
%SGD = Stochastic Gradient Descent  
  
weights = weights - learn_rate*gweights;  
biases = biases - learn_rate*gbiases;  
end
```

Published with MATLAB® R2019a

Figure A.5: SGD parameters update function.

```

function [wres, bres, RMSEres, w, b] =
    trainModelSGD(structure, activation, w, b, trainVals, validationVals, validation_period)
%TRAINMODELSGD Trains model with SGD optimization

trainValsr = trainVals(:, randperm(size(trainVals, 2)));
Xr = trainValsr(1:(size(trainVals, 1)-1),:);
Yr = trainValsr(size(trainVals, 1),:);

nResults = fix(max_epochs/validation_period);

validationResults = zeros(nResults, size(validationVals, 2));
wres = zeros(nResults, size(w, 2));
bres = zeros(nResults, size(b, 2));
RMSEres = zeros(1, nResults);
n_validation = 1;

SOCpredict = zeros(1, size(Yr, 2));

gws=zeros(size(w,1),size(w,2));
gbs=zeros(size(b,1),size(b,2));

for i=1:max_epochs

    if (randomize==1)
        trainValsr = trainVals(:, randperm(size(trainVals, 2)));
        Xr = trainValsr(1:(size(trainVals, 1)-1),:);
        Yr = trainValsr(size(trainVals, 1),:);
    end

    for j=1:size(Yr, 2)

        [x, y, yout] = ForwardProp(structure, activation, w, b, Xr(:, j)');

        SOCpredict(1, j) = yout(1, 1);

        [gw, gb] = BackProp(structure, activation, w, b, x, y,
            Yr(1, j));

        gws = gws + gw + reg_param*w;
        gbs = gbs + gb;

        if (rem(j, mini_batch_size)==0)
            gws=gws/mini_batch_size;
            gbs=gbs/mini_batch_size;
            [w, b] = TrainSGD(w, b, gws, gbs, learn_rate);
            gws(1, :) = 0;
            gbs(1, :) = 0;
        end
    end

    gws(1, :) = 0;
    gbs(1, :) = 0;
end

```

Figure A.6: SGD algorithm training function - page (1/2).

```

SOCerror=Yr-SOCpredict;

rootMSE = RMSE(SOCerror);
meanAE = MAE(SOCerror);

fprintf('iteracao %d -> RMSE = %f%% |-| MAE = %f%% |-| MAX = %f%
%f%% |-| MIN = %f%%\n', i, rootMSE*100, meanAE*100, max(SOCerror)*100,
min(SOCerror)*100);

if (rem(i,learn_rate_drop_period)==0)
    learn_rate = learn_rate*learn_rate_drop_factor;
    fprintf('learn rate baixou %d para %d \n',
learn_rate_drop_factor, learn_rate);
end

if (rem(i,validation_period)==0)

    for j=1:size(validationVals,2)

        [x,y,yout] =
ForwardProp(structure,activation,w,b,validationVals(1:
(size(validationVals,1)-1),j));

        validationResults(n_validation,j) = yout(1,1);

    end

    SOCerror = validationVals(size(validationVals,1),:) -
validationResults(n_validation,:);

    rootMSE = RMSE(SOCerror);
    meanAE = MAE(SOCerror);

    wres(n_validation,:) = w;
    bres(n_validation,:) = b;
    RMSEres(1,n_validation) = rootMSE;

    fprintf('Validacao %d -> RMSE = %f%% |-| MAE = %f%% |-| MAX
= %f%% |-| MIN = %f%%\n', n_validation, rootMSE*100, meanAE*100,
max(SOCerror)*100, min(SOCerror)*100);

    n_validation = n_validation + 1;

end
end
end

```

Published with MATLAB® R2019a

Figure A.7: SGD algorithm training function - page (2/2).

```
function [weights, biases, wa, ba] = TrainSGDM(weights, biases,
    gweights, gbiases, learn_rate, wa, ba, momentum)
%TRAINS GDM Alters weights with SGDM optimization
%SGDM = Stochastic Gradient Descent with Momentum
%Typical momentum values k = 0.9 ou >0.9
%wa and ba store previous values of weights and biases

auxW = weights;
auxB = biases;

weights = weights - learn_rate*gweights + momentum*(weights-wa);
biases = biases - learn_rate*gbiases + momentum*(biases-ba);

wa = auxW;
ba = auxB;

end
```

Published with MATLAB® R2019a

Figure A.8: SGDM parameters update function.

```

function [wres, bres, RMSEres, w, b] =
    trainModelSGDM(structure,activation,w,b,trainVals,validationVals,validation_perio
%TRAINMODELSGDM Trains model with SGDM optimization

wa=w;
ba=b;

trainValsr = trainVals(:, randperm(size(trainVals, 2)));
Xr = trainValsr(1:(size(trainVals, 1)-1),:);
Yr = trainValsr(size(trainVals, 1),:);

nResults = fix(max_epochs/validation_period);

validationResults = zeros(nResults, size(validationVals, 2));
wres = zeros(nResults,size(w,2));
bres = zeros(nResults,size(b,2));
RMSEres = zeros(1,nResults);
n_validation = 1;

SOCpredict = zeros(1,size(Yr,2));

gws=zeros(size(w,1),size(w,2));
gbs=zeros(size(b,1),size(b,2));

for i=1:max_epochs

    if (randomize==1)
        trainValsr = trainVals(:, randperm(size(trainVals, 2)));
        Xr = trainValsr(1:(size(trainVals, 1)-1),:);
        Yr = trainValsr(size(trainVals, 1),:);
    end

    for j=1:size(Yr,2)

        [x,y,yout] = ForwardProp(structure,activation,w,b,Xr(:,j)');

        SOCpredict(1,j) = yout(1,1);

        [gw,gb] = BackProp(structure, activation, w, b, x, y,
        Yr(1,j));

        gws = gws + gw + reg_param*w;
        gbs = gbs + gb;

        if (rem(j,mini_batch_size)==0)
            gws=gws/mini_batch_size;
            gbs=gbs/mini_batch_size;
            [w, b, wa, ba] =
            TrainSGDM(w,b,gws,gbs,learn_rate,wa,ba,momentum);
            gws(1,:) = 0;
            gbs(1,:) = 0;
        end
    end
end

```

Figure A.9: SGDM algorithm training function - page (1/2).

```

end

gws(1,:) = 0;
gbs(1,:) = 0;

SOCerror=Yr-SOCpredict;

rootMSE = RMSE(SOCerror);
meanAE = MAE(SOCerror);

fprintf('iteracao %d -> RMSE = %f%% |-| MAE = %f%% |-| MAX = %f%%
% |-| MIN = %f%%\n', i, rootMSE*100, meanAE*100, max(SOCerror)*100,
min(SOCerror)*100);

if (rem(i,learn_rate_drop_period)==0)
    learn_rate = learn_rate*learn_rate_drop_factor;
    fprintf('learn rate baixou %d para %d \n',
learn_rate_drop_factor, learn_rate);
end

if (rem(i,validation_period)==0)

    for j=1:size(validationVals,2)

        [x,y,yout] =
ForwardProp(structure,activation,w,b,validationVals(1:
(size(validationVals,1)-1),j));

        validationResults(n_validation,j) = yout(1,1);

    end

    SOCerror = validationVals(size(validationVals,1),:) -
validationResults(n_validation,:);

    rootMSE = RMSE(SOCerror);
    meanAE = MAE(SOCerror);

    wres(n_validation,:) = w;
    bres(n_validation,:) = b;
    RMSEres(1,n_validation) = rootMSE;

    fprintf('Validacao %d -> RMSE = %f%% |-| MAE = %f%% |-| MAX
= %f%% |-| MIN = %f%%\n', n_validation, rootMSE*100, meanAE*100,
max(SOCerror)*100, min(SOCerror)*100);

    n_validation = n_validation + 1;
end
end
end

```

Published with MATLAB® R2019a

Figure A.10: SGDM algorithm training function - page (2/2).

```

function [weights, biases, vw, vb] = TrainRMSProp(weights, biases,
    gweights, gbiases, learn_rate, vw, vb, B2, epsilon)
%TRAINRMSPROP Alters weights with RMSProp optimization
%RMSProp
%Typical decay rate value B2 = (0.9, 0.99, 0.999)
%Typical epsilon value = 10^-8 (just to help stabilize small number
    division)

for i=1:size(vw,2)
    vw(1,i) = B2 * vw(1,i) + (1-B2)*(gweights(1,i)^2);
    weights(1,i) = weights(1,i) - learn_rate*gweights(1,i)/
        (sqrt(vw(1,i))+epsilon);
end

for i=1:size(vb,2)
    vb(1,i) = B2 * vb(1,i) + (1-B2)*(gbiases(1,i)^2);
    biases(1,i) = biases(1,i) - learn_rate*gbiases(1,i)/
        (sqrt(vb(1,i))+epsilon);
end

end

```

Published with MATLAB® R2019a

Figure A.11: RMSPROP parameters update function.

```

function [wres, bres, RMSEres, w, b] =
    trainModelRMSPROP(structure,activation,w,b,trainVals,validationVals,validation_pe
%TRAINMODELRMSPROP Trains model with RMSPROP optimization

vw=zeros(size(w,1),size(w,2));
vb=zeros(size(b,1),size(b,2));

trainValsr = trainVals(:, randperm(size(trainVals, 2)));
Xr = trainValsr(1:(size(trainVals, 1)-1),:);
Yr = trainValsr(size(trainVals, 1),:);

nResults = fix(max_epochs/validation_period);

validationResults = zeros(nResults, size(validationVals, 2));
wres = zeros(nResults,size(w,2));
bres = zeros(nResults,size(b,2));
RMSEres = zeros(1,nResults);
n_validation = 1;

SOCpredict = zeros(1,size(Yr,2));

gws=zeros(size(w,1),size(w,2));
gbs=zeros(size(b,1),size(b,2));

for i=1:max_epochs
    if (randomize==1)
        trainValsr = trainVals(:, randperm(size(trainVals, 2)));
        Xr = trainValsr(1:(size(trainVals, 1)-1),:);
        Yr = trainValsr(size(trainVals, 1),:);
    end

    for j=1:size(Yr,2)

        [x,y,yout] = ForwardProp(structure,activation,w,b,Xr(:,j)');

        SOCpredict(1,j) = yout(1,1);

        [gw,gb] = BackProp(structure, activation, w, b, x, y,
Yr(1,j));

        gws = gws + gw + reg_param*w;
        gbs = gbs + gb;

        if (rem(j,mini_batch_size)==0)
            gws=gws/mini_batch_size;
            gbs=gbs/mini_batch_size;
            [w, b, vw, vb] =
TrainRMSProp(w,b,gws,gbs,learn_rate,vw,vb,B2,epsilon);
            gws(1,:) = 0;
            gbs(1,:) = 0;
        end
    end
end

```

Figure A.12: RMSPROP algorithm training function - page (1/2).

```

end
gws(1,:) = 0;
gbs(1,:) = 0;
SOCerror=Yr-SOCpredict;
rootMSE = RMSE(SOCerror);
meanAE = MAE(SOCerror);

fprintf('iteracao %d -> RMSE = %f%% |-| MAE = %f%% |-| MAX = %f%%
% |-| MIN = %f%%\n', i, rootMSE*100, meanAE*100, max(SOCerror)*100,
min(SOCerror)*100);

if (rem(i,learn_rate_drop_period)==0)
    learn_rate = learn_rate*learn_rate_drop_factor;
    fprintf('learn rate baixou %d para %d \n',
learn_rate_drop_factor, learn_rate);
end
if (rem(i,validation_period)==0)
    for j=1:size(validationVals,2)

        [x,y,yout] =
ForwardProp(structure,activation,w,b,validationVals(1:
(size(validationVals,1)-1),j));
        validationResults(n_validation,j) = yout(1,1);
    end
    SOCerror = validationVals(size(validationVals,1),:) -
validationResults(n_validation,:);

    rootMSE = RMSE(SOCerror);
    meanAE = MAE(SOCerror);

    wres(n_validation,:) = w;
    bres(n_validation,:) = b;
    RMSEres(1,n_validation) = rootMSE;

    fprintf('Validacao %d -> RMSE = %f%% |-| MAE = %f%% |-| MAX
= %f%% |-| MIN = %f%%\n', n_validation, rootMSE*100, meanAE*100,
max(SOCerror)*100, min(SOCerror)*100);
    n_validation = n_validation + 1;
end
end
end

```

Published with MATLAB® R2019a

Figure A.13: RMSPROP algorithm training function - page (2/2).

```

function [weights, biases, vw, vb, mw, mb] = TrainADAM(weights,
    biases, gweights, gbiases, learn_rate, vw, vb, mw, mb, B1, B2,
    epsilon, iter)
%TRAINADAM Alters weights with ADAM optimization
%ADAM
%Typical decay rate value B1 = 0.9
%Typical decay rate value B2 = 0.999
%Typical epsilon value = 10^-8 (just to help stabilize small number
    division)
%Iter is the current iteration

for i=1:size(mw,2)

    mw(1,i) = B1 * mw(1,i) + (1-B1)*gweights(1,i);
    vw(1,i) = B2 * vw(1,i) + (1-B2)*(gweights(1,i)^2);
    if (iter<19000)
        weights(1,i) = weights(1,i) - learn_rate*(mw(1,i)/(1-
B1^iter))/(sqrt((vw(1,i)/(1-B2^iter)))+epsilon);
    else % If iter>19000, B1^iter=B2^iter=+-0, no point in keeping
        calculating this
        weights(1,i) = weights(1,i) - learn_rate*mw(1,i)/
(sqrt(vw(1,i))+epsilon);
    end
end

for i=1:size(mb,2)

    mb(1,i) = B1 * mb(1,i) + (1-B1)*gbiases(1,i);
    vb(1,i) = B2 * vb(1,i) + (1-B2)*(gbiases(1,i)^2);
    if (iter<19000)
        biases(1,i) = biases(1,i) - learn_rate*(mb(1,i)/(1-B1^iter))/
(sqrt((vb(1,i)/(1-B2^iter)))+epsilon);
    else % If iter>19000, B1^iter=B2^iter=+-0, no point in keeping
        calculating this
        biases(1,i) = biases(1,i) - learn_rate*mb(1,i)/
(sqrt(vb(1,i))+epsilon);
    end
end

end

```

Published with MATLAB® R2019a

Figure A.14: ADAM parameters update function.

```

function [wres, bres, RMSEres, w, b] =
    trainModelADAM(structure,activation,w,b,trainVals,validationVals,validation_perio
%TRAINMODELADAM Trains model with ADAM optimization

mw=zeros(size(w,1),size(w,2));
mb=zeros(size(b,1),size(b,2));
vw=zeros(size(w,1),size(w,2));
vb=zeros(size(b,1),size(b,2));
iter = 1;

trainValsr = trainVals(:, randperm(size(trainVals, 2)));
Xr = trainValsr(1:(size(trainVals, 1)-1),:);
Yr = trainValsr(size(trainVals, 1),:);

nResults = fix(max_epochs/validation_period);

validationResults = zeros(nResults, size(validationVals, 2));
wres = zeros(nResults,size(w,2));
bres = zeros(nResults,size(b,2));
RMSEres = zeros(1,nResults);
n_validation = 1;

SOCpredict = zeros(1,size(Yr,2));

gws=zeros(size(w,1),size(w,2));
gbs=zeros(size(b,1),size(b,2));

for i=1:max_epochs

    if (randomize==1)
        trainValsr = trainVals(:, randperm(size(trainVals, 2)));
        Xr = trainValsr(1:(size(trainVals, 1)-1),:);
        Yr = trainValsr(size(trainVals, 1),:);
    end

    for j=1:size(Yr,2)

        [x,y,yout] = ForwardProp(structure,activation,w,b,Xr(:,j)');

        SOCpredict(1,j) = yout(1,1);

        [gw,gb] = BackProp(structure, activation, w, b, x, y,
        Yr(1,j));

        gws = gws + gw + reg_param*w;
        gbs = gbs + gb;

        if (rem(j,mini_batch_size)==0)
            gws=gws/mini_batch_size;
            gbs=gbs/mini_batch_size;
            [w, b, vw, vb, mw, mb] = TrainADAM(w,b,gws,gbs,learn_rate,
            vw, vb, mw, mb, B1, B2, epsilon, iter);

```

Figure A.15: ADAM algorithm training function - page (1/2).

```

        if (iter<20000)
            iter = iter + 1;
        end
        gws(1,:) = 0;
        gbs(1,:) = 0;
    end
end

gws(1,:) = 0;
gbs(1,:) = 0;
SOCerror=Yr-SOCpredict;
rootMSE = RMSE(SOCerror);
meanAE = MAE(SOCerror);

fprintf('iteracao %d -> RMSE = %f%% |-| MAE = %f%% |-| MAX = %f%%
% |-| MIN = %f%%\n', i, rootMSE*100, meanAE*100, max(SOCerror)*100,
min(SOCerror)*100);

if (rem(i,learn_rate_drop_period)==0)
    learn_rate = learn_rate*learn_rate_drop_factor;
    fprintf('learn rate baixou %d para %d \n',
    learn_rate_drop_factor, learn_rate);
end
if (rem(i,validation_period)==0)

    for j=1:size(validationVals,2)

        [x,y,yout] =
        ForwardProp(structure,activation,w,b,validationVals(1:
        (size(validationVals,1)-1),j));
        validationResults(n_validation,j) = yout(1,1);
    end
    SOCerror = validationVals(size(validationVals,1),:) -
    validationResults(n_validation,:);

    rootMSE = RMSE(SOCerror);
    meanAE = MAE(SOCerror);

    wres(n_validation,:) = w;
    bres(n_validation,:) = b;
    RMSEres(1,n_validation) = rootMSE;

    fprintf('Validacao %d -> RMSE = %f%% |-| MAE = %f%% |-| MAX
= %f%% |-| MIN = %f%%\n', n_validation, rootMSE*100, meanAE*100,
max(SOCerror)*100, min(SOCerror)*100);

    n_validation = n_validation + 1;
end
end
end

```

Published with MATLAB® R2019a

Figure A.16: ADAM algorithm training function - page (2/2).