Universität Stuttgart
Institut für Fahrzeugtechnik Stuttgart

Development of a Data-Driven Battery Aging Model

Development of a Data-Driven Battery Aging Model

Bachelorarbeit

von
cand. mach Patrick Richter
Matr. Nr. 3346952

Betreuer: Andre Ebel, M.Sc.

Prüfer: Prof. Dr.-Ing. H.-C. Reuss

Vorgelegt an der Universität Stuttgart
Institut für Fahrzeugtechnik Stuttgart
Lehrstuhl Kraftfahrzeugmechatronik

2020

List of Contents

List of Abbreviations

EV	**E**lectric **V**ehicle
LCO	**L**ithium **C**obalt **O**xide
NMC	Lithium **N**ickel **M**anganese **C**obalt Oxide
NCA	Lithium **N**ickel **C**obalt Oxide doped with **A**lumina
LMO	**L**ithium **M**anganese **O**xide
LFP	**L**ithium Iron **Ph**os**p**hate
CC-CV	**C**onstant **C**urrent-**C**onstant **V**oltage
LSTM	**L**ong **S**hort-**T**erm **M**emory Network
BPTT	**B**ack**p**ropagation **T**hrough **T**ime
MSE	**M**ean **S**quared **E**rror
MAE	**M**ean **A**bsolute **E**rror
SGD	**S**tochastic **G**radient **D**escent
AdaGrad	**Ada**ptive **Grad**ient Algorithm
RMSProp	**R**oot **M**ean **S**quare **Prop**agation
Adam	**Ada**ptive **m**oment estimation
ReLU	**Re**ctified **L**inear **U**nit
DC	**D**irect **C**urrent

Formula Symbols

C	A·h	Capacity
$I(t)$	A	Current
C_N	A·h	Nominal capacity
W	W·h	Electrical Energy
$\overline{U}$	V	Average voltage
$P(t)$	W	Electric power
$U(t)$	V	Voltage
$\mathrm{C-rate}(t)$	-	Current normalized against the capacity (C-rate)
SOC	-	State of charge
SOH	-	State of health
$C_{\mathrm{remaining}}$	-	Remaining capacity
$\tilde{y}$	-	Output of a basic Neural Network unit
$\widetilde{x_\iota}$	-	Input of a basic Neural Network unit
$\widetilde{w_\iota}$	-	Input weight of a basic Neural Network unit
$\tilde{b}$	-	Bias of a basic Neural Network unit
$x(t)$	-	Input of an LSTM unit at time step t
$h(t)$	-	Output of an LSTM unit at time step t
$c(t)$	-	Internal state of an LSTM unit at time step t
$f(t)$	-	Forget gate of an LSTM unit at time step t
$i(t)$	-	Input gate of an LSTM unit at time step t
$\tilde{c}(t)$	-	Input state update of an LSTM unit at time step t

$o(t)$	-	Output gate of an LSTM unit at time step t
W_{fh}, W_{fx}	-	Forget gate weights
W_{ih}, W_{ix}	-	Input gate weights
$W_{\tilde{c}h}$, $W_{\tilde{c}x}$	-	Internal state update weights
W_{oh}, W_{ox}	-	Output gate weights
b_f	-	Forget gate bias
b_i	-	Input gate bias
$b_{\tilde{c}}$	-	Internal state update bias
b_o	-	Output gate bias
Q	-	Loss function
Q_{MSE}	-	Mean sqaured error loss function
Q_{MAE}	-	Mean absolute error loss function
$y(t)$	-	Predicted value of at time step t
$\hat{y}(t)$	-	Expected value of at time step t
w	-	Trainable parameters
∇Q	-	Loss function gradient
η	-	Learning rate

List of Figures

List of Tables

Abstract

Predicting battery aging for lithium-ion batteries is crucial, because of this component's exceptional impact on the overall electric vehicle's durability. Since it often takes months or years to run experiments examining battery aging, battery aging models can safe plenty of development time. Conventional models often find it difficult to depict the complex, nonlinear aging mechanisms. Battery aging models based on Machine Learning have shown promising results. In this work, a data-based Long Short-Term Memory network (LSTM) model is developed, ultilizing a publicly available dataset of 174 lithium-ion batteries. The LSTM's hyperparameters are optimized with a grid search, whereby the best model achieves a mean average error (MAE) of only 0.33 % in predicting the course of the state of health (SOH). It predicts the SOH profile of all validation samples in a 1.3 % MAE margin, with 72 % of the samples in a margin of 0.3 %. By applying the model to a real-life EV charging profile, the work advances into unchartered territory and proposes a model advancement to overcome the problems encountered.

1 Introduction

As the deployment of electric powertrains increases and will continue to do so in the years to come, the automotive industry faces new challenges in developing and designing the components. The battery plays a key role in the electric powertrain. Its characteristics impact range, charging behavior, weight, power, energy consumption, and durability of electric vehicles (EVs).

A major issue with batteries in EVs is aging. By causing capacity loss and reducing the battery's power output, battery aging determines the electric powertrain's durability. Modeling battery aging for different charging profiles and battery configurations, though, is a difficult task in the development process.

Models, accurately estimating battery aging, would safe a substantial amount of time in the development process, as it often takes months or years to get experimental feedback in terms of battery aging. The impact of new fast-charging profiles on battery aging could be estimated within seconds, instead of running trials taking several months or years.

Several models estimate battery aging, with the most common approaches being physicochemical and fatigue models [1]. Lately, with advances in computational power and data generation, mathematical models based on Neural Networks have become more attractive. A number of studies have already employed data-based mathematical models to predict battery aging with impressive results on datasets with specific boundary conditions [2, 3, 4, 5, 6]. No study, however, has employed their models on a more general dataset.

This work goes one step further, by developing a data-based battery aging model and then applying it to a more general dataset in form of a real-life EV charging profile, checking it for plausibility. For training the model, a publicly available dataset of 174 lithium-ion battery cells cycled to failure is used. The model is based on an LSTM, a Neural Network specializing in sequence prediction problems, built with the Keras Machine Learning library in Python.

2 Literature Review

2.1 Lithium-ion Batteries

Lithium-ion batteries are rechargeable batteries containing lithium compounds in their electrodes and the electrolyte. They are commonly used in mobile electronic devices such as cellular phones and laptops. Additionally, their importance in electric mobility is rising and will continue to do so. Applications include electric bicycles, scooters, buses, and cars. Driven by the EV segment's growth, the global battery market is set to tenfold its sales in GWh from 120 GWh in 2017 to 1200 GWh in 2030 [7]. Lithium-ion batteries' advantages are their long cycling life, high energy density, low self-discharge rates, and high efficiency during charging and discharging compared to other batteries [8, 9].

2.1.1 Battery Specifications and Definitions

Before focusing on the principle of lithium-ion batteries, this chapter will describe important battery specifications and definitions.

Capacity:

Capacity C, mostly stated in Ah, is defined as the maximum amount of electric charge a fully charged battery can deliver under specific discharge conditions, such as discharge current, temperature, cut-off voltage, and the stage of battery aging. It is calculated as the integral of the discharge current $I(t)$ over time for one cycle. [9]

$$\mathrm{C} = \int I(t)dt \tag{2.1}$$

The nominal capacity C_N is the amount of electric charge a fresh battery (no battery aging) can deliver.

Electrical energy:

The electrical energy W, the product of the capacity C and the average discharge voltage $\overline{U}$, describes the amount of energy a battery can deliver. Its unit is Wh. [9]

$$W = C \cdot \overline{U} \tag{2.2}$$

Electric Power:

The electric power $P(t)$ (unit is W) is defined as the product of the discharge voltage $U(t)$ and the discharge current $I(t)$. [9]

$$P(t) = U(t) \cdot I(t) \tag{2.3}$$

C-rate:

Instead of using Ampere (A), the current for batteries is usually described with the dimensionless C-rate. It normalizes against the battery capacity that often varies between different battery cells. The C-rate is defined as the quotient of the current in A and the battery capacity in Ah. [10]

$$\mathrm{C-rate}(t) = \frac{I(t) \cdot \mathrm{h}}{C_N} \tag{2.4}$$

A C-rate of x is often referred to as x C.

SOC:

The SOC describes the ratio of the remaining battery capacity (battery not fully charged) compared to the maximum battery capacity. [10]

$$\mathrm{SOC} = \frac{C_{remaining}}{C} \tag{2.5}$$

SOH:

The SOH describes the ratio of the battery capacity after a particular number of cycles compared to the nominal battery capacity. [10]

$$\mathrm{SOH} = \frac{C}{C_N} \tag{2.6}$$

2.1.2 Components and Basic Principle

Figure 2.1 shows the set-up of a lithium-ion battery. In between the two electrodes, the lithium-ion-conducting (Li^+) electrolyte is placed. The cathode (negative electrode) uses aluminum as the conductor, while copper acts as the conductor for the anode (positive electrode). Both electrodes are made of active materials. The electrodes are protected from direct contact with each other by the separator, a porous membrane that lets lithium-ions through. [9]

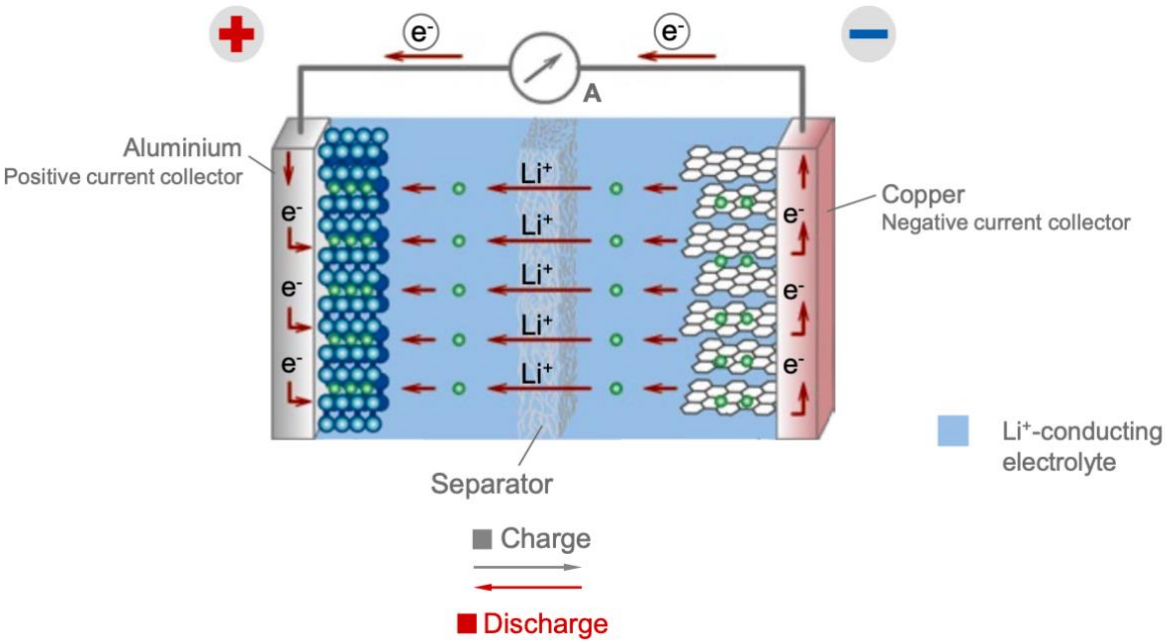

Figure 2.1 Basic principle of a lithium-ion battery **[1]**

The discharging process, shown in Figure 2.1, transforms chemical energy into electrical energy. Due to the potential difference between the two electrodes, electrons migrate to the positive electrode through the conductor. Simultaneously, lithium-ions are deintercalated from the negative electrode and move to the positive electrode through the electrolyte so that both electrodes are electrically neutral. Charging, on the other hand, transforms electrical energy into chemical energy by reversing the discharging process. [9]

Battery cells exist in various shapes, e.g., cylindrical, prismatic, or pouch cells. Several battery cells can be connected in series or parallel, dependent on the required capacity, current, and voltage, to form a battery system. In most cases, the battery system uses a battery management system to control and monitor cell voltage, temperature, and current. [9]

2.1.3 Cathode Materials

The cathode of a lithium-ion battery consists of lithium transition metals, strongly impacting the battery's characteristics, e.g., capacity, nominal voltage, and energy density.

Various materials are used as cathodes, with the most common being lithium cobalt oxide (LCO), lithium nickel manganese cobalt oxide (NMC), lithium nickel cobalt oxide doped with alumina (NCA), lithium manganese oxide (LMO), and lithium iron phosphate (LFP). [11]

Table 2.1 compares the characteristics of these cathode materials.

Table 2.1 Capacity, nominal voltage, and energy density of the different materials **[9, 12]**

Material	**Capacity/Ah kg^{-1}**	**Nominal voltage/V**	**Energy density/Wh·kg^{-1}**
LCO	160	3,9	195
NMC	160	3,7	205
NCA	200	3,7	220
LMO	100	4,1	150
LFP	150	3,3	90-130

2.1.4 Charging Procedures

Constant Current–Constant Voltage (CC-CV), shown in Figure 2.2, is the standard charging procedure for lithium-ion batteries. Firstly, the battery cell is charged by a predefined constant current. During this current-regulation phase, the voltage will increase to the maximum safe threshold. Subsequently, the battery cell enters the voltage-regulation phase. It is charged by a constant voltage while the current decreases. The process either ends after a particular time has passed or when a specific current value is reached. [13, 14, 9]

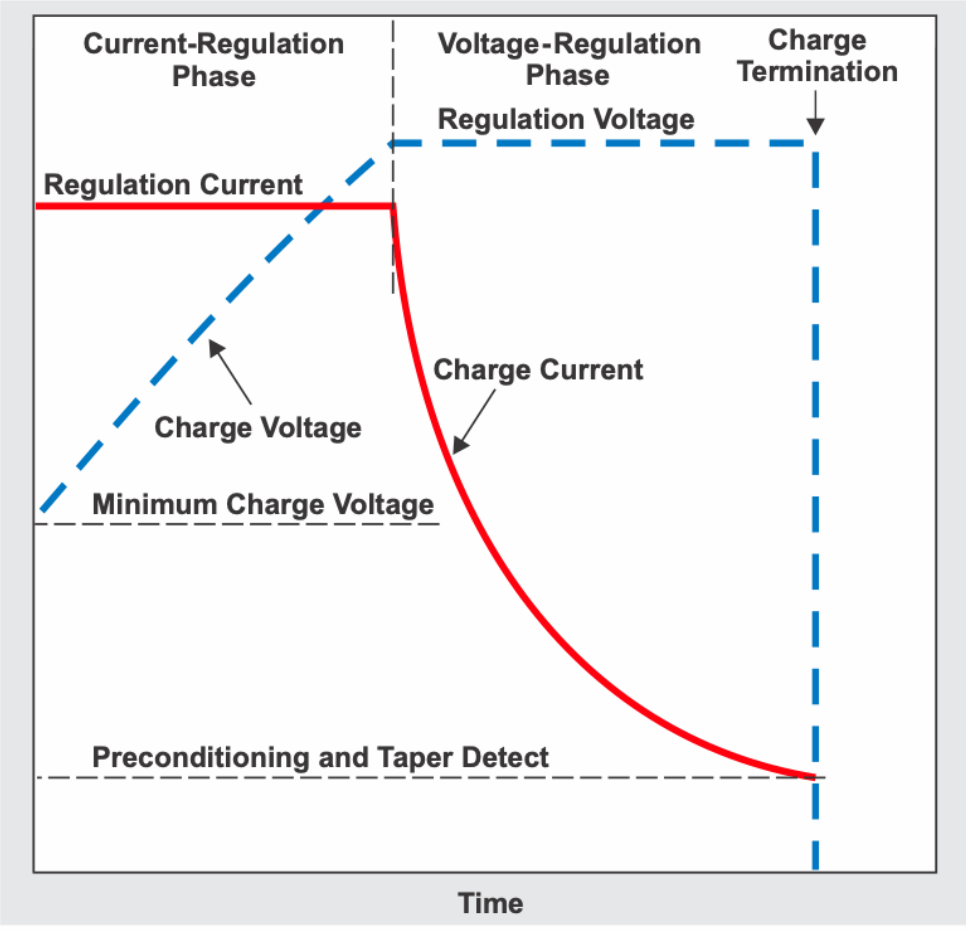

Figure 2.2 CC-CV charging profile for lithium-ion batteries **[14]**

2.1.5 Battery Aging and Cycle Life

Characteristics of a battery alter over its lifetime. Many different materials are in contact inside a battery cell. They unintentionally react with each other, leading to battery aging. This causes a decrease in the maximum battery capacity over time. Furthermore, the cell's internal resistance decreases, reducing the power output in the process. [9]

The cycle life usually describes the number of cycles (fully charged and discharged) until the battery reaches a failure threshold in terms of remaining capacity or internal resistance. Most manufacturers use 80% of the nominal capacity as a threshold value [15].

Battery aging can be separated into two types: calendar aging and cycling aging. Calendar aging refers to the time related aging mechanisms, whereas cycling aging is directly related to the charging and discharging of the battery.

Calendar aging is mainly impacted by the storage SOC of the battery and potential of the anode, whereby high storage SOCs and anodes with low potential should be avoided. [16]

Many different stress factors influence the velocity of cycling aging. The temperature has a crucial impact, with a minimum degradation at around 25°C. Extreme Temperatures (above 45°C and below 0°C) accelerate capacity loss [17]. Additional factors are charge and discharge current and depth of discharge, which describes how much capacity is removed from the fully charged battery each cycle. This work will only concentrate on cycling aging.

As the correlations between stress factors and battery degradation are often nonlinear, it is challenging to model battery aging. Chapter 2.2 will focus on battery aging models in detail.

2.2 Battery Aging Models

There already exist several models to predict battery aging over its cycle life. Those models can be grouped into three different approaches: physicochemical models, fatigue models, and mathematical models.

2.2.1 Physicochemical Models

The development of physicochemical models can be separated into two steps. In the first step, a battery model, built on the cell's electrochemical equations, fundamental laws of electrodynamics, and knowledge of diffusion processes, is developed. With this model, the most crucial state variables in the cell can be determined at any time, e.g., local potential, local state of charge, local current density, etc. [18]

As the cell's internal state is known, the information is now used to draw conclusions on battery aging processes in the second step. That step involves results from experiments on aging effects in the laboratory. In theory, this approach allows a prediction of the battery aging's current state at any time and every condition. [18]

However, there are several problems with these models. Firstly, to develop a physicochemical model of the cell, many parameters that might evolve during aging have to be known. Furthermore, aging mechanisms are partly known, but partly still vague and, therefore, difficult to formulate into an equation. [1]

2.2.2 Fatigue Models

Fatigue models are inspired by the idea that single events influence the aging process incrementally. Thus, the capacity loss is seen as an accumulation of various events that decrease the remaining capacity, whereby the amount of loss depends on a number of factors. This is a standard approach in other fields of engineering, such as materials science. [1]

A common approach is the Wöhler method. It is based on the concept that failure results from cyclic stress accumulation over the lifetime and that the damage accumulated during one cycle cumulates with the previous damage. [1]

A further fatigue model is the weighted Ah model, an evolution of the Wöhler method. Instead of cycles, the weighted Ah model focuses on the amount of electric charge throughput, which is a better metric for batteries. The damage occurring per Ah throughput is weighted based on various parameters. [1]

Fatigue models do not require that many cell parameters and, thus, are pretty easy to implement. Nevertheless, the two hypotheses that aging events do not influence each other and that damage always cumulates to existing damage, limit fatigue models' validity. [1]

2.2.3 Mathematical Models

Mathematical models set themselves apart from other battery aging models, as they use Machine Learning methods to predict battery aging. This chapter will only concentrate on Neural Networks since they are mainly used for mathematical models. A neural network can be considered as a black box, connecting input data to output data through a combination of nonlinear functions (see chapter 2.3). Thus, the model can predict battery aging by inputting data, such as current, voltage, and temperature profiles. [1]

Several publications are already employing Neural Networks to predict battery aging. Severson et al. [2] and Attia et al. [3] developed an early cycle life prediction based on Neural Networks. Both, Veeraraghavan et al. [4] and Choi et al. [5] predict the course of the SOH over cycles, whereas Lucu et al. [6] predict calendar aging under several conditions.

The greatest challenge with mathematical models is, getting a large and good enough dataset so that Neural Networks can detect patterns, generalizing on unseen data.

2.3 Neural Networks

Neural Networks are a Machine Learning method inspired by the way human brains work [19]. Like human brains, Neural Networks consist of many neurons that are connected to each other. For consistency, neurons in a Neural Network will be referred to as units from now on.

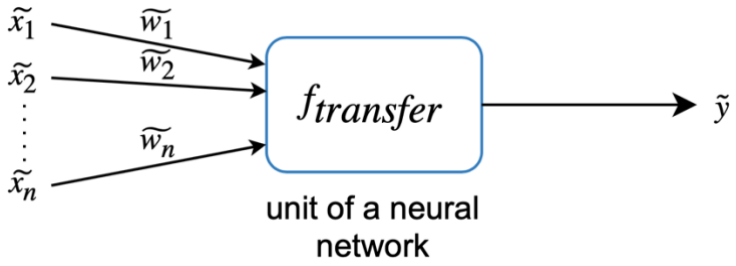

Figure 2.3 Basic model of a single unit

Figure 2.3 shows the model for a single unit, receiving input signals $\tilde{x_\iota}$ via connections to other units. The unit can be expressed as a mathematical function with the transfer function $f_{transfer}(x)$, processing the difference of the weighted input signals and the bias $\tilde{b}$. In most cases, the $f_{transfer}(x)$ carries out the following function: If the difference is greater or equal to 0, the unit should pass 1 to the next unit. If not, the unit should output 0. [19]

$$\tilde{y} = f_{transfer}\left(\left(\sum_{i=1}^{n} \tilde{w_\iota} \cdot \tilde{x_\iota}\right) - \tilde{b}\right) \tag{2.7}$$

This task can either be realized with the step function or approximated with the sigmoid function (see Figure 2.4).

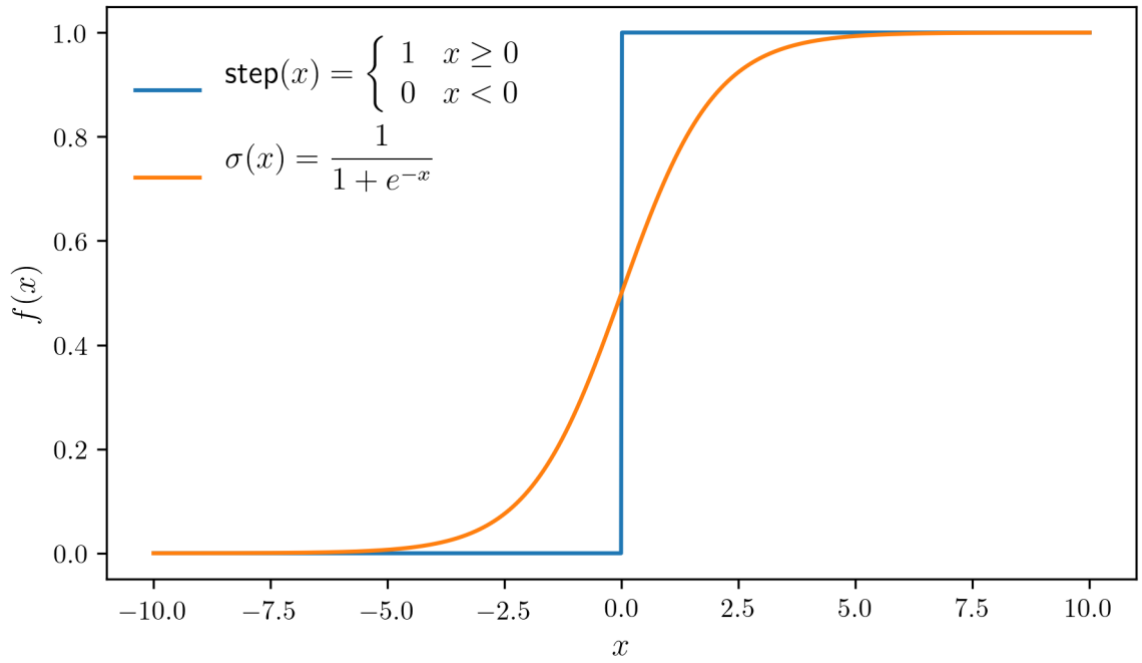

Figure 2.4 Step function and sigmoid function

Units of a neural network are structured in linear arrays, called layers. The layer's units are connected to the units of the successive layer. A Neural Network consists of one input layer, one or multiple hidden layers, and one output layer. The input layer receives the inputs of the Neural Network and passes them to the first hidden layer. Hidden layers further process the signals, detecting patterns. The final output layer gives out the Neural Network's result, called prediction. Designing the Neural Network's architecture involves defining the number of hidden layers and units. [19]

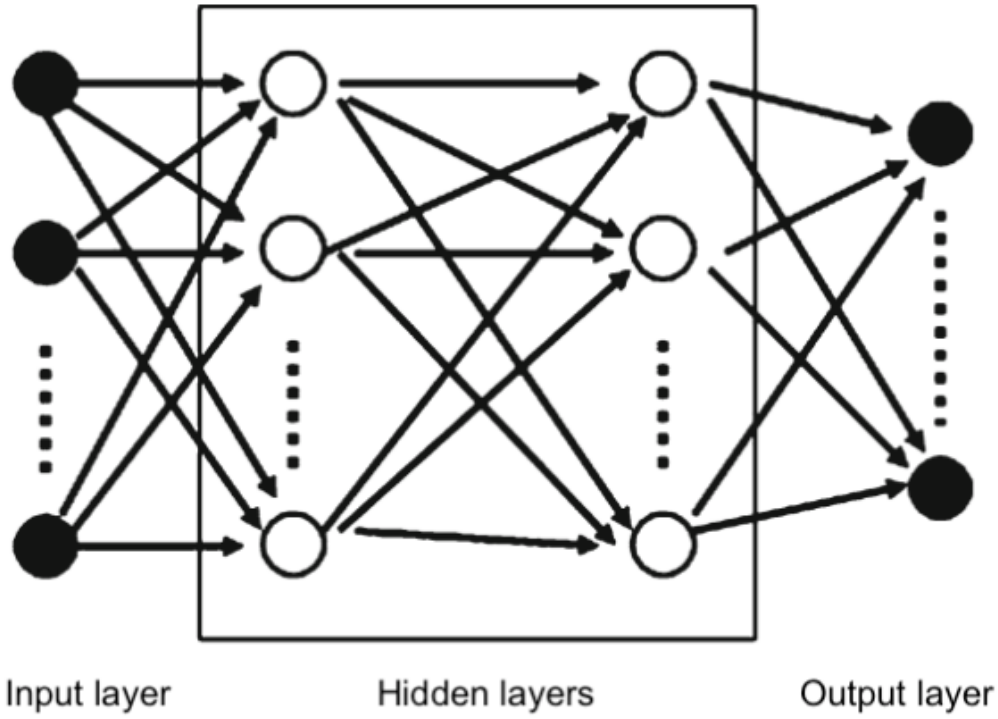

Figure 2.5 Structure of a Neural Networks **[20]**

Neural Networks improve their performance with a training process. During training, weights and biases are adjusted in a way that the error between the predicted and expected output is minimized. The metric for this error is the loss function. [19]

2.4 Long Short-Term Memory Networks

The Long Short-Term Memory network (LSTM) is a neural network especially designed for sequence problems. By adding the temporal dimension to Neural Networks, it can detect temporal patterns within sequences. This skill makes LSTMs extremely useful for many problems in Machine Learning. Particularly, the rise of voice assistants over the last five years can be attributed to LSTM's impressive results in speech recognition [21]. Further applications include image caption generation, text translation, handwriting recognition and generation, and sequence forecasting (for example, stock prices) [22]. LSTMs can also be used for video pattern recognition or classification, e.g., for autonomous driving [23].

2.4.1 LSTM unit

LSTM layers consist of multiple LSTM units. An LSTM unit can be thought of as a loop repeated for every time step of the input sequence. What makes LSTMs units so special is the ability to pass signals, including the unit's output $h(t)$ and the internal state $c(t)$, "laterally" to the successive time step. Therefore, LSTMs can memorize information from previous time steps.

On the left side of Figure 2.6, a single LSTM unit is illustrated, processing the unit's input $x(t)$ and then passing $c(t)$ and $h(t)$ to the following time step. On the right, the LSTM unit is depicted over a sequence of three input time steps (rolled out in time). There are two ways of how an LSTM unit processes the input, depending on its application. It can either give out an output every time step (the full output sequence $h(t)$) or only return an output at the sequence's end.

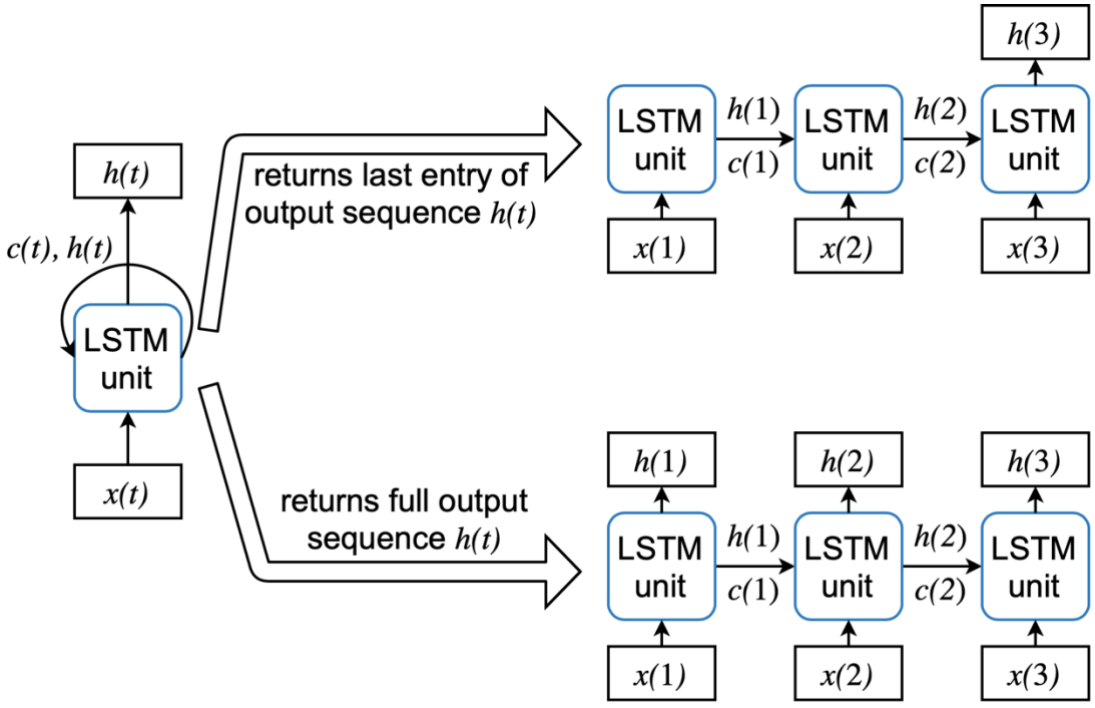

Figure 2.6 A single LSTM unit rolled out in time

The architecture of an LSTM unit is more complicated than the one of a basic neural network unit. Figure 2.7 shows its structure, containing a forget gate, an input gate, and an output gate.

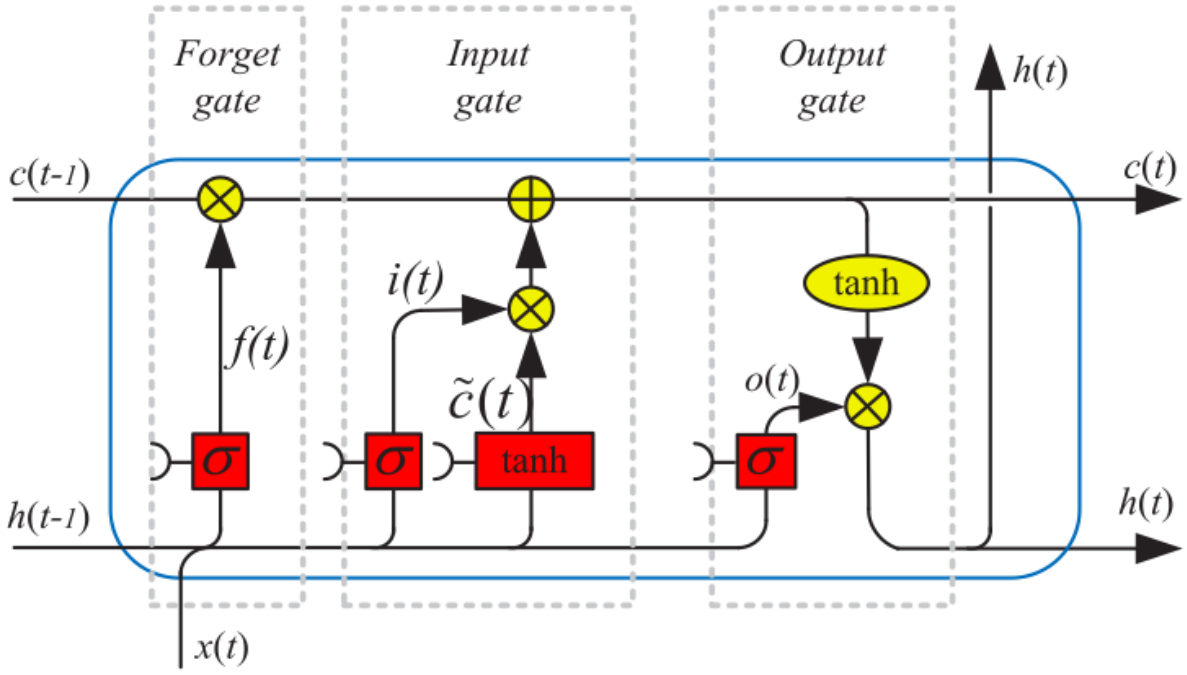

Figure 2.7 Architecture of an LSTM unit **[19]**

The unit receives $h(t-1)$ and $c(t-1)$ as recurrent signals from the previous time steps and passes $h(t)$ and $c(t)$ to the next time step. The forget gate $f(t)$, a sigmoid function in dependence upon the input time step vector $x(t)$, $h(t-1)$, weights, and biases, decides, whether the internal state should be reset.

$$f(t) = \sigma\left(W_{fh}h(t-1) + W_{fx}x(t-1) + b_f\right) \tag{2.8}$$

Updating the internal state is the task of the input gate. The input gate $i(t)$ and the internal state update $\tilde{c}(t)$ are calculated for that purpose.

$$i(t) = \sigma(W_{ih}h(t-1) + W_{ix}x(t-1) + b_i) \tag{2.9}$$

$$\tilde{c}(t) = \tanh(W_{\tilde{c}h}h(t-1) + W_{\tilde{c}x}x(t-1) + b_{\tilde{c}}) \tag{2.10}$$

Finally, the new internal state $c(t)$ is determined with $f(t)$, $\mathrm{c}(\mathrm{t}-1)$, $\mathrm{i}(\mathrm{t})$, and $\tilde{c}(t)$.

$$c(t) = f(t) \cdot \mathrm{c}(\mathrm{t}-1) + \mathrm{i}(\mathrm{t}) \cdot \tilde{c}(t) \tag{2.11}$$

The output gate is responsible for the output $h(t)$, that is calculated as a product of the output gate $o(t)$ and $\tanh\big(c(t)\big)$.

$$o(t) = \sigma(W_{oh}h(t-1) + W_{ox}x(t-1) + b_o) \tag{2.12}$$

$$h(t) = o(t) \cdot \tanh\big(c(t)\big) \tag{2.13}$$

2.4.2 Loss Function

The loss function is crucial for an LSTM model. The training algorithm uses the loss function to evaluate the model's current performance and tries to minimize it to improve the performance. It is a challenging task to combine all good and bad aspects of a complex model into a single number, which allows a comparison between different predicted outputs. [24]

The choice of a loss function Q depends on the problem type the LSTM faces. This chapter will only focus on regression problems (a real-value quantity is predicted). The two main loss functions for regression problems are mean squared error (MSE) and mean absolute error (MAE). [22]

MSE is defined as the mean of the squared error. n is the number of time steps, $y(t)$ is the predicted value at time step t, and $\hat{y}(t)$ is the expected value at time step t.

$$Q_{\mathrm{MSE}} = \frac{1}{n}\sum_{n}(\hat{y}(t) - y(t))^2 \tag{2.14}$$

MAE is defined as follows.

$$Q_{\mathrm{MAE}} = \frac{1}{n}\sum_{n}|\hat{y}(t) - y(t)| \tag{2.15}$$

Both performance indicators have advantages and disadvantages. Compared to MAE, MSE weights the larger errors way more, whereas MAE weights all errors the same. Thus, it is hard to compare different MSE results, because one large error is enough for the MSE to shoot up. That is why, an optimization of the MSE is mainly focused on large errors, often neglecting smaller errors. [25]

It can be shown: When the MAE is minimized, the prediction's mean tries to match the expected values' mean. However, as the MSE is minimized, the prediction's average targets the expected values' average. What does that mean? Optimization of the MAE will result in a bias ($\sum_n(\hat{y}(t) - y(t)) \neq 0$), whereas optimization of the MSE is unbiased. [25]

In summary, which loss function to use cannot be generalized and depends on the specific application.

2.4.3 Backpropagation Through Time

Backpropagation through time (BPTT) is the training algorithm used for LSTMs. It expands the standard backpropagation algorithm by including the time dimension.

The goal of training a model is to improve its prediction performance by minimizing the loss function. The loss $Q(w_1, \dots, w_m)$ is a function of the model's m trainable parameters w_i (weights and biases) with $i \in \mathbb{N},\ 1 \leq i \leq m$. To approximate the minimum of the multi-variable function $Q(w_1, \dots, w_m)$, the gradient descent, a common numeric approach, is used. The idea of gradient descent is to find a minimum of a multi-variable function by taking iterative steps into the negative gradient's direction from the current point on. The components of the loss function's gradient ∇Q are the partial derivatives of $Q(w_1, \dots, w_m)$.

$$\nabla Q = \begin{pmatrix} \frac{\partial Q}{\partial w_1} \\ \vdots \\ \frac{\partial Q}{\partial w_\mathrm{m}} \end{pmatrix} \tag{2.16}$$

In case of the most straightforward model – one input, one output, and one unit per layer with one trainable parameter – the partial derivative of $Q(w_1, \dots, w_n)$ with respect to w_i (i-th weight counted from the input layer) is calculated with the chain rule as follows.

$$\frac{\partial Q}{\partial w_i} = \frac{\partial Q}{\partial w_m} \frac{\partial w_m}{\partial w_{n-1}} \dots \frac{\partial w_{i+1}}{\partial w_i} \tag{2.17}$$

If the partial derivative of $Q(w_1, \dots, w_n)$ with respect to the following weight w_{i+1} is known, the calculation simplifies significantly.

$$\frac{\partial Q}{\partial w_i} = \frac{\partial Q}{\partial w_{i+1}} \frac{\partial w_{i+1}}{\partial w_i} \tag{2.18}$$

In a complex Neural Network with layers that consist of many trainable parameters, influencing all of the successive layers' trainable parameters, the calculation is not that easy. However, the partial derivative can still be expressed as a sum of those terms over all the successive layers' trainable parameters.

The standard backpropagation algorithm is a way to calculate these partial derivatives by employing the chain rule. The first step is to propagate a training input through the network forward. With the predicted output and the expected output from the training data, the loss function computes the error. Next, the algorithm propagates through the network backward and calculates the partial derivatives $\frac{\partial Q}{\partial w_i}$, using the partial derivatives of the successive layer's trainable parameters. Now that ∇Q is known, the trainable parameters are adjusted with the optimization algorithm based on the

gradient descent (see chapter 2.4.4), and the backpropagation algorithm is repeated. [22]

BPTT is the application of the backpropagation algorithm for LSTMs. By unrolling the LSTM over all input time steps, the network can be considered one big deep Neural Network – deep in time [26]. Backpropagation can now be carried out on that network, as described earlier. [22]

The computational expense of BPTT is strongly dependent upon the number of input time steps because they determine the number of derivatives required for a single update of the trainable parameters. Many input time steps can cause the trainable parameters to go to zero or explode, negatively impacting the model's ability to learn a problem. [22]

2.4.4 Optimization Algorithm

The optimization algorithm or optimizer defines the way the trainable parameters w (includes weights and biases) are adjusted iteratively.

$$w = \begin{pmatrix} w_1 \\ \vdots \\ w_m \end{pmatrix} \tag{2.19}$$

Every optimizer used for training LSTMs is based on the gradient descent, adjusting w in the negative direction of the loss function's negative gradient ∇Q that is determined with BPTT. The learning rate η sets the adjustment's step size.

$$w := w - \eta \nabla Q \tag{2.20}$$

The main difference between various optimizers is the learning rate. Setting it too high can cause the model to diverge, and setting it too low slows down convergence [24]. Besides the classic stochastic gradient descent with a fixed learning rate, multiple extensions have an adaptive learning rate (see chapter 3.3.2).

2.4.5 Overfitting and Dropout

A common problem with Neural Networks, especially deep LSTMs, is overfitting. An overfit model performs well on the training dataset and continues to improve. At the same time, the performance on the validation dataset only improves up to a certain point and deteriorates from then on (see Figure 2.8). That happens because, in a vast network, some units rely on other units to fix their mistakes. Therefore, the model co-adapts to the training dataset. Those co-adaptations do not generalize on unseen data, leading to worse performance on the validation dataset. [27]

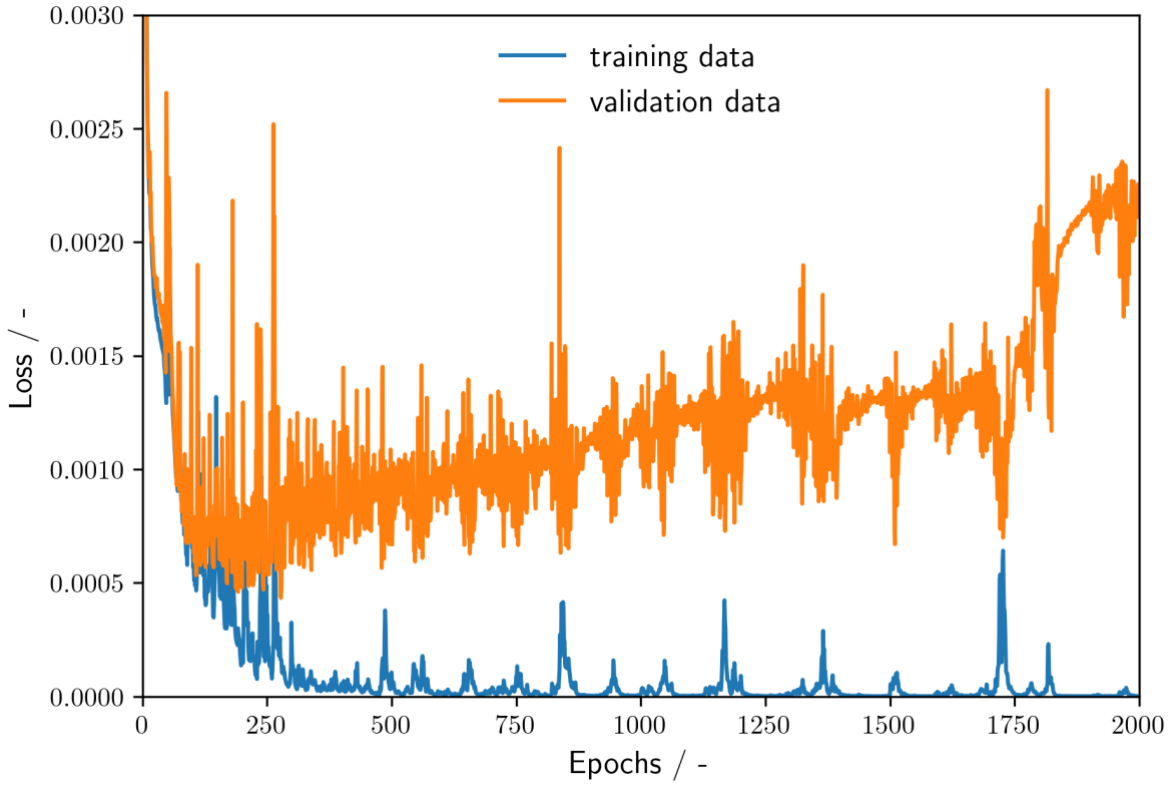

Figure 2.8 An extreme example of overfitting. Epochs are the number of passes through the samples of the training set.

The core concept of dropout is that random units, along with their connections in the network, are dropped out during training. That means they are temporally removed from the network, therefore, not affecting the output (see Figure 2.9). Every unit in a Neural Network trained with dropout learns to deal with a random sample of other units,

no longer relying on other units to fix up their own mistakes, making the model more robust to unseen data. [27]

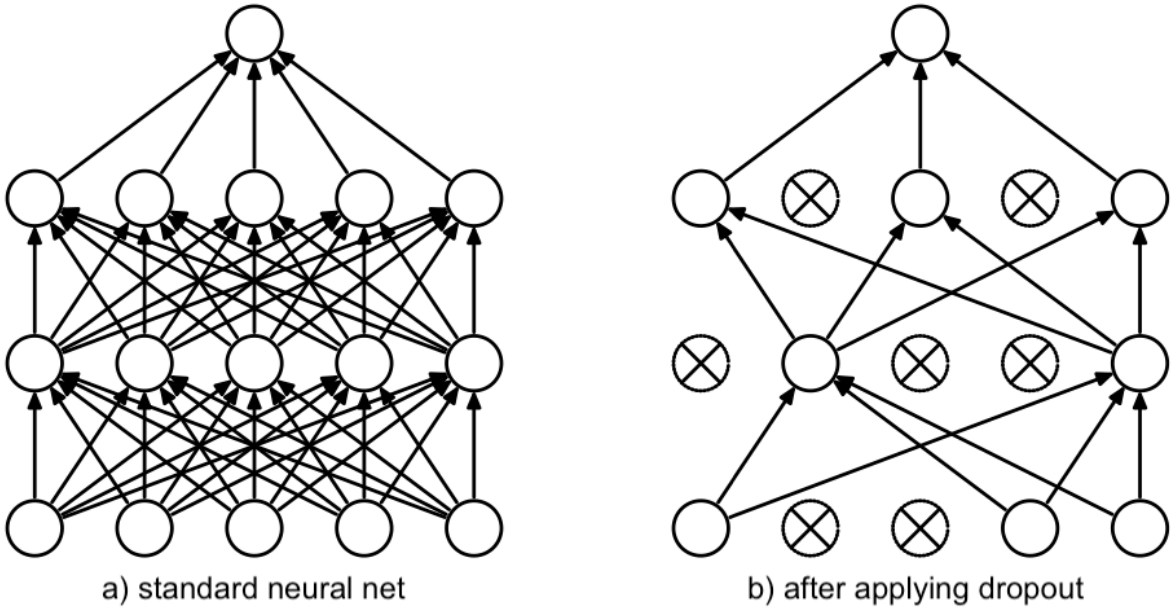

Figure 2.9 Dropout Neural Network model where dropout is applied to every layer **[27]**

3 Methods and Results

This chapter deals with the development process of the data-based battery aging model. The model's input contains the charging profiles current, voltage, and temperature over a particular number of cycles. It predicts the decrease in capacity over those cycles, using a Machine Learning approach. As this is a sequence-to-sequence prediction problem, an LSTM is employed for the model by implementing the Keras Machine Learning library in Python.

3.1 Data

The dataset used to train and validate the model, including 174 LFP cells cycled under varying fast-charging conditions, comprises publicly available datasets from two papers. Both publications, Severson et al. [2] and Attia et al. [3], use the data to develop an early cycle life prediction model based on Machine Learning.

The LFP cells, manufactured by A123 Systems (APR18650M1A), are identical in both datasets. They were cycled in a forced convection temperature chamber, with the temperature set to 30°C. The battery cells have a nominal voltage of 3,3 V and a nominal capacity of 1,1 Ah.

All cells in the dataset were charged with a CC-CV charging procedure. However, during the current regulation phase (until 80% SOC), there is either a one-, two-, or four-step fast-charging policy in place. Thus, the constant current switches in that phase. Figure 3.1 visualizes the three different fast-charging policies. All cells were discharged at 4C.

The cells were cycled to failure and, in some cases, even beyond, with the number of cycles varying between 200 to more than 2000. Throughout each cycle, current, voltage, temperature, and time were measured in intervals of 3 s - 4 s. Additionally, the cell's internal resistance and the charged and discharged capacity are measured for each cycle.

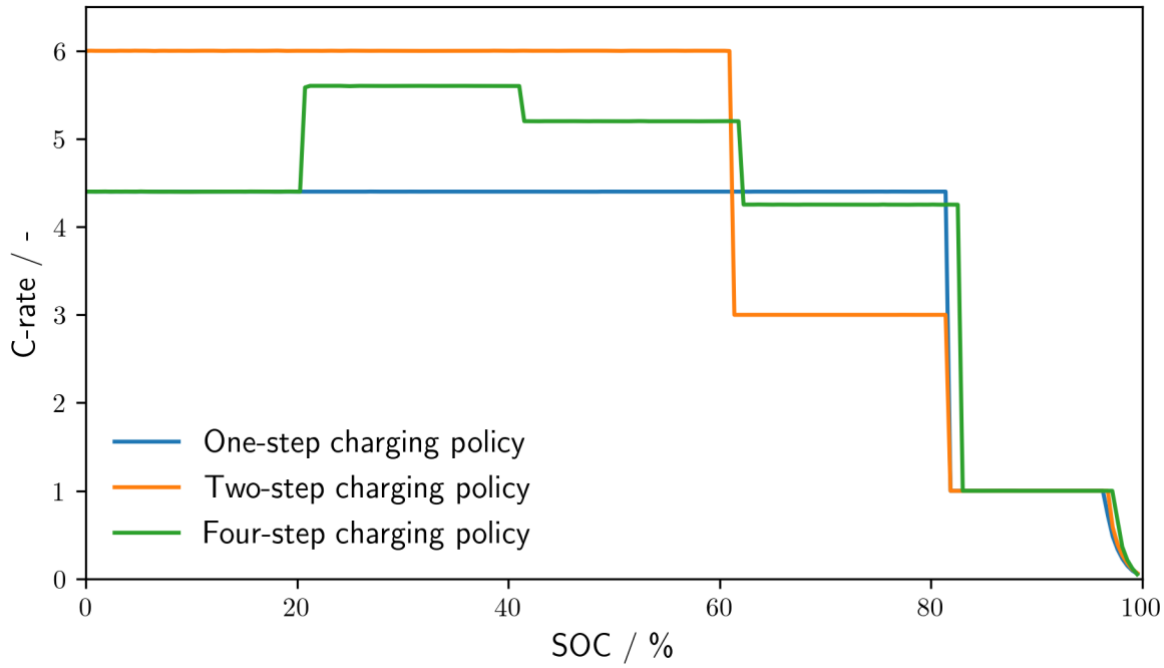

Figure 3.1 Examples of different fast charging policies

3.2 Data Preparation

In order to get good results from the LSTM model, the input and output data needs to be prepared beforehand.

Generally, the LSTM expects the data to have three dimensions: samples, time steps, and features. The number of time steps should be limited to 200-400 for the BPTT to work correctly, making the decision on how to shape and split the data crucial [22].

To evaluate the model's performance, the initial dataset must be split into a training and validation set. The training set is the data employed to train the model. In contrast, the validation set is not used for training and therefore provides an unbiased evaluation of the model's performance on unseen data. Here, the training set contains 80 % of the samples, and 20 % of the samples are used for validation.

3.2.1 Number of Cycles

Figure 3.2 shows the distribution of cycles run for all samples in the raw dataset. As there are only eight cells cycled for more than 1500 cycles and 24 for more than 1070 cycles, the model would not have enough training data to sufficiently learn the problem for cycles greater than 1070 cycles. Therefore, all samples are limited to a maximum of 1070 cycles.

Roughly half of the raw dataset's battery cells were cycled to a SOH of 80 %, which is the threshold to determine the cycle life in most technical applications. The other 85 cells were cycled beyond their cycle life. As the cell's behavior beyond the cycle life is irrelevant for this application, these samples were cut to 80 % SOH. Figure 3.2 shows the distribution of cycles in the adjusted dataset and compares it to the raw dataset.

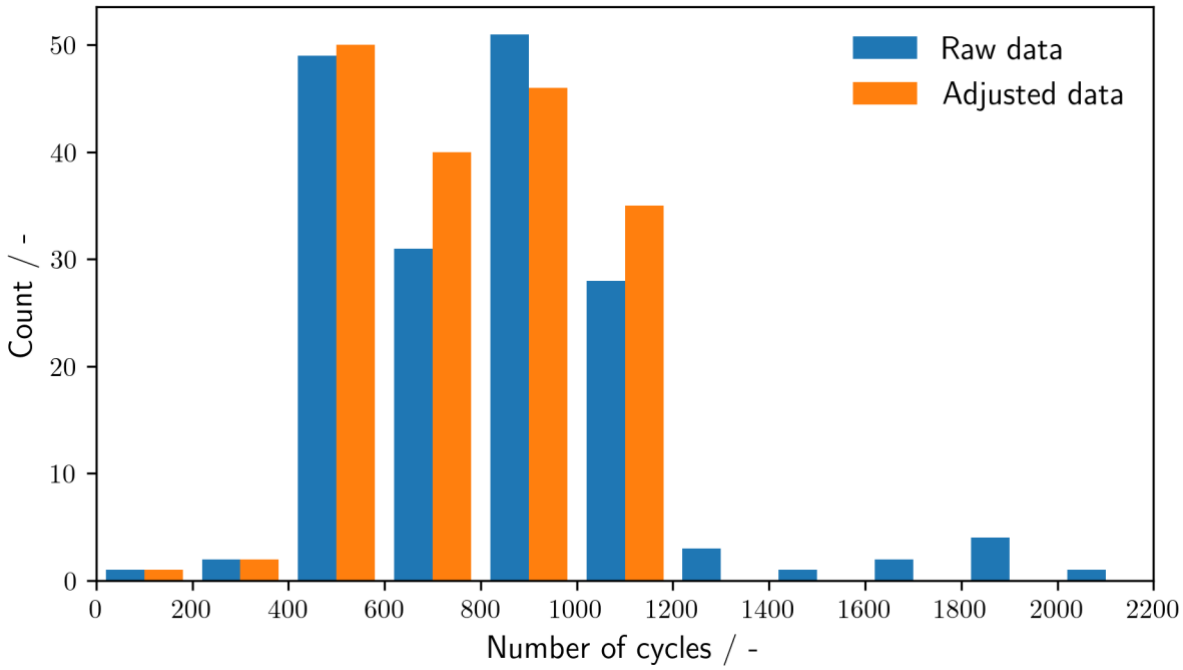

Figure 3.2 Number of cycles in the raw dataset compared to the adjusted dataset

3.2.2 Input Data

The input data for the LSTM model consists of the cell's current, voltage, and temperature profiles. The initial dataset includes the profiles for charging and for

constant discharging of 174 battery cells. For each cycle, the measurement period for charging is about 30 min, containing roughly 800 measurement points. Since the discharging profiles are not relevant for the model that focuses on fast charging, only the charging profiles are used for the input data.

Keras expects a vectorized representation of the data, requiring each sequence to have the same length. However, in the utilized dataset, every cycle's measurement file has a different length due to the varying charging and measurement conditions. Thus, the sequences are padded with a specific masking value at the end, adjusting every sequence to the same length. To notify the model that parts of the input data are padded and should be ignored, the input data is masked in the LSTM (see chapter 3.3.1). One time step in the input data set equals one minute in the measurement file. Even though the longest sequence only lasts 40 minutes, the number of time steps is set to 120, allowing the model to handle longer sequences in a possible application.

The whole dataset includes 68 cycles with data anomalies, where the charging process was suddenly interrupted (see Figure 3.3). Those cycles were masked so that they do not impact the model.

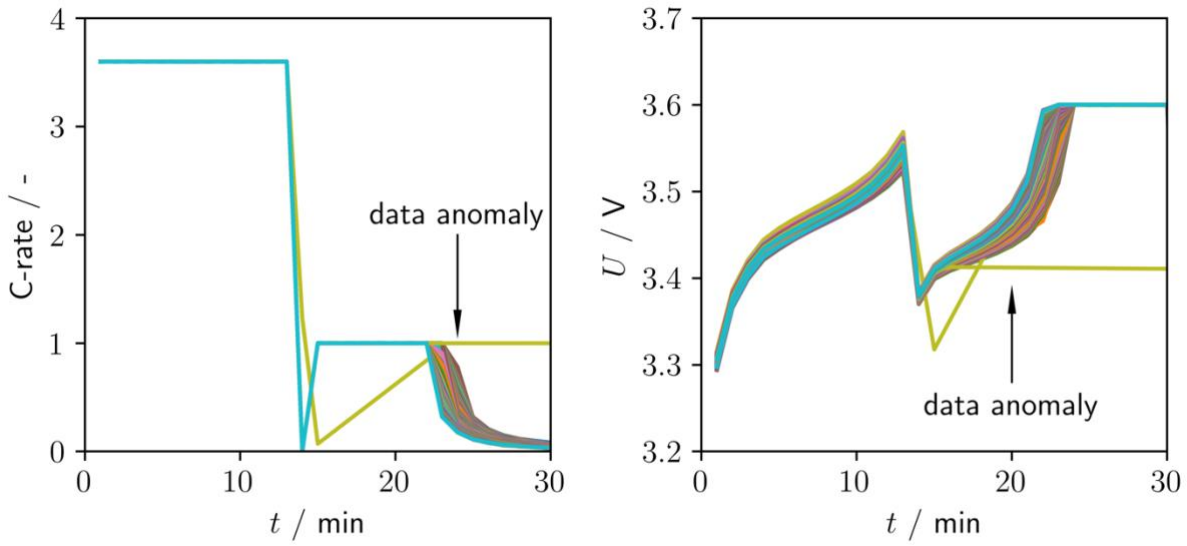

Figure 3.3 Data anomaly in the current and voltage profiles

Additionally, the input data needs to be scaled when training an LSTM. Unscaled data slows down the learning and convergence of the model and, in many cases, even prevents the model from effectively learning the problem. Normalization and

standardization are mostly used for scaling. Normalization rescales the data so that all values are within the range of 0 and 1, expecting knowledge of the minimum and maximum observable values. On the other hand, standardization rescales the distribution of values so that the mean is 0 and the variance is 1, requiring knowledge of the mean and the variance. This can be thought of as centering the data. Estimates of the dataset's mean and variance are often more robust to new data than the minimum and maximum values. [22]

The type of scaling for the input data must be well thought of. It must be applicable to new data with different boundary conditions that use the scaling parameters as this input dataset.

The initial current profile ranges from 0 C (no charging) up to 8 C (maximum charging current). As the minimum value already lies conveniently, the profile is normalized so that all values are within the range of 0 and 1. The range of voltage profiles depends on the type of cathode the cell contains. Since the model also has to fit other lithium-ion batteries besides LFP cells, the current is first normalized for every cathode type. Subsequently, the current profile is standardized with the scaling parameters of the original current profile. Temperatures range from 25 °C up to 38 °C. It is difficult for the model to learn from such values. Therefore, the temperature profile is standardized. Figure 3.4 shows the original current, voltage, and temperature profiles compared to the scaled ones for one representative cell.

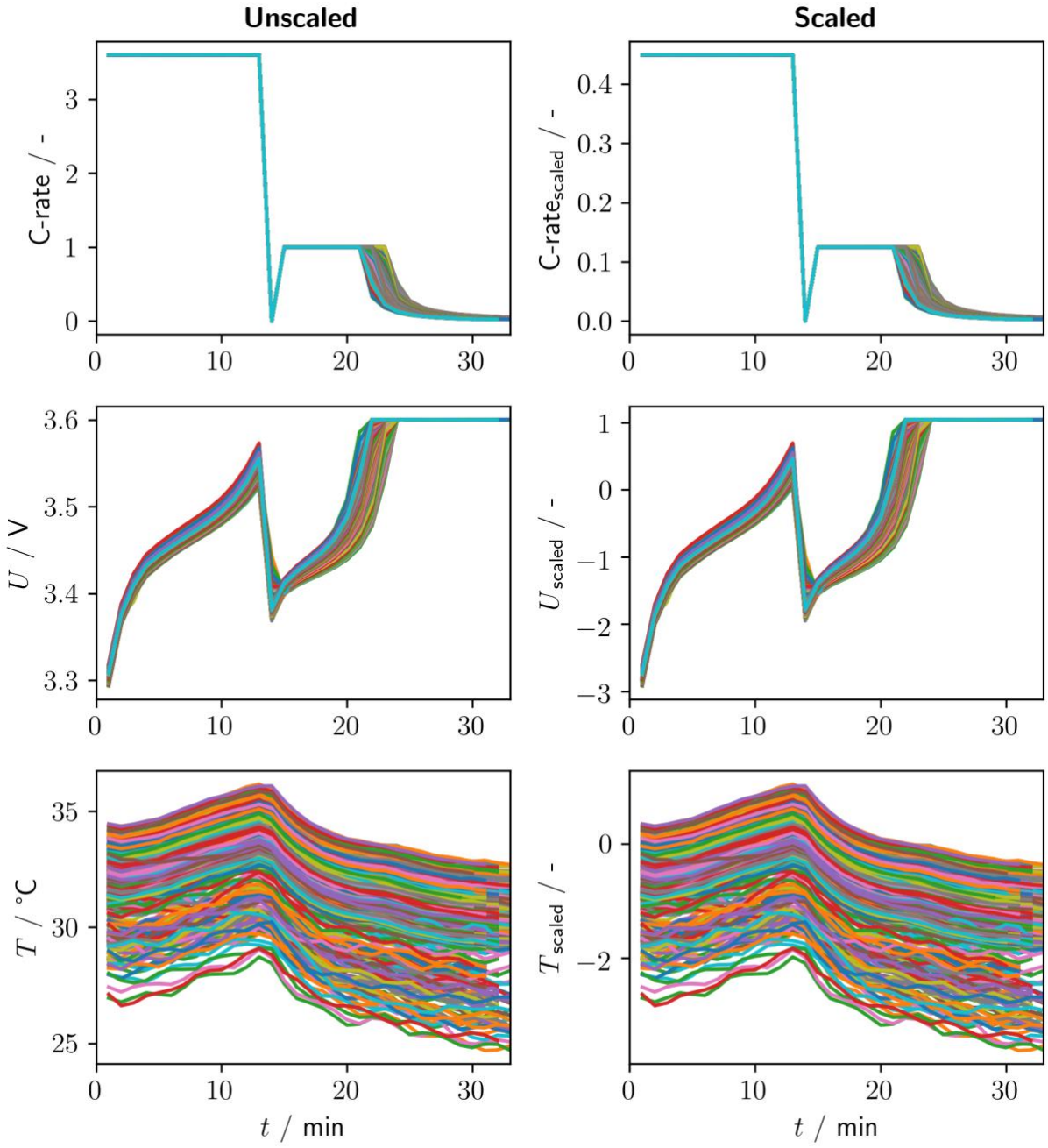

Figure 3.4 The figure shows current, voltage, and temperature profiles of a representative sample for all 1070 cycles. On the left side: unscaled date. On the right side: scaled data.

As previously mentioned, Keras expects the data to have the three dimensions samples, time steps, and features. Here, however, the input data is four-dimensional, with the number of battery cells, the measured sequences per cycle, the cycles, and the features (current, voltage, and temperature) as dimensions. Therefore, the cycles are seen as additional features, bringing the number of features to $1070 \cdot 3 = 3210$ in

total. For the sample j and the time step t, the features are arranged as follows (n is the number of cycles, x is the model's input data).

$$x(j,t,:) = \begin{pmatrix} \mathrm{C-rate}(j,t,1) \\ U(j,t,1) \\ T(j,t,1) \\ \vdots \\ \mathrm{C-rate}(j,t,n) \\ U(j,t,n) \\ T(j,t,n) \end{pmatrix} \tag{3.1}$$

3.2.3 Output Data

The model's output data consists of the SOH profile over the number of cycles. However, the initial dataset does not include the SOH but the capacity for every cycle. Thus, the capacity is divided by the maximum capacity to get the SOH.

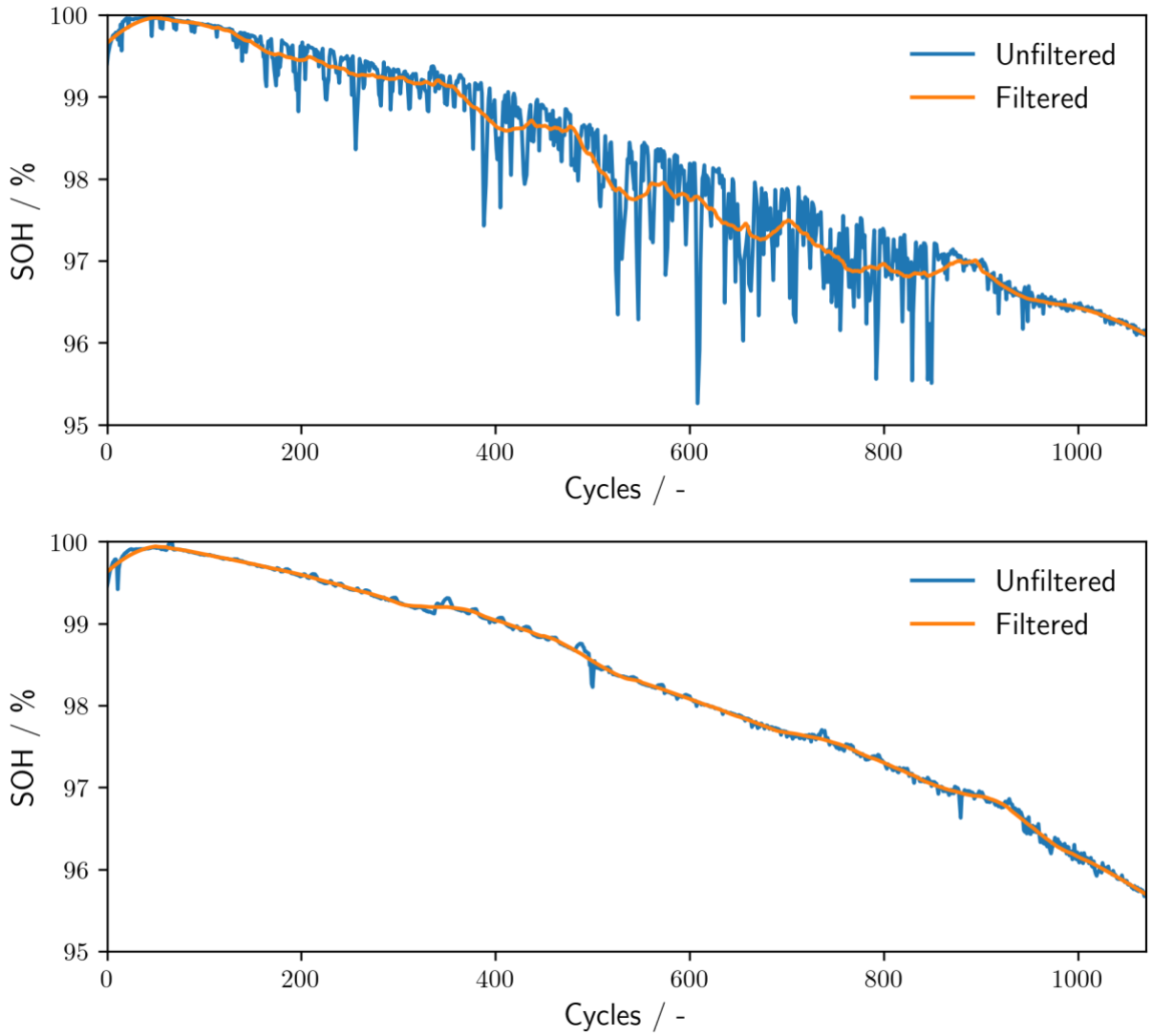

Figure 3.5 Comparison of the unfiltered and filtered SOH profiles

Since the original measurements contained some outliers and were noisy at times, the data is filtered with a Savitzky–Golay filter. The data is fit with a polynomial of third degree and a filter window length of 101. Figure 3.5 visualizes the unfiltered and filtered SOH profiles for two noisy samples, and Figure 3.6 shows the final SOH profiles for all samples.

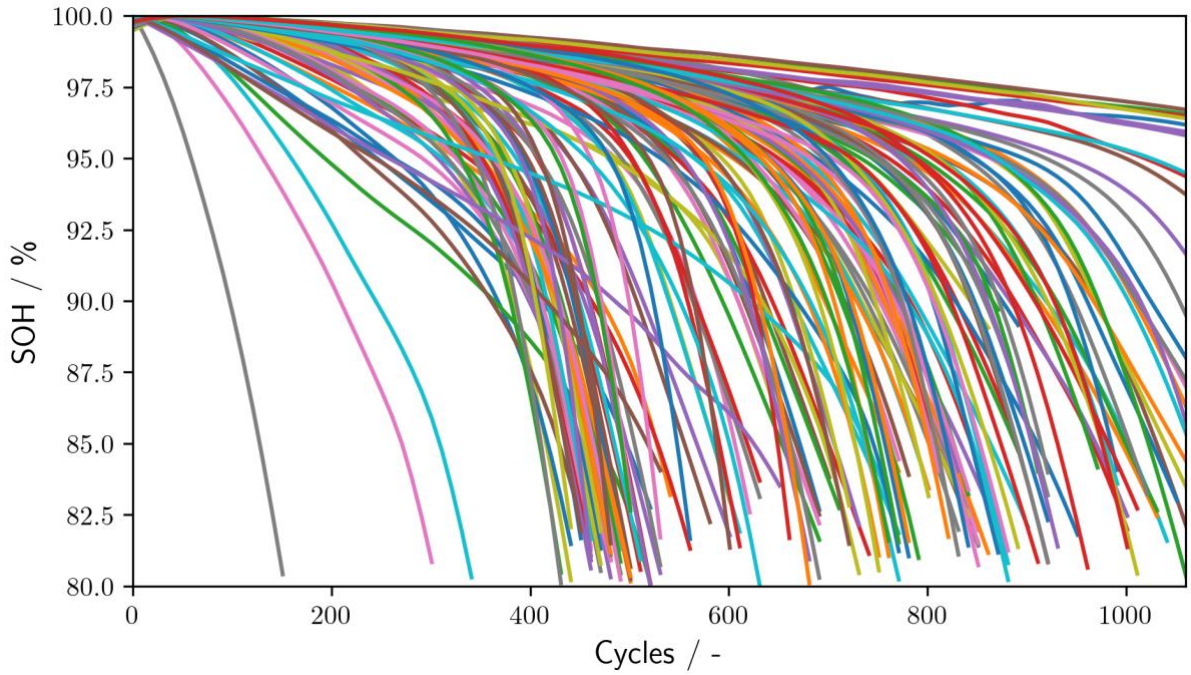

Figure 3.6 Final SOH profiles for all samples

The output data for the model must have the shape: samples, time steps, and features. The samples include the 174 battery cells, the cycles represent the time steps, and the only feature of the output data is the SOH. As the number of cycles is larger than 200-400, the cycles were scaled so that only every tenth cycle is one time step. Hence, the output time steps are limited to 107. Equally to the input sequence, the output sequences vary in length and are padded at the end.

3.3 LSTM model

For the LSTM model to predict battery cell aging, Keras deep learning library in Python is used [28]. Keras contains implementations of the most important building blocks for Neural Networks, e.g., layers, loss functions, activation functions, and optimizers. Apart from standard Neural Networks, Keras also supports recurrent Neural Networks such as LSTMs.

This chapter will describe the architecture of the LSTM model in dependence on hyperparameters. The hyperparameters have a remarkable influence on the model's

performance in terms of accuracy and learning speed. Chapter 3.4 will focus on the individual hyperparameters in detail.

As this is a sequence-to-sequence prediction problem where the number of input and output time steps vary, an Encoder-Decoder architecture proven very effective for this problem is used. This approach comprises the Encoder and the Decoder. The Encoder maps a two-dimensional input sequence with the shape (input time steps, input features) to a fixed-length vector. The Decoder maps the vector representation of the input to the two-dimensional output sequence (output time steps, output features) (see Figure 3.7) [29].

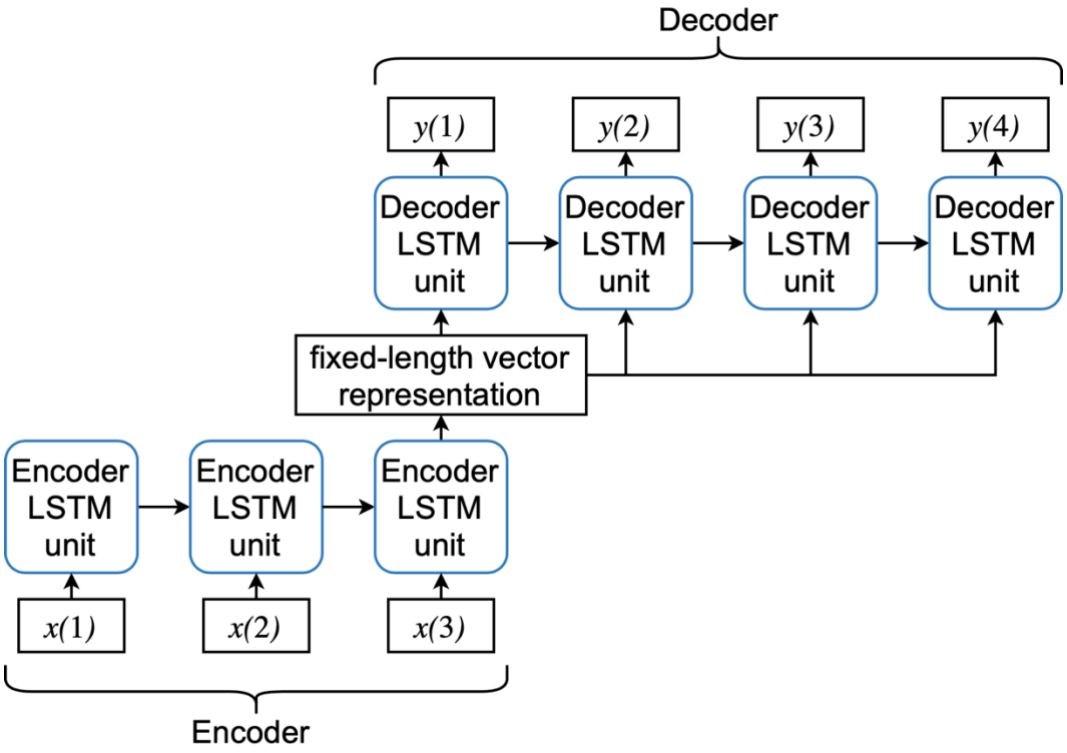

Figure 3.7 Exemplary Encoder-Decoder architecture rolled out in time with three input time steps and four output time steps. The fixed-length vector results from multiple Encoder LSTM units (they are not depicted in this figure).

3.3.1 Model Architecture with Keras

The model itself is an object from the Keras Sequential class that groups a linear stack of layers into a model. With the add method, a layer can be added to the model. The

first layer in the model is a Masking layer, skipping the inputs masked with the mask value. The layer expects two arguments. The input_shape argument defines the input time steps and features, and the mask_value argument sets the input value that is being ignored by the model.

The Encoder, consisting of multiple LSTM layers, succeeds the masking layer. The model gains depth by stacking multiple hidden layers, making it more accurate in predicting complex problems. The additional hidden LSTM layers combine the learned representation from previous layers and create new representations at a higher level of abstraction [22].

In Keras, each LSTM unit requires a two-dimensional input sequence (time steps, input features). When processing a two-dimensional input sequence, each unit will give out only the output from the final time step by default. The combined output is a vector with the number of units as its length. To stack LSTM layers in Keras, the layers must provide a two-dimensional output sequence for the following LSTM layer. By setting the return_sequences argument on the LSTM layer to True (defaults to False), each unit outputs a full sequence (every time step one output). Thus, the combined output sequence is two-dimensional (time steps, number of units) and can be processed by the successive layer. [22]

The number of stacked LSTM layers in the Encoder is a hyperparameter of the model. Every layer apart from the last returns a two-dimensional output, with the argument return_sequences set to True. The last layer returns a vector representation of the input. In most models using an Encoder-Decoder architecture, the number of units per layer declines through the Encoder [30]. This approach is also used for the LSTM model here. The number of units in the first LSTM layer, a power of two, is a hyperparameter. The number of units halves at each layer. For 128 units in the first layer, e.g., the units would decline as follows: 64, 32, 16, 8, …

The RepeatVector layer serves as a bridge between the Encoder and the Decoder, repeating the vector representation to give out a two-dimensional output for the first Decoder layer. The repetition factor n sets the length of the output sequence that will not be changed through the Decoder. Hence, n is set to the number of output time steps.

The first part of the Decoder is another stacked LSTM. It has the same structure as the Encoder in terms of layers and units, just in the opposite direction. So, the number of units doubles at every layer until it reaches the hyperparameter units at the final layer. The only difference is that the Decoder LSTM gives out a two-dimensional output, and, hence, the return_sequences argument on the last LSTM layer is set to True as well.

To prevent the model from overfitting, the Decoder contains a Dropout layer after the stacked LSTM. Keras offers a Dropout layer that randomly sets inputs to 0 at a given dropout rate during model training. The other inputs are scaled up by 1 / (1 – dropout rate) so that the sum over all inputs does not change. In this model, the Dropout layer is located before the model's final layer, as proposed in the original paper from Hinton et al. [31]. The layer's dropout rate argument is a hyperparameter of this model.

The final layer of the Decoder is an output layer. It contains a TimeDistributed wrapper, applying a Dense layer to every time step of the input. This combination is often used for outputting sequence predictions. Dense is a fully-connected layer, i.e., every unit processes the full input. Each unit computes the equation $\text{output} = \text{activation}(\text{input vector} \cdot \text{weight vector} + \text{bias})$, where $\text{activation}(\ )$ is the activation function hyperparameter passed as an argument. The number of units determines the number of output features and, therefore, is set to one.

Figure 3.8 visualizes the architecture of the LSTM model.

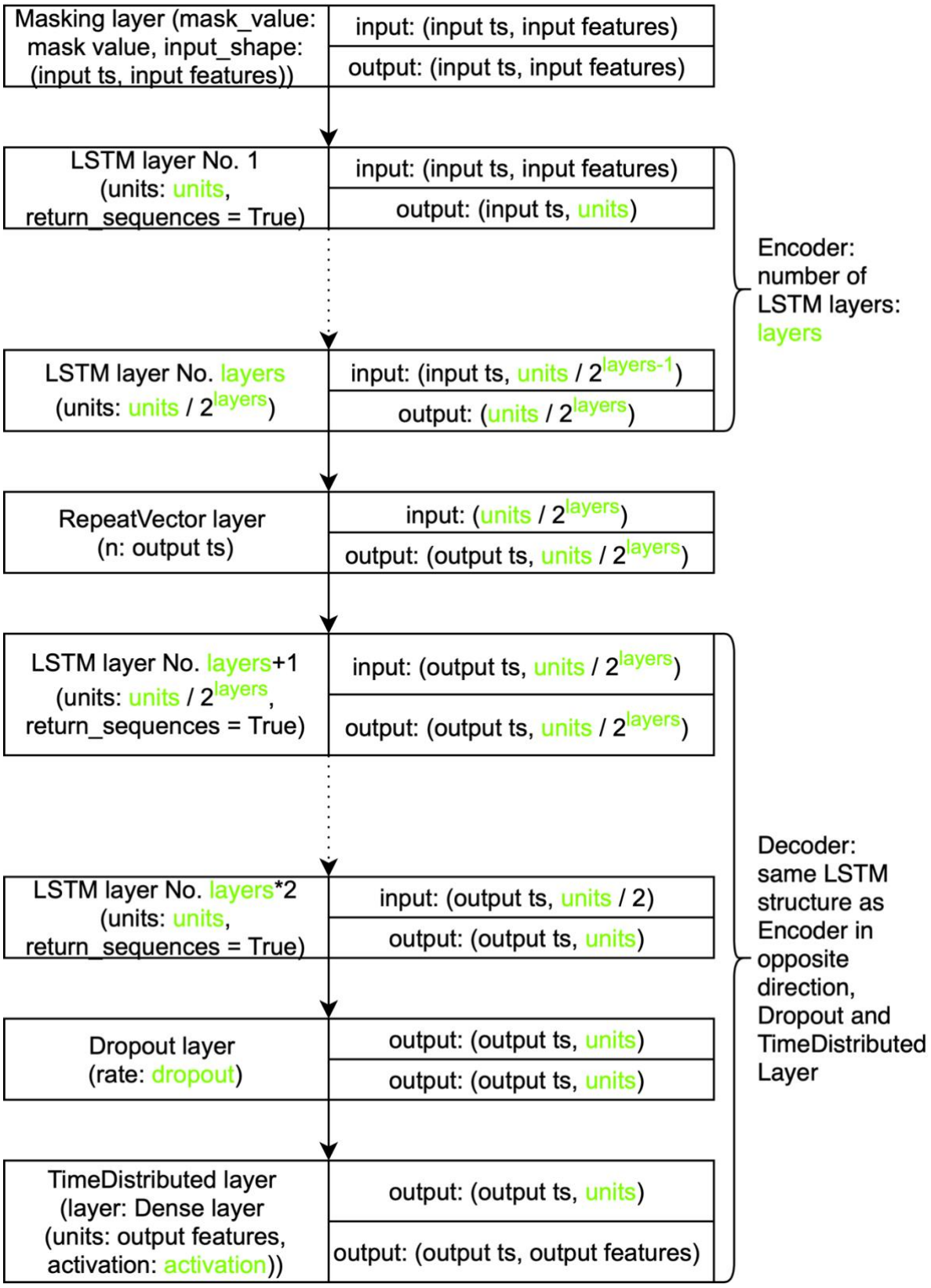

Figure 3.8 The flow chart shows the architecture of the LSTM model in Keras. The model hyperparameters are marked in green. Left: layer and given arguments. Right: shape of the input and output.

3.3.2 Model Compilation

The Compilation transforms the stack of defined layers into a highly efficient series of matrix multiplications that can be processed by the CPU or GPU. The compile method, applied to the model, requires a number of parameters to be specified. Specifically, the optimization algorithm to train the model, the loss function to evaluate the model, and the type of sample weighting applied to outputs. [22]

Optimization algorithms for LSTMs are all based on the gradient descent. The learning rate η differs for the various optimization algorithms. The original stochastic gradient descent (SGD) comes with a fixed learning rate that is pretty small, making it very slow to converge. Several extensions are based on an adaptive learning rate, causing large weight changes in the first iterations, whereas later iterations only fine-tune. Adaptive gradient algorithm (AdaGrad), root mean square propagation (RMSProp), and adaptive moment estimation (Adam) are the most common adaptive optimizers. Adam has proven to outperform the existing optimizers [32] and is therefore used for this model.

As previously mentioned, MSE and MAE are the only loss functions employed for regression problems. Here, MAE is chosen to optimize the model, as MSE mainly focused on samples with large errors neglected the remaining samples in own trials.

Since the output sequence is padded, the error for the padded time steps may not affect the total loss of the prediction. This goal can be achieved with temporal sample weighting. Thus, the sample_weight_mode argument is set to “temporal”, allowing it to establish individual loss weights for every time step during training.

3.3.3 Model Training

The model can be trained with the fit method, adapting the trainable parameters to minimize the loss on the training set. BPTT is used to determine the gradient of the

trainable parameters. Subsequently, the model is optimized according to the optimization algorithm. The fit method requires several parameters to be set.

X and y expect the training set of input and output data. Additionally, the validation set used to show the validation loss after every epoch is given with the validation_data parameter.

The batch_size parameter defines the number of samples, after which the trainable parameters are updated. This subset of samples is called a batch. The batch size is a hyperparameter for this model.

With the epochs parameter, the number of passes through all training set samples is settled, determining how long the model is trained. After the number of epochs, the model should have converged towards the optimal result. The choice of epochs depends on the complexity of the dataset. It is set to 2000 epochs, as this number has worked well in own trials.

Since the sample_weight_mode was set to “temporal” in the model compilation, the model expects a two-dimensional sample weight array as a parameter in the fit method. The weights for the sample’s padded time steps are 0, whereas the remaining weights of the sample are scaled up so that the sum of all weights equal the number of output time steps.

3.4 Hyperparameters

This chapter will describe the model’s hyperparameters in detail and sets the range in which the hyperparameters will be optimized in the next step.

Number of LSTM layers:

This hyperparameter describes the number of stacked layers in the Encoder and Decoder. It is a crucial hyperparameter, as Graves et al. [26] and Mohamed et al. [33] found that the depth of the LSTM is more important than the number of LSTM units in a given layer in terms of model accuracy. Nonetheless, too many stacked LSTM layers can cause overfitting and slow down learning. The ideal number of layers depends on the complexity of the problem and cannot be generalized. Brownlee [22] suggests 2-4

stacked layers. Here, the hyperparameter's value range for optimization is settled to 2, 3, and 4, i.e., 4, 6, and 8 LSTM layers in total (Encoder and Decoder).

Number of LSTM Units in the first Encoder layer:

Since the number of LSTM layer halves every layer, the number of LSTM units must be a power of two. As in the case of the number of layers, there is no ideal value that works for every model. In many cases, the number of units lies between 60 and 120 [22]. For the hyperparameter optimization, the three values 64, 128, and 256, are chosen for the first LSTM layer.

Dropout rate in the Dropout layer:

The dropout rate defines the rate at which inputs are set to zero to prevent overfitting. In the original paper that first applied dropout to Neural Networks, the authors found a dropout rate of 0.5 is close to optimal for many applications [27]. A dropout rate of 0.4 has worked pretty sufficiently as well in own trails. Therefore, these two values are used for optimization.

Activation function in the Dense layer:

The activation function is the function the final model layer uses to map the inputs to an output $(\text{output} = \text{activation}(\text{input vector} \cdot \text{weight vector} + \text{bias}))$. It is crucial because the function determines the format of the outputs. For regression problems, mainly two functions are chosen: the linear activation function and the rectified linear unit (ReLU) (see Figure 3.9). Glorot et al. [34] have found a better training performance for the ReLU function due to its nonlinearity. However, the nonlinear activation function's impact in a deep LSTM with several nonlinear functions might be limited.

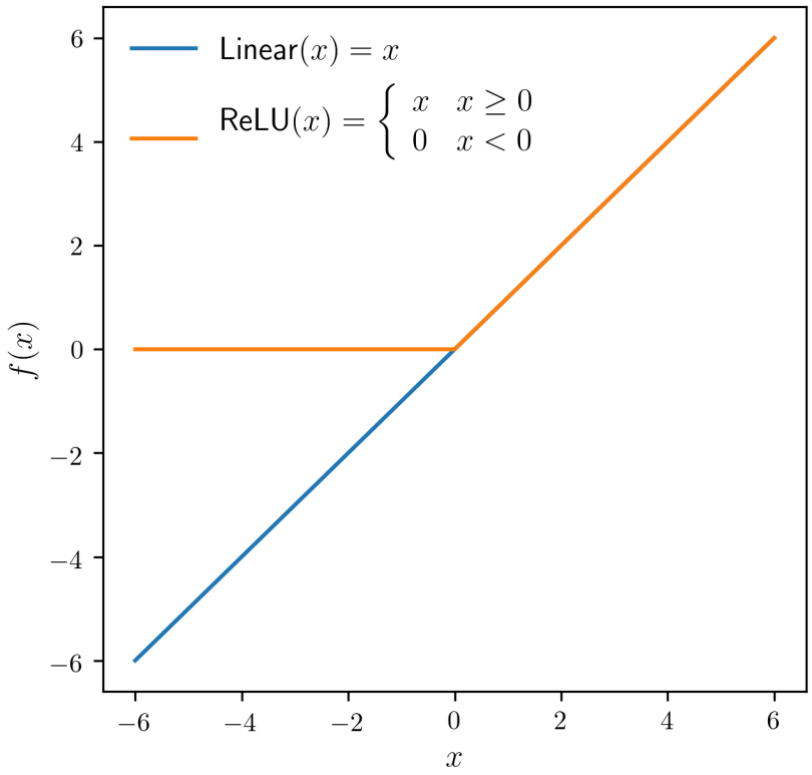

Figure 3.9 Comparison between the linear and ReLU activation function

Batch size:

The batch size sets the number of samples per trainable parameter update. A larger batch size decreases the training time. Common values for the batch size are 32, 64, and 128 [22]. Here 32 and 64 are used to optimize the model.

3.4.1 Grid Search

The model's hyperparameters were tuned with a grid search, training the model with all possible hyperparameter combinations. As two hyperparameters can assume three different values, and the other three can assume two values, this brings the total number of combinations to 72.

Comparing the model's performance with different hyperparameters is problematic since a training process is random and, therefore, not reproducible. Neural Networks use randomness on purpose to ensure that the training process converges to the global minimum of the loss function. LSTMs utilize randomness in the initialization of trainable

parameters, for the dropout, and in the optimization function. There are two main solutions to this problem. The first solution is to run the training so often (more than 30 times) that the model's performance can be evaluated with statistics. This approach is not applicable in this case, as it already takes roughly three hours to run the training for only one model. Hence, the second approach is employed here. By setting the random number generator to a fixed seed at the beginning of training, the results become reproducible and can be compared between different hyperparameter configurations.

3.4.2 Results

The grid search's results are evaluated with the validation dataset. Each one of the 72 models performs surprisingly well on the validation dataset. The validation MAEs range from 0.33 % up to 0.65 %. Hereafter, the results of the best three models are shown in more detail.

Table 3.1 Comparison between the best three models

Model	Hyperparameter					$Q_{\mathrm{MAE\ validation}}$ / %
	Layers	Units	Dropout	Activation	Batch size	
A	3	64	0.5	linear	64	0.3288
B	3	64	0.5	linear	32	0.3293
C	4	64	0.4	linear	32	0.3428

Table 3.1 compares the hyperparameter configurations and the MAE on the validation set between those models. What strikes out, is that 64 LSTM units in the first Encoder layer seem to work best. Additionally, the linear activation function has outperformed the ReLU activation function every time and, thus, the best ten models all have a linear activation function.

Figure 3.10 shows the course of the MAE on the validation set $Q_{\mathrm{MAE\ validation}}$ over the training epochs. The models B and C with the batch size of 64 converge earlier than model A with a batch size of 32. However, this effect is compensated by less training

time due to less model updates for a larger batch size. All models, though, have already converged at epoch 1100. The number of epochs, in hindsight, was chosen a little too large.

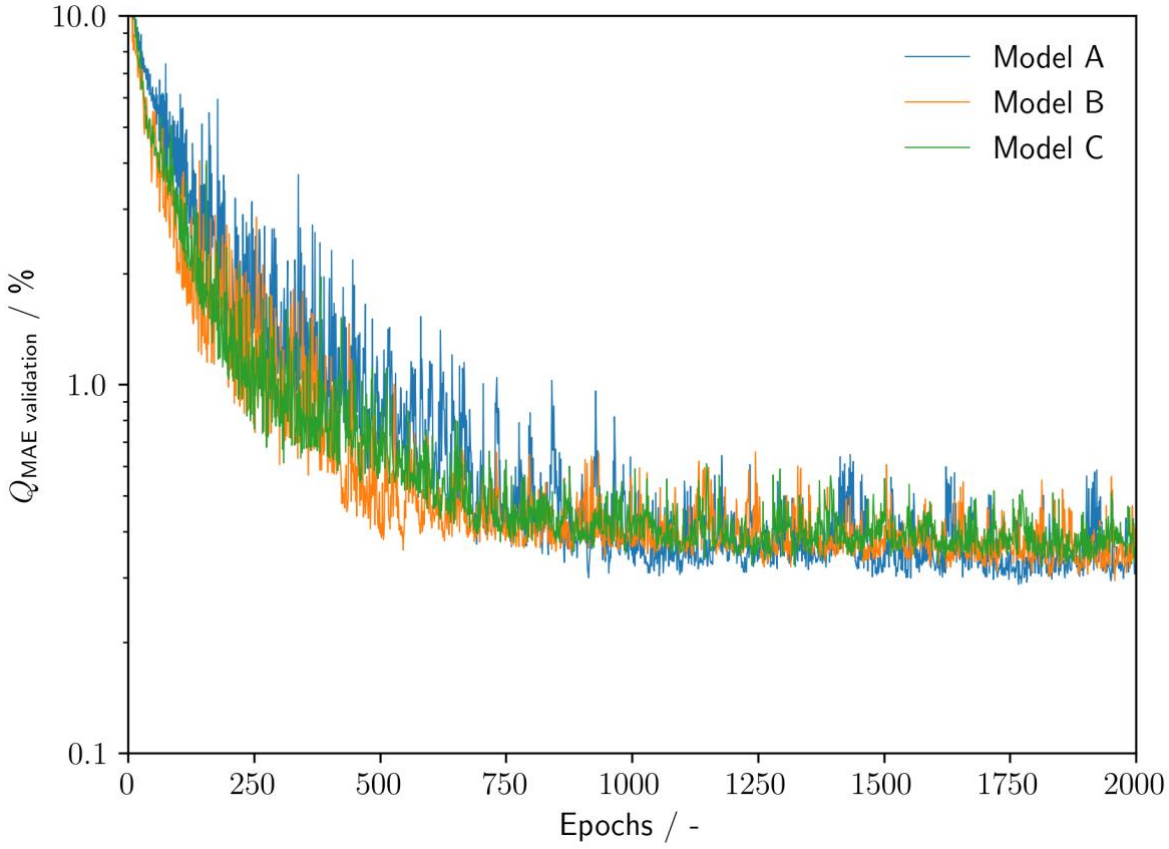

Figure 3.10 Validation loss over epochs for model A, B, C

As shown in Figure 3.11, Model B performs best on the majority of samples. However, model B has one outlier sample with an MAE of 2.8 %, worsening its average loss. All in all, the three models perform extremely well, predicting 95 % of the validation samples within an MAE margin of 0.6 %.

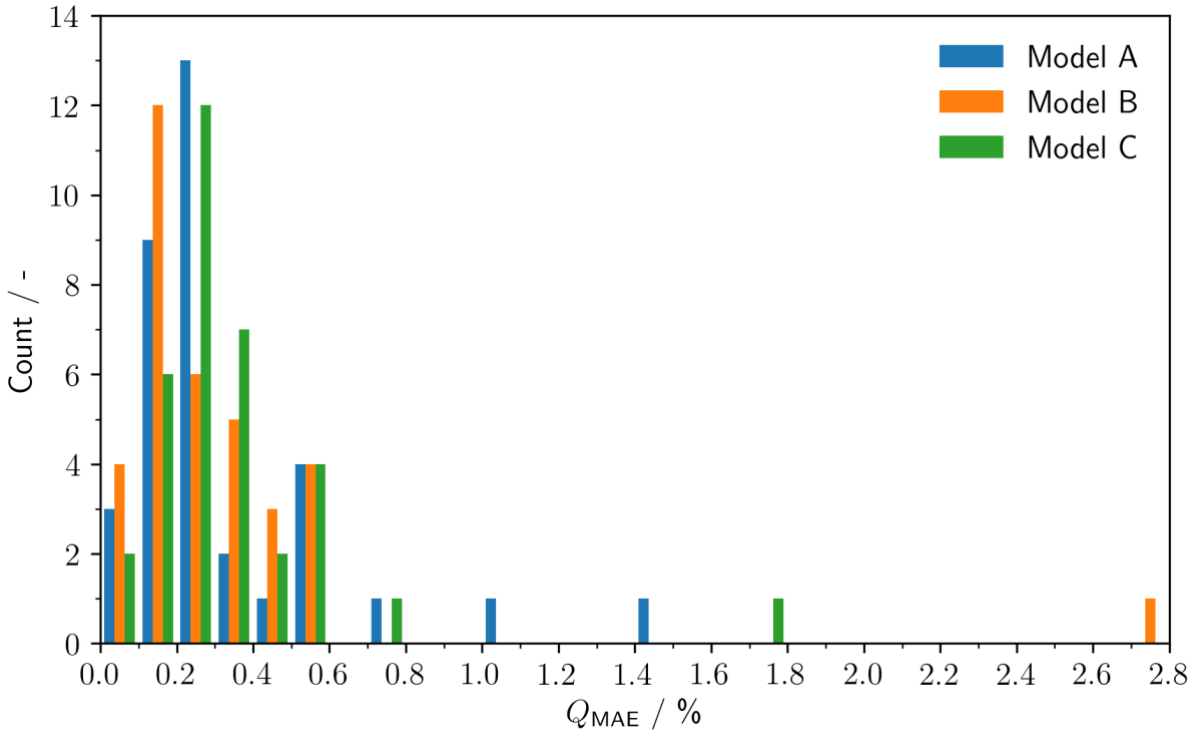

Figure 3.11 MAE distribution for every validation sample

Finally, Figure 3.12 compares the expected SOH profiles to the predictions from model A, B, and C.

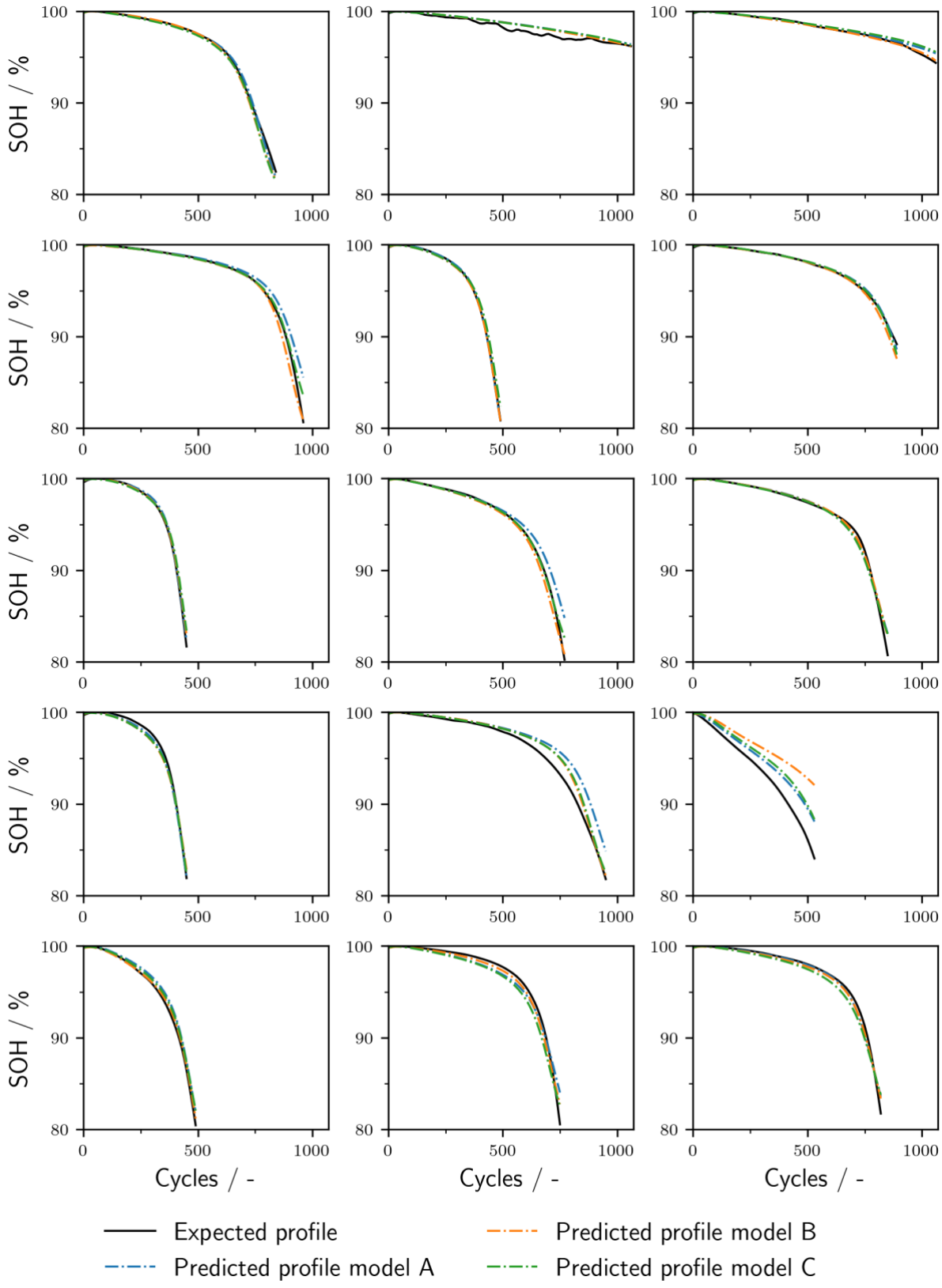

Figure 3.12 Expected SOH profiles compared to the the predictions from model A, B, and C for 15 validation samples

4 Application

In this chapter, the developed model is applied to a real-life EV charging profile. Since the applied charging profile strongly differs from the charging profile in the dataset and another cathode material is used, the results have no claim to be accurate. The point of this application is to check the model for plausibility. A higher current for example, should result in accelerated battery aging.

4.1 Real-Life EV Charging Profile

The real-life charging profile data includes current, voltage, and temperature profiles from an EV charged with DC. The current profile was measured directly, whereas voltage and temperature were estimated based on an already existing battery simulation.

As the EV charging profile is not a fast-charging profile (maximum 0.9 C), one cycle takes way longer than one of the initial dataset. A cycle in the EV charging profile takes 115 minutes, compared to an average cycle of 35 minutes in the training dataset. The trainable parameters of the LSTM, though, are only trained to a maximum of 40 minutes, potentially leading to odd results for longer sequences. Therefore, the time is scaled on the EV charging profile. The current increases by the same factor the time decreases by, whereas the influence of time on voltage and temperate profiles can be neglected.

The time scaling basically transforms the EV charging profile into a fast-charging profile. Here, two different time scaling factors are examined: 2 and 4. Figure 4.1 compares the different current profiles to a cycle from the training set.

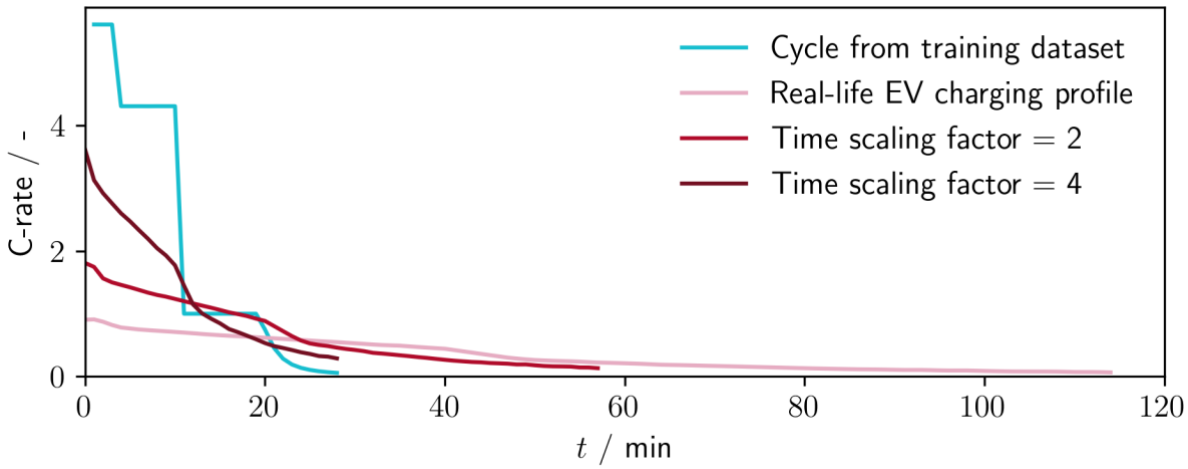

Figure 4.1 Current profile comparison between a representative cycle from the training set, the real-life EV charging profile, and the time scaled profiles

The lithium-ion battery used in the EV charging profile is an NMC cell, having a different voltage range during charging compared to the LFP cells from the training set (see Figure 4.2). As explained in chapter 3.2.2, the voltage profiles are first normalized and, subsequently, standardized with the scaling parameters from the initial dataset. Therefore, the scaled profiles are roughly within the same range as the training set voltage profile (see Figure 4.4).

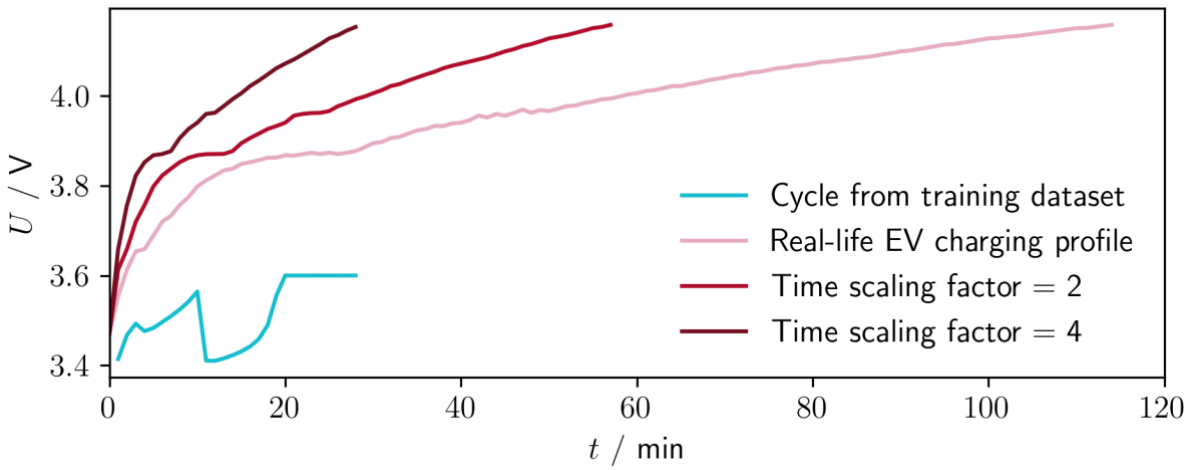

Figure 4.2 Voltage profile comparison between a representative cycle from the training set, the real-life EV charging profile, and the time scaled profiles

The ambient temperature of the EV charging profile is 30 °C, matching the initial dataset's one (see Figure 4.3). However, the temperature profiles vary heavily between

the different samples of the initial dataset (see Figure 3.4), making it difficult to compare the temperature profiles.

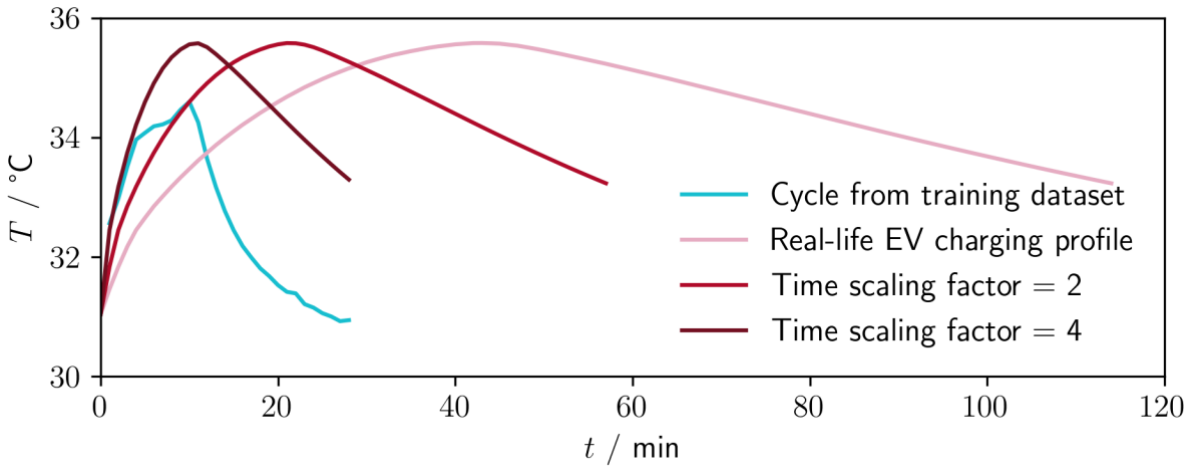

Figure 4.3 Temperature profile comparison between a representative cycle from the training set, the real-life EV charging profile, and the time scaled profiles

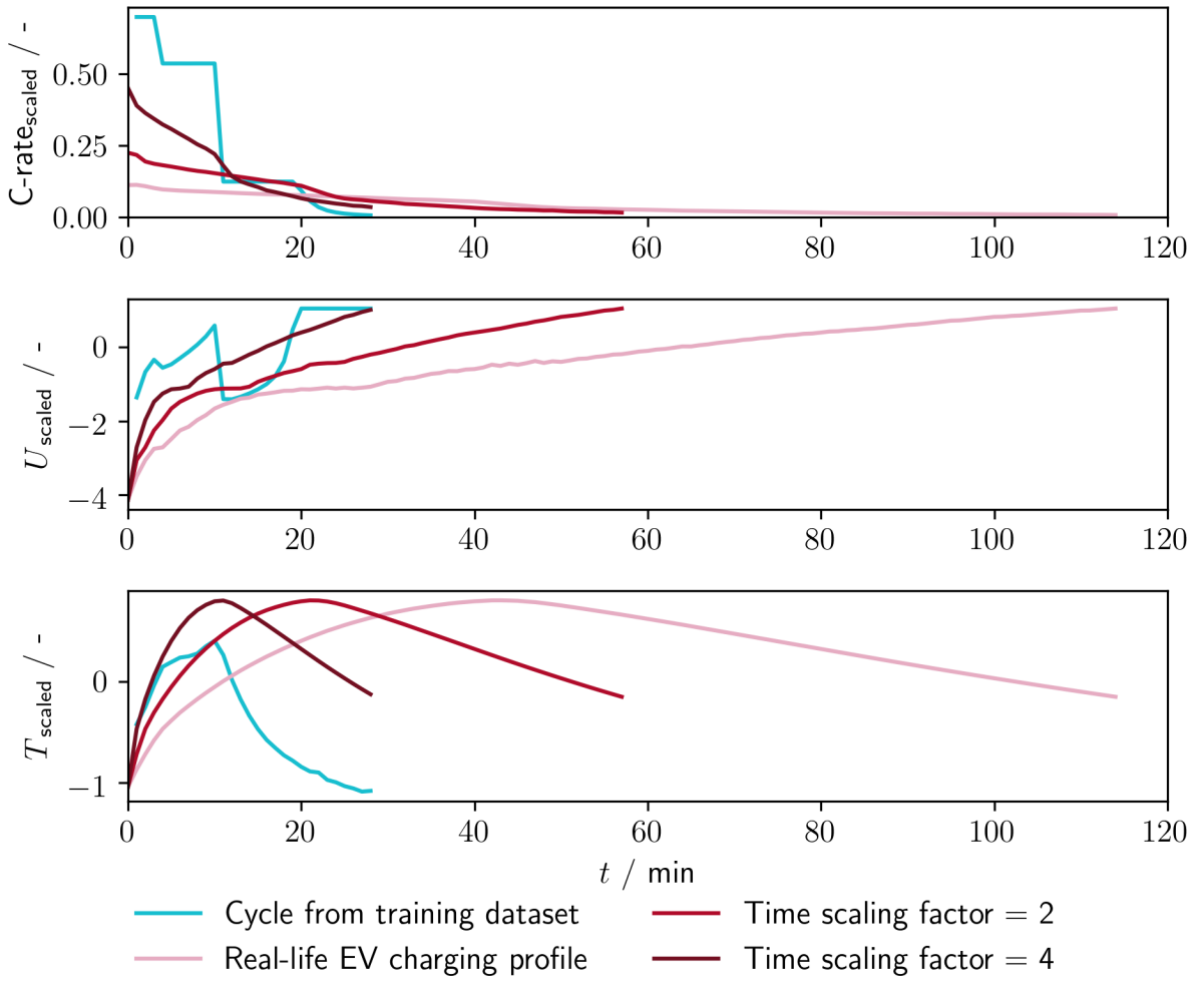

Figure 4.4 Comparison of the scaled current, voltage, and temperature profiles

4.2 Results

To predict results for the three regarded profiles, the number of input cycles must be specified. As the profiles only contain one cycle, the input array is filled up with copies of the current, voltage, and temperature profiles. The number of copied cycles determines the number of cycles the SOH is predicted for. The model predicts the SOH for 600, 700, 800, 900, 1000, and 1070 input cycles. For less than 600 cycles, the model does not have enough training samples to deliver a good prediction. Model A was used for the prediction.

Figure 4.5 shows the results for the unscaled EV charging profile. Since all predictions nearly course on the same path independently from the number of input cycles, the

results seem to be plausible. The cell reaches the threshold value of 80 % SOH at 700 cycles.

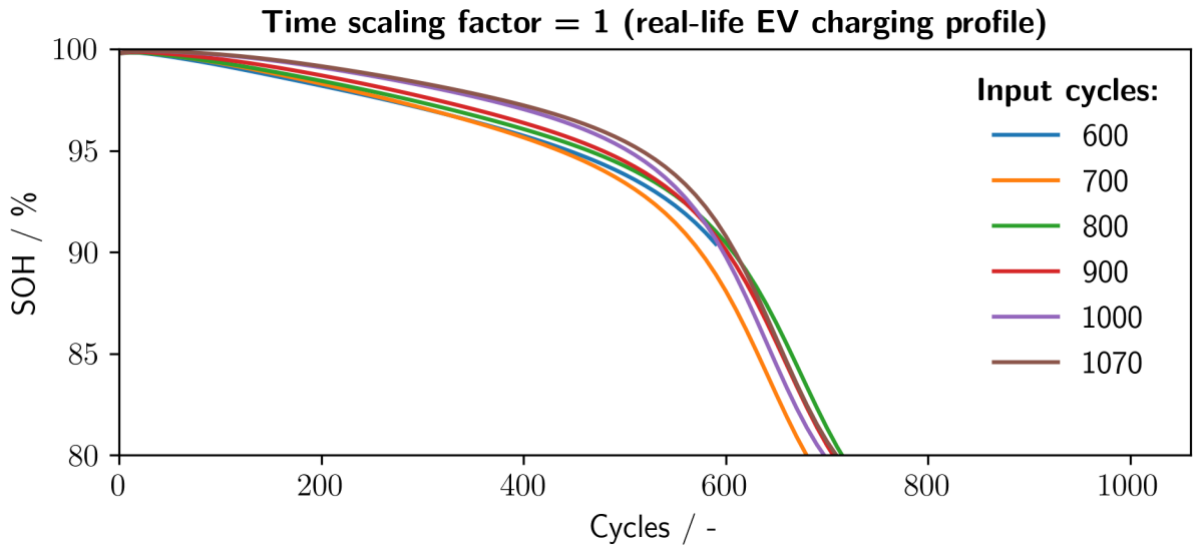

Figure 4.5 Results for the unscaled EV charging profile

When scaling time with a factor of 2, the results seem to be less plausible (see Figure 4.6). The different predictions do not follow the same path and the cell reaches 80 % SOH at 800 cycles on average. This contradicts the expectation that cells charged with a higher current age faster.

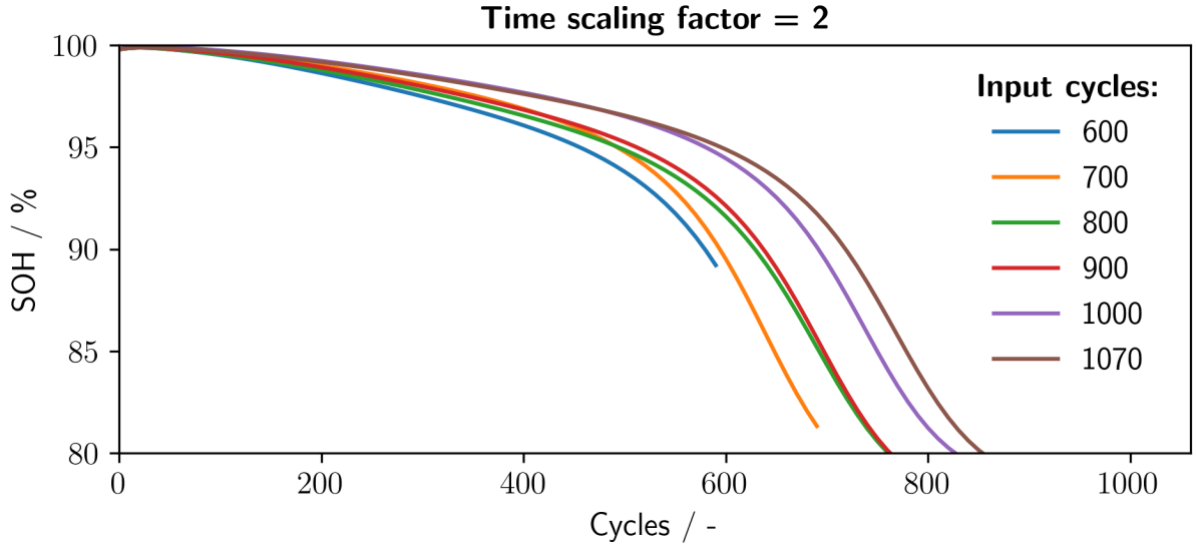

Figure 4.6 Results for the time scaled EV charging profile (time scaling factor = 2)

For a time scaling factor of 4, the results look even more odd (see Figure 4.7). The predictions do not follow the same path at all, and the cell does not seem to age very much, despite the fact that this is by far the profile with the highest current (see Figure 4.1).

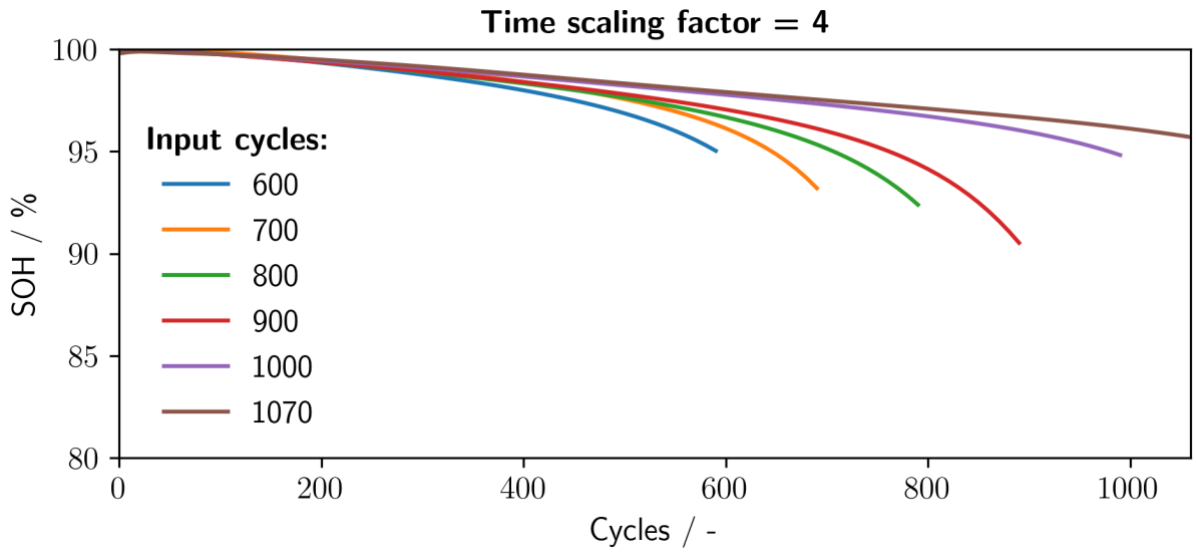

Figure 4.7 Results for the time scaled EV charging profile (time scaling factor = 4)

All in all, the model does not pass the plausibility check for all tested profiles. The predictions for different numbers of input cycles vary strongly, especially for the two time scaled profiles. In addition, the model predicts less battery aging for profiles with a higher charging current, contradicting the expectation for this plausibility check. Similar results were achieved with the models B, C, and further models from the grid search.

5 Conclusion

In conclusion, this work demonstrates the promises and challenges of battery aging models based on Machine Learning approaches. On the one hand, the developed model delivers an outstanding performance on the initial dataset, making extremely accurate predictions. On the other hand, this work points out the issues mathematical models are facing when it comes to application on new data with different boundary conditions.

This work only looks into battery aging occuring during charging. Battery aging during discharging and calendar aging are considered as neglectable consistent with the objectives, and, thus, are not taken into account.

On the validation set, the best model A achieves an MAE of only 0.33 %. It predicts the SOH profile of all validation samples in a 1.3 % MAE margin, with 72 % of the samples in a margin of 0.3 %. Those results can be attributed to the extensive hyperparameter optimization, running a grid search of 72 different hyperparameter configurations in total. The final hyperparameter combinations are surprising insofar that the number of units for the first LSTM layer is only 64, the minimum value for this hyperparameter. Thus, the LSTM layer with the minimum number of units has 16 for model A and B, and only 8 for model C. This supports the findings from Graves et al. [26] and Mohamed et al. [33], stating that depth is key for LSTMs, whereas the number of units only play a minor role in the model's performance.

The application to a real-life EV charging profile shows the boundaries of the model. Predictions for a different number of input cycles diverge, when they theoretically should lie on the same chart. Furthermore, the model predicts less battery aging for profiles with a higher charging current. Those odd predictions exhibit the model's boundaries resulting from the specific conditions of the initial dataset, as well as from the architecture of the LSTM.

One obvious boundary is that the cells in the training dataset are cycled under extreme fast-charging conditions, with the maximum current ranging from 4 C up to 8 C. Lithium-ion batteries do not reach such high currents in real-life applications, as 3 C

was the state of the art for EVs in 2019 [35]. As a result, the model is only trained for a sequence of 40 minutes, while almost every real-life charging process takes longer. In addition, the initial dataset only employs one-, two-, or four-step charging profiles. Hence, the model only learns from static charging profiles, whereas most real-life charging profiles are dynamic (for example the one applied to the model here). As the model is trained only with LFP cells, the application to other cathode types might be limited. Apart from the varying current range (see chapter 3.2.2), aging mechanisms also vary between different cathodes.

But the application also shows the downsides of the chosen Encoder-Decoder architecture. In the initial dataset, most samples end with an SOH of 80 % - 83 %. As the Encoder first processes all input time steps, the model learns from the correlation of the end SOH and the number of input samples provided. In other words, the model predicts the SOH after a particular cycle based on information from future cycles. This explains the varying predictions for different numbers of input cycles. The core of this problem is that the input data is actually four-dimensional but must be transformed to a three-dimensional input array for the LSTM. Here, the approach to just add the input cycles to the input features was selected, resulting in the described issue.

This causal analysis suggests that with a different LSTM architecture this problem can be overcome in future work. Figure 5.1 compares the proposed architecture to the Encoder-Decoder architecture. The proposed architecture is based on the idea that the SOH for a specific cycle is predicted with the information from this cycle and the knowledge of already occurred battery aging from previous cycles. Therefore, the data for one cell is split so that one sample only contains the data from one cycle. This sub dataset comprises the measured sequence from only one cycle, with current, voltage, and temperature as three features. The model only predicts one SOH value after the processed cycle. How can the model access information from previous cycles, though? In ordinary LSTMs the internal state is reset after each sample. This is where stateful LSTMs come into play. They pass the internal state of the last time step to the first time step of the following sample, providing the knowledge of previous battery aging. After the LSTM processed all cycles for one cell, the internal state can be reset for the following cell.

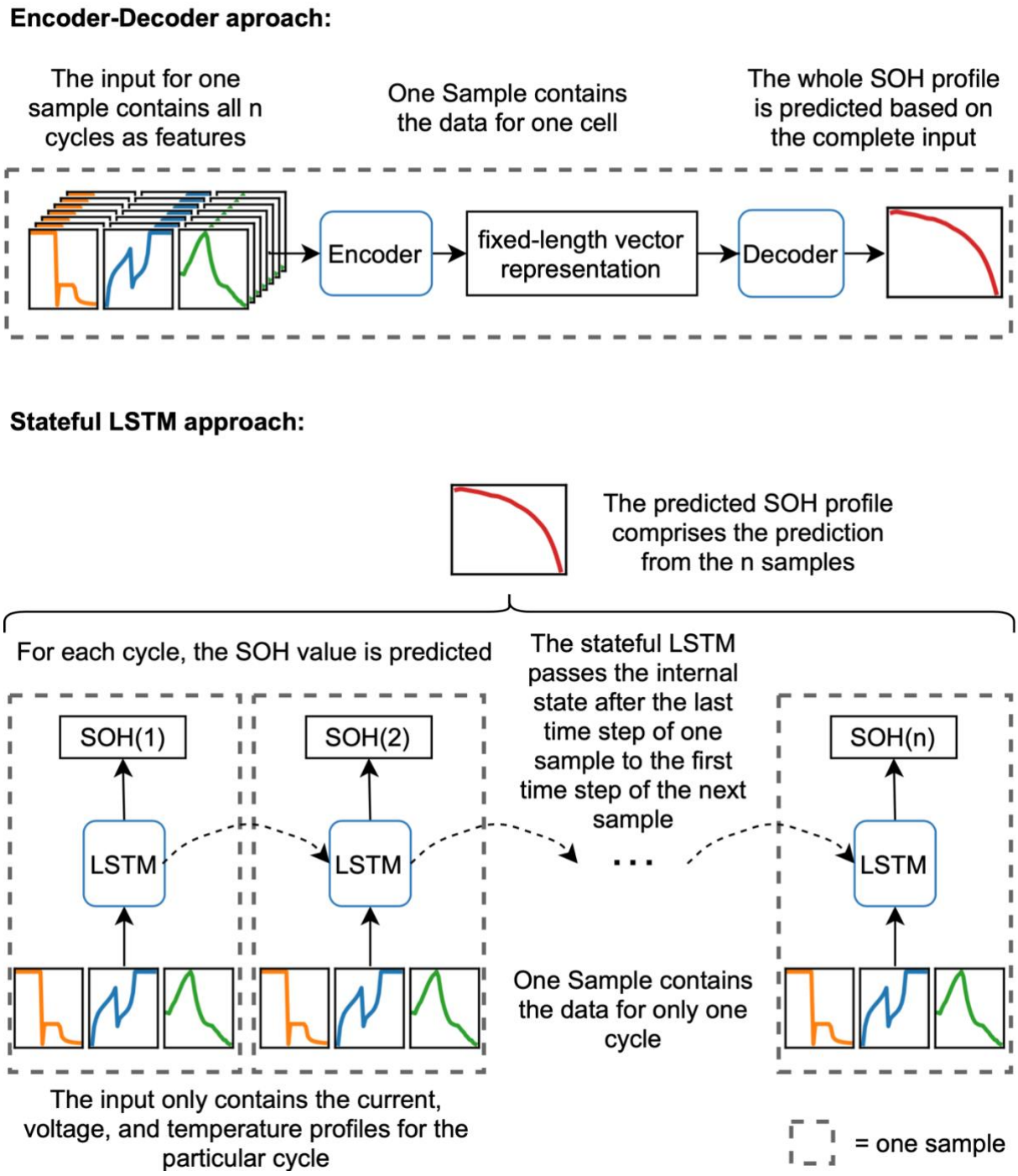

Figure 5.1 Comparison between the used Encoder-Decoder approach and a stateful LSTM approach

Based on the already impressive battery aging predictions the Encoder-Decoder architecture achieves on the initial dataset, it is proposed to combine the stateful LSTM architecture with more training data from different datasets to advance the model, making it robust to real-life EV charging profiles.

6 Bibliography

[1] Q. Badey, G. Cherouvrier, Y. Reynier, S. Franger and D. Jean-Marc, "Ageing forecast of lithium-ion batteries for electric and hybrid vehicles," in *Current Topics in Electrochemistry*, 2011.

[2] K. A. Severson, P. M. Attia, N. Jin, N. Perkins, B. Jiang, Z. Yang, M. H. Chen, M. Aykol, P. K. Herring, D. Fraggedakis, M. Z. Bazant, S. J. Harris, W. C. Chueh and R. D. Braatz, "Data-driven prediction of battery cycle life before capacity degradation," *Nature Energy,* vol. 4, p. 383–391, 2019.

[3] P. M. Attia, A. Grover, N. Jin, K. A. Severson, T. M. Markov, Y.-H. Liao, M. H. Chen, B. Cheong, N. Perkins, Z. Yang, P. K. Herring, M. Aykol, S. J. Harris, R. D. Braatz and S. Ermon, "Closed-loop optimization of fast-charging protocols for batteries with machine learning," *Nature,* vol. 578, p. 397–402, 2020.

[4] A. Veeraraghavan, V. Adithya, *A.* Bhave and S. Akella, "Battery aging estimation with deep learning," in 2017 IEEE Transportation Electrification Conference (ITEC-India), Pune, India, 2017.

[5] Y. Choi, S. Ryu, *K.* Park and H. Kim, "Machine Learning-Based Lithium-Ion Battery Capacity Estimation Exploiting Multi-Channel Charging Profiles," IEEE Access, vol. 7, pp. 75143-75152, 2019.

[6] M. Lucu, E. Martinez-Laserna, *I.* Gandiaga, K. Liu, H. Camblong, W. D. Widanage and J. Marco, "Data-driven nonparametric Li-ion battery ageing model aiming at learning from real operation data," *Journal of Energy Storage,* vol. 30, 2020.

[7] C. Pillot, "Impact of *the xEV* Market growth on Lithium-Ion Batteries and Raw Materials Supply 2018-2030," Avicenne Energy, Strasbourg, France, 2019.

[8] B. Writer, Lithium-Ion Batteries, *Cham: Springer* Nature Switzerland AG, 2019.

[9] R. Korthauer, Lithium-Ion Batteries: *Basics and* Applications, Berlin: Springer-Verlag GmbH Germany, 2018.

[10] MIT Electric Vehicle Team, "A *Guide to* Understanding Battery Specifications," December 2008. [Online]. Available: http://web.mit.edu/evt/summary_battery_*specifications.pdf.* [Accessed 8 October 2020].

[11] S. W. Michael Pozin, *"Chapter 12* Li-Secondary Battery Damage Control," in Electrochemical Power Sources: Fundamentals, Systems, and Applications, Elsevier B.V., 2019, pp. 507-629.

[12] Epec, „Lithium battery technologies," *Epec,* [Online]. Available: http://www.epectec.com/batteries/lithium-battery-technologies.html. [Zugriff am 5 November 2020].

[13] K. Liu, K. Li, *Q.* Peng and C. Zhang, "A brief review on key technologies in the battery management system of electric vehicles," *Front. Mech. Eng*., vol. 14, p. 47–64, 2019.

[14] M. Beheshti, "Selecting the *right charge-management* solution," Texas Instruments Incorporated, 2009. [Online]. Available: https://www.ti.com/lit/an/slyt334/slyt334.pdf. [Accessed 8 October 2020].

[15] EEMB Co., Ltd., „*Lithium-ion Battery* DATA SHEET Battery Model: LIR18650 2600mAh," November 2010. [Online]. Available: https://www.ineltro.*ch/media/*downloads/SAAItem/45/45958/36e3e7f3-2049-4adb-a2a7-79c654d92915.pdf. [Zugriff am 9 October 2020].

[16] P. Keil, S. F. *Schuster,* J. Wilhelm, J. Travi, A. Hauser, R. C. Karl and A. Jossen, "Calendar Aging of Lithium-Ion Batteries," Journal of *The Electrochemical Society,* vol. 163, no. 9, pp. 1872-1880, 2016.

[17] T. Waldmann, M. Wilka, *M.* Kasper, M. Fleischhammer and M. A. Wohlfahrt-Mehrens, "Temperature dependent ageing mechanisms in Lithium-ion batteries - A Post-Mortem *study," Journal of Power Sources,* vol. 262, pp. 129-135, September 2014.

[18] D. U. Sauer and H. *Wenzl,* "Comparison of different approaches for lifetime prediction of electrochemical systems—Using lead-acid batteries as example," Journal of Power Sources, vol. 176, no. 2, pp. 534-546, *2008.*

[19] Y. Yu, X. Si, *C.* Hu and J. Zhang, "A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures," Neural Computation, vol. 31, no. 7, pp. 1235 - 1270, 2019.

[20] J. Zou, Y. Han *and S.*-S. So, "Chapter 2: Overview of Artificial Neural Networks," in *Artificial Neural Networks*, Sandown, Humana Press, 2008, pp. 15-23.

[21] E. Culurciello, „The fall *of RNN* / LSTM," Towards Data Science, 13 April 2018. [Online]. Available: https://towardsdatascience.com/the-fall-of-rnn-lstm-2d1594c74ce0. [Zugriff am 1 November 2020].

[22] J. Brownlee, Long Short-Term Memory *Networks With* Python, 2019.

[23] Z. Gu, Z. Li, *X.* Di and R. Shi, "An LSTM-Based Autonomous Driving Model Using a Waymo Open Dataset," Applied Science, vol. 10, no. 6, p. 2046, 2020.

[24] I. Goodfellow, Y. Bengio *and A.* Courville, Deep Learning, MIT Press, 2016.

[25] N. Vandeput, „Forecast KPI: *RMSE,* MAE, MAPE & Bias," Towards Data Science, 5 July 2019. [Online]. Available: https://towardsdatascience.com/forecast-kpi-rmse-mae-mape-bias-cdc5703d242d. [Zugriff am 30 October 2020].

[26] A. Graves, A.-r. *Mohamed and* G. Hinton, "Speech Recognition with Deep Recurrent Neural Networks," in ICASSP 2013, Vancouver, 2013.

[27] N. Srivastava, G. Hinton, *A.* Krizhevsky, I. Sutskever and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research,* vol. 15, pp. 1929-1958, 2014.

[28] F. Chollet, "keras," *2015. [Online*]. Available: https://github.com/keras-team/keras.

[29] K. Cho, B. van *Merriënboer, C*. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk and Y. Bengio, "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Doha, 2014.

[30] C. Ranjan, "Step-by-step understanding *LSTM Autoencoder* layers," Towards Data Science, 4 June 2019. [Online]. Available: https://towardsdatascience.com/step-by-step-understanding-lstm-autoencoder-layers-ffab055b6352. [Accessed 27 October 2020].

[31] G. E. Hinton, N. *Srivastava,* A. Krizhevsky, I. Sutskever and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," Toronto, 2012.

[32] D. P. Kingma and J*. Ba,* "Adam: A Method for Stochastic Optimization," in 3rd International Conference for Learning Representations, San Diego, 2015.

[33] A.-r. Mohamed, G. *E. Dahl* and G. Hinton, "Acoustic Modeling Using Deep Belief Networks," IEEE Transactions on Audio Speech and Language Processing, vol. 20, no. 1, pp. 14-22, 2012.

[34] X. Glorot, A. Bordes *and Y.* Bengio, "Deep Sparse Rectifier Neural Networks," in Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, Fort Lauderdale, FL, 2011.

[35] eucar, „Battery requirements for future *automotive applications*,“ July 2019. [Online]. Available: https://eucar.be/wp-content/uploads/2019/08/20190710-EG-BEV-FCEV-Battery-requirements-FINAL.pdf. [Zugriff am 10 November 2020].

[36] S. Patoux, L. Sannier, *H.* Lignier, Y. Reynier, C. Bourbon, S. Jouanneau, F. Le Cras and S. Martinet, "High voltage nickel manganese spinel oxides for Li-ion batteries," *Electrochimica Acta,* vol. 53, no. 12, p. 4137–4145, 2008.