HERAUSRAGENDE MASTERARBEITEN AM DISC

FACHBEREICH Science & Engineering

STUDIENGANG Software Engineering for Embedded Systems

MASTERARBEIT

Conception, Implementation and Evaluation of Machine Learning Algorithms for AI-based Simulation Models for Electric Powertrain Components

AUTOR/IN

Stefan Holbach

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

DISTANCE AND INDEPENDENT STUDIES CENTER

Declaration

Ich versichere, dass ich diese Masterarbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Place, date

Siganture

Abstract

This thesis aims at investigating the capability and feasibility of Machine Learning algorithms for developing models simulating the behavior of E/E powertrain components. Machine learning based simulation models possess the advantage of being trained via real measurement data and no time-consuming manual set up of equation and parameter adaptions are needed to get a proper simulation model of the component.
For this purpose, the thesis starts with the introduction of E/E powertrain components of interest. Moreover, Machine Learning algorithms are introduced that support model based and supervised training and are hence of interest for behavior simulation.
The design, implementation, training and optimization of the different Machine Learning based simulation models according to the provided data is presented. These models are not only simulation models of the single introduced components but also models of the composition of these components.
The resulting models are evaluated against test data which has not been used for training. This evaluation illustrates the ability and inability of the different Machine Learning algorithms to simulate and generalize specific powertrain components. It also illustrates the necessary scope of the models according the number of composite components and their accuracy.

Table of Contents

Abbreviations

AC	Alternating Current
ADAM	Adaptive Moment Estimation
AI	Artificial Intelligence
API	Application Programming Interface
DC	Direct Current
E/E	Electric/Electronic
ELU	Exponential Linear Unit
FNN	Feedforward Neural Network
GPU	Graphics Processing Unit
GRU	Gated Recurrent Units
HIL	Hardware In the Loop
IGBT	Insulated Gate Bipolar Transistor
LSTM	Long Short Term Memory
MAE	Mean Absolute Error
MAPE	Mean Absolute Percentage Error
ML	Machine Learning
MOSFET	Metal Oxide Semiconductor Field Effect Transistor
MSE	Mean Squared Error
PIL	Processor In the Loop
PWM	Pulse Width Modulation
RELU	Rectified Linear Unit
RMSE	Root Mean Squared Error
RMSPE	Root Mean Squared Percentage Error
RMSRE	Root Mean Squared Relative Error
RNN	Recurrent Neural Network
SELU	Scaled Exponential Linear Unit
SGD	Stochastic Gradient Descent
SiC	Silicium Carbit
SIL	Software In the Loop
SOC	State Of Charge
SPM	Surface Mounted Permanent Magnet Synchronous Machine
SVM	Space Vector Modulation

List of Symbols

α	factor for exponential function used in ELU and SELU act. function
β_1	sloping moment parameter
β_2	sloping squared moment parameter
ϵ	smooting value for Adam optimizer
η	learning rate
θ	parameter of a Machine Learning algorithm
$\boldsymbol{\theta}$	parameter set
λ	linear factor for SELU activation function
σ^2	variance
φ	phase shift
$\phi()$	activation function
Ψ	flux linkage
ω	angular frequency
∇	gradient
b	bias term
f	frequency
I	DC current
i	AC current
$\hat{i}$	AC current‘s amplitude
J()	loss function
k	discrete point in time
L	electrical inductance
$\mathbf{m}$	moment set for Adam optimizer
$\hat{\mathbf{m}}$	filtered moment set for Adam optimizer
M	torque
n	rotation speed
p	number of pole pairs
Q	electrical charge
R	electrical resistance
$\mathbf{s}$	squared moment set for Adam optimizer
$\hat{\mathbf{s}}$	filtered squared moment set for Adam optimizer
t	time
U	DC voltage
u	AC voltage

$\hat{u}$	AC voltage‘s amplitude
w	weight value in Neural Networks
$\mathbf{W}$	weight set
x	input value
$\mathbf{X}$	input set
y	output value
$\hat{y}$	predicted output value
$\mathbf{Y}$	output set

List of Tables

List of Algorithms

List of Figures

1. Introduction

This thesis investigates the feasibility of different Machine Learning approaches for the modeling and simulation of common E/E components, which are used in the powertrain of battery electric and hybrid electric vehicles.

1.1. Thesis' Motivation

Simulation of systems or part of systems is state of the art in product development of mechatronic systems today. The benefits for the use of simulations are e.g.:

- Enabling of system determination at a very early stage of development, with no physical hardware yet available
- Cost-saving opportunity as no physical system as well as no physical infrastructure to perform the tests is needed for the experiments
- Enables fast, automated testing, when used in applications like hardware in the loop or similar

Defining a model of the system at hand might become a challenging task. The reason for this is, that the assumptions of the system‘s properties while modeling might be identified to be wrong after modeling and comparing with the system at hand. This can lead to new modeling, which needs additional time. Assumptions about the system must be made regarding the paramters used as well as the equations that form the basis of the model. Also the definition of the interdependencies of the system‘s sub-components might be incomplete.
In case the inner structure is not known at all, which might be the case by purchased systems, the system‘s transfer function must be identified. Approaches for identifications are described in [5]. This can be very time consuming especially if there is discontinuous and/or non-linear behavior in the system at hand. Specific behavior can lead to specific necessary identification approaches.
Machine Learning algorithm possess the property that, if enough data is available, they can reconstruct the input-output correlation. This is also valid for non-linear systems without any change in the ML algorithm. Hence the definition of the identification method does not need to be defined before the identification itself, which can also be a source of failures. This points out an advantage compared to conventional approaches.

1.2. Thesis' Goal

The thesis' goal is the investigation of the feasibility of Machine Learning algorithms for the simulation of E/E components in automotive powertrain applications.
For this reason different components, which interact in an electrically driven or electrically assisted vehicle, are defined to be investigated. The input-output relations of those components need to be identified. This information is then used to train the Machine Learning models. Afterwards, the behavior of these Machine Learning models must be verified against the original values. By this approach the following scientific questions shall be answered:

1. Which Machine Learning algorithms are capable of modeling the behavior of electric/electronic component?
2. What is the precision of the simulation models trained by Machine Learning algoritms compared to the original data?
3. What is the necessary scope of the Machine Learning based modeling w.r.t. the number of components that can be simulated within one model?
4. Which concrete applications can be identified for the Machine Learning based simulation models?

1.3. Terminology

Artificial Intelligence
Artificial Intelligence is the umbrella term that describes a machine‘s ability to fulfill human-like behavior, which includes learning, judging and problem solving. Domains of Artificial Intelligence are e.g. Machine Learning, Natural Language Processing and Machine Vision [18].

Machine Learning
Machine Learning defines a computer program that can learn to produce a behavior that is not explicitly programmed by the developer by using defined input and output data [18].

Deep Learning
The research of Deep Learning examines Neural Networks with deep stacks. Whereas the accurate number of layers for a Neural Network necessary to be called Deep Neural Networks is not defined. In the 1990s Neural Networks with 2 or more layers have already been called Deep Neural Network. Nowadays some problems are solved by using Neural

Networks with more than 100 layers [10].

Used Term in this Thesis

Due to the fuzzy definition of Deep Learning and the fact that in this thesis the problems to be solved need the ability to learn according defined input and output data, in the further course of this thesis the term Machine Learning will be used.

1.4. Thesis' Structure

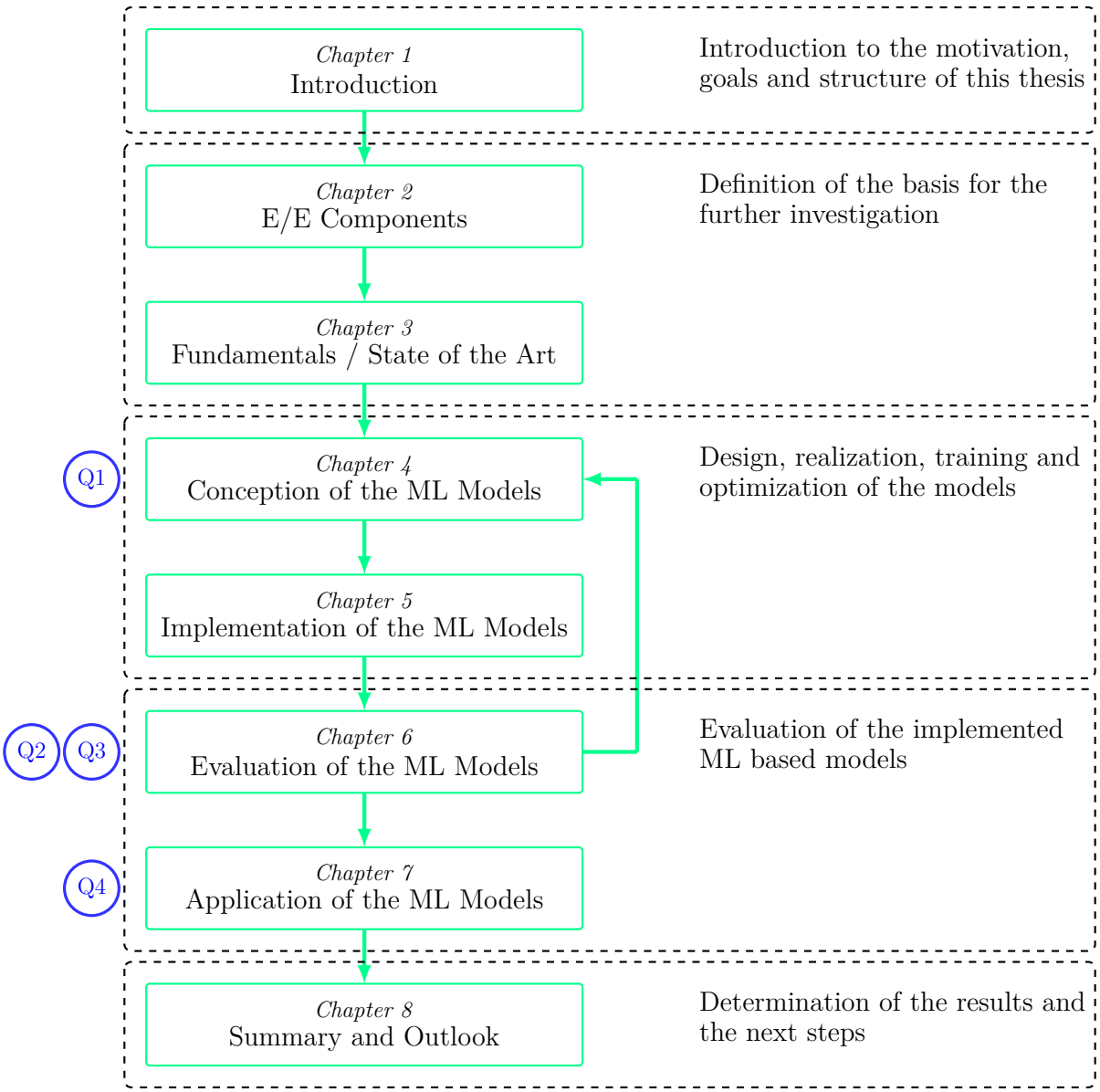


Figure 1: Overview of the thesis' structure

After pointing out the basics that are used for the thesis‘ investigation, the development process of different Machine Learning models is presented. Figure 1 describes the flow of the thesis and the interplay of the individual chapters. The figure also shows which chapter investigates the questions defined in chapter 1.2 highlighted in blue circles.
A property of this process, which is worthwhile to mention, is that design, training, opimization and evaluation of the Machine Learning algorithm is an iterative process. For the sake of readability these phases are written in a sequential manner.

2. Determination of E/E Powertrain Components of Interest

In this chapter the components to be modeled via Machine Learning algorithms are presented. Figure 2 shows the schematic overview of the main E/E components used for the powertrain in battery electric vehicles (BEV) and plug in hybrid electric vehicles (PHEV). The battery is the energy storage of the electric system. Its supplies the inverter with electrical DC power. The inverter transforms this DC power to 3 phase AC values. These AC values are needed by the electric machine to transform the electrical power to mechanical rotation power for the vehicle‘s traction.

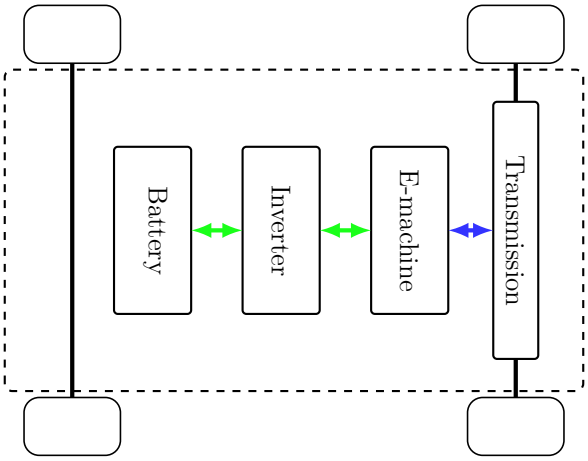


Figure 2: Overview of the E/E powertrain components

2.1. Battery

The battery is the electrochemical energy storage used in BEVs and PHEVs. Currently the most common used battery technology for the usage in BEVs and PHEVs is the lithium-ion battery technology. The reason for this is the high cell voltage as well as the high power density of the lithium-ion battery [33]. There are a lot of different types of lithium-ion batteries, which differentiate mostly in the material used for the electrodes, like $LiFePO_4$/graphite (LFP) and $LiNiMnCoO_2$/graphite (NMC). In this thesis the battery AMP20M1HD-A from the producer A123 is used, which is based on the LFP technology[1].

One of the information which is mostly important for the battery management system is the state of charge (SOC). The SOC is a dimensionless quantity that describes the ratio

[1]For the further course of this thesis the term battery will be used to describe the $LiFePO_4$/graphite battery type

of the available capacity $Q_{\mathrm{b}}(t)$ to the the nominal capacity $Q_{\mathrm{b},0}$, [32].

$$\mathrm{SOC} = \frac{Q_{\mathrm{b}}(t)}{Q_{\mathrm{b},0}} \tag{1}$$

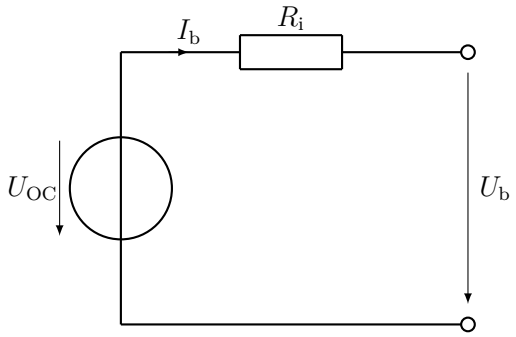


Figure 3: Equivalent circuit diagram of the battery

Figure 3 shows the equivalent circuit diagram of a battery in stationary operation mode. The battery can be described as a voltage source with the open circuit voltage U_{OC} in series to the internal resistance R_{i}. The open circuit voltage is the cell voltage that is measureable at the terminals when no current flow is active. It is depending on the battery‘s SOC, $U_{\mathrm{OC}}(\mathrm{SOC})$ [31]. The SOC dependency for the battery used in this thesis is shown in figure 4.

The internal resistance is depending on the battery‘s SOC as well as on the battery current direction and the battery‘s temperature, $R_{\mathrm{i}}(\mathrm{SOC}, sgn(I_{\mathrm{b}}), T)$, [31]. The temperature influence has been neglected in this thesis. The SOC and the current dependency of the internal resistance is shown in the figures 5 and 6.

The parameters of the battery used in this thesis can be seen in appendix D.

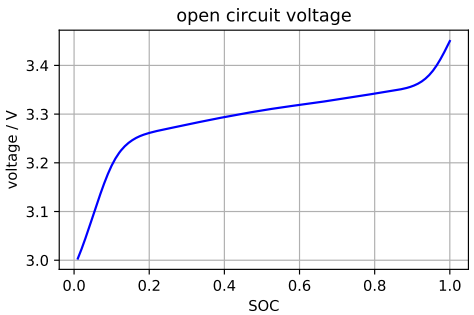


Figure 4: Dependency of the open circuit voltage on the state of charge

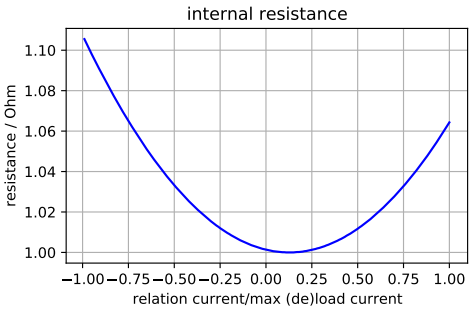
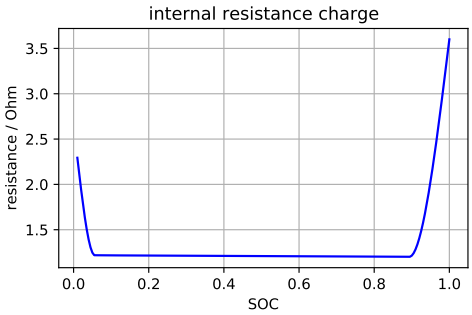
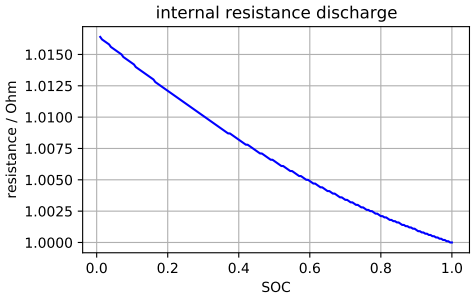


Figure 5: Dependency of the internal resistance on the battery current



(a) Resistance at charge



(b) Resistance at discharge

Figure 6: Dependency of the internal resistance on the State Of Charge

2.2. Inverter

The inverter is used to transform DC power to AC power during motor operation and also AC power to DC power during regeneration. The inverter is necessary to control the electric machine as the frequency and the amplitude of the applied 3 phase sinusoidal voltage can be controlled by using the inverter.
Figure 7 shows the structure of the inverter used in this thesis. The used topology is a 2-level 3-phase inverter. This means that there are 3 half bridges with 2 switches on each half bridge. One high side switch and one low side switch. Between those switches the phase connection is applied. The switches are used in a manner that they can either be fully closed, this means the switch is conducting or fully opened, which means the switch is not conducting. The two switches of a halfbridge must not be closed at the same time as this would lead to a short circuit of the battery. When the upper switch is closed, the DC voltage is applied to the electric machine, if the lower one is closed the inverter‘s ground potential is applied to the respective phase of the electric machine.

As switches (SiC-)MOSFETs or IGBTs are most commonly used. All real switches have in common that they do not switch immediately. The switching procedure takes some time, mostly in the range of some hundred nano seconds. So the upper switch must not directly

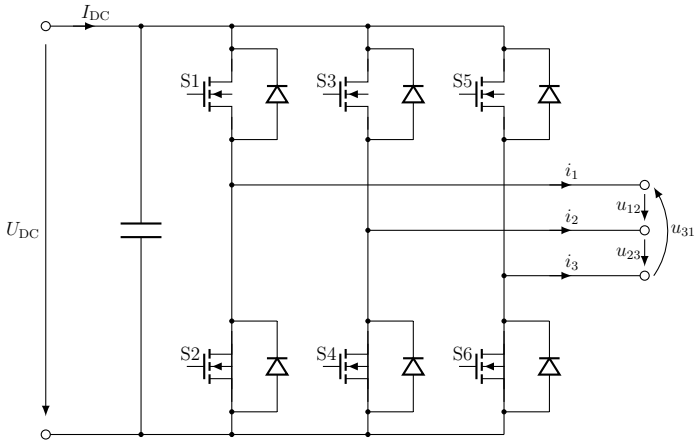


Figure 7: Equivalent circuit diagram of the inverter

close when the lower one is switching off and vice versa, as this can lead to a short circuit. Hence, it is necessary to implement a so called dead time where no switch is allowed to operate. As there is a current flow on the windings that must not be interrupted as this could lead to voltage peaks, $u = L\frac{\mathrm{d}i}{\mathrm{d}t}$, diodes are implemented in an antiparallel manner to the switches.

To be able to produce sinusoidal current with the discrete switching device, different switching patterns can be used like carrier signal based modulation, flat top modulation or space vector modulation. In this thesis the space vector modulation is used as it has a better voltage utilization than the carrier based signal and produces less harmonics than the flat top modulation [17].

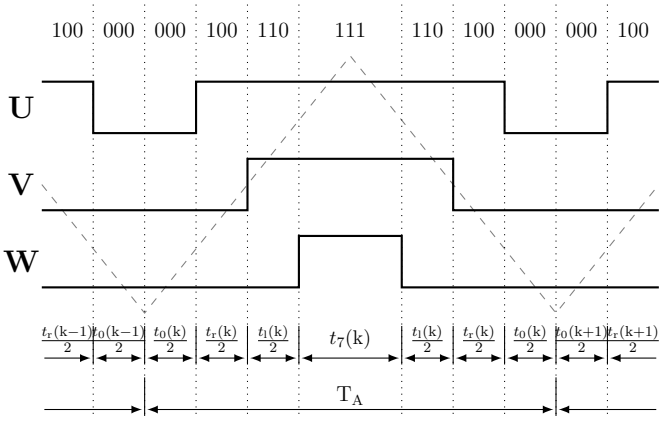


Figure 8: Switching pattern of the space vector modulation [15]

In figure 8 the switching pattern of the space vector modulation is shown. 1 means the upper switch is conducting, 0 means the lower switch is conducting. The dead time is not shown in this figure. It can be seen, that during one cycle six switching operation

take place and four different vectors are applied. When the two zero vectors are applied, their on-time are t_0 and t_7, there is no energy flow from energy source to the machine or vice versa. This whole switching pattern takes place within 1 PWM cycle. The length of this cycle is a compromise between the amount of harmonics of the phase current and the semiconductor‘s switching losses. Important parameters that have to be taken into account for chosing the PWM cycle are the inductance of the electric machine, the max. electrical frequency of the electric machine and the heat dissipation of the power electronics.
The switching frequency in this thesis is 10 kHz, which means that the six switching operations take place within 100 micro seconds.

2.3. Electric Machine

The electric machine type that is used in this thesis is the 3 phase surface mounted permanent magnet synchronous machine (SPM)[2]. This permanent magnet synchronous machine has the advantages over other machine types, like induction machine or reluctance machine, that it has a high power density, high efficiency, low noise and the inverter does not have to provide reactive power for magentization [2]. The SPM consists of a stator with a 3 phase winding and a permanent magnet excited rotor. The stator field exerts a force on the rotor which leads the rotor to turn with the same rotation speed as the stator field. So the rotor turns synchronously to the stator field. The rotor turns with a frequency of $f_{\mathrm{el}} = p \cdot f_{\mathrm{m}}$. This impelling force is called Lorentz force. The Lorentz force has the highest value when the stator field has an angle of 90 ° to the rotor field [3]. Figure 9 shows the schematic cross section of the SPM together with the rotor oriented frame. It can be seen, that the rotor oriented frame is depending on the angle of the rotor to the U axis.
The 3 phase voltage equations of the SPM is defined as follows

$$u_{\mathrm{U}} = R_{\mathrm{s}} \cdot i_{\mathrm{U}} + \frac{\mathrm{d}\Psi_{\mathrm{U}}}{\mathrm{d}t} \tag{2}$$

$$u_{\mathrm{V}} = R_{\mathrm{s}} \cdot i_{\mathrm{V}} + \frac{\mathrm{d}\Psi_{\mathrm{V}}}{\mathrm{d}t} \tag{3}$$

$$u_{\mathrm{W}} = R_{\mathrm{s}} \cdot i_{\mathrm{W}} + \frac{\mathrm{d}\Psi_{\mathrm{W}}}{\mathrm{d}t} \tag{4}$$

After transformation in the rotor oriented frame[3] the voltage can be described with help of the complex space vector, that consists of the d-part and of the q-part:

[2] In the following chapters the term electric machine will be used for the machine type at hand
[3] The transformation rule for the Clarke and Park transformation which are necessary for the transformation in the rotororiented frame can be found in the appendix A and appendix B

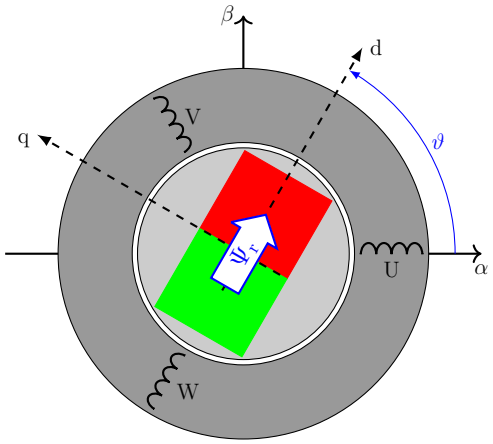


Figure 9: Cross section of the SPM [15]

$$u_\mathrm{d} = R_\mathrm{s} \cdot i_\mathrm{d} + L_\mathrm{s,d} \cdot \frac{\mathrm{d}i_\mathrm{d}}{\mathrm{d}t} - \omega \cdot L_\mathrm{s,q} \cdot i_\mathrm{q} \tag{5}$$

$$u_\mathrm{q} = R_\mathrm{s} \cdot i_\mathrm{q} + L_\mathrm{s,q} \cdot \frac{\mathrm{d}i_\mathrm{q}}{\mathrm{d}t} + \omega \cdot (L_\mathrm{s,d} \cdot i_\mathrm{d} + \Psi_\mathrm{r}) \tag{6}$$

Equation 7 shows the definition of the SPM's torque, [29].

$$M_\mathrm{el} = \frac{3}{2} \cdot \frac{1}{\omega} \cdot \Re\{\underline{\vec{u_\mathrm{p}}}^\mathrm{R} \cdot \underline{\vec{i}}^{\mathrm{R}\,*}\} \tag{7}$$

After solving the equation and neglecting the reluctance torque, which is acceptable in case a SPM is used, the torque equation can be simplified to equation 9

$$M_\mathrm{el} = \frac{3}{2} \cdot i_\mathrm{q} \cdot \Psi_\mathrm{r} \tag{8}$$

It can be seen, that the torque of the SPM is only depending on the q current, not on the d-current.
The electro-motoric force (emf), which is the voltage that is induced due to the movement of the rotating permanent magnet in the electric machine depends on the rotation speed and the rotor flux linkage

$$u_\mathrm{i} = \omega \cdot \Psi_\mathrm{r} \tag{9}$$

The parameters of the electric machine used in this thesis can be seen in appendix C.

3. Fundamentals and State of the Art

3.1. Modeling and Simulation of Dynamic Systems

In this chapter the basics of modeling and simulation of dynamic systems are discussed in brief which helps to categorize the models to be deployed in this thesis.
Simulation enables the estimation of the system‘s behavior without actually stimulating the system [6]. This has the advantage that no costly physical system and infrastructure needs to be available to investigate whether a system works as intended or to find possible malfunctions. Also depending on the goal of the simulation no real situation must be provoked that can be dangerous to life and environment. The needed information can be produced virtually.
Models are a prerequisite for simulation. A model is a simplified representation of the (partial) reality [6]. Simplified in this case means that it only represents the behavior and in- and outputs of interest but neglects the values not necessary for the scope of the particular simulation. As the scope of the simulation can vary, the structure and approach of models can vary as well. In the following the main differentiations of modeling types are presented.

(Quasi-)Continuous vs Discontinuous models
A continuous model transfers data to the internal and external variables at every point in time. The model can be expressed as differential equation and can be continuously solved. In case of using a discrete system for simulation, e.g. a computer or a microprocessor, the calculation is only processed during certain points in time. The model must be adapted and the differential equations must be transformed to difference equations. This transformation is called discretization. As these models still have to simulate continuous behavior they are often called quasicontinuous models.
Discontinuous models describe a specific state of a system, which is changed in an event driven manner. This can be the opening or closing of a switch, the change of a state machine, etc. For simulating this behavior no difference equation is used but logic expressions.
In many cases, the simulation of dynamic systems is a combination of (quasi-)continuous and discontinuous calculation [28]. So difference equation might only be solved during a certain state of the system‘s model.
In this thesis no event-driven changes of the component‘s behavior will be investigated. As the data for Machine Learning is discrete data, the resulting models of this work will be quasicontinuous models.

Distributed vs Concentrated Parameters

In case the system‘s behavior of interest is only depending on time, the parameters are concentrated parameters. The model can be represented by ordinary differential equations. If there is a spacial dependency next to the temporal dependency, the parameters are spacial distributed parameters [28]. The whole model must be described as partial differential equation.

In this thesis only the timely behavior of the E/E components is investigated.

White Box vs Black Box Modeling

White box modeling, also called theoretically based modeling, means the conception of the models based on theoretical, physical laws, e.g. the Maxwell‘s equations. The behavior of the input and output as well as the behavior of internal variables is known. For this kind of modeling no real system is necessary.

For black box modeling only the input and output relation is of interest. There is no need of knowing the internal physical dependencies of the system. However the system to be modeled must be existent in a physical way. Different experiments must be conducted and the the input and output data must be stored. With usage of this data the system‘s behavior can be discovered and reproduced [5]. For this purpose different system identification techniques are known, like parameter estimation, measuring the step response or Neural Networks.

Machine Learning algorithms learn the input-output relation by the use of the training data without any knowledge of the physical dependencies. So the ML based models are categorized as black box models.

Causal vs Acausal Modeling

The differentiation between causal and acausal modeling depends on whether the causal link relation can be changed during runtime. If there is a clear assignment from input value to output value which cannot be changed during simulation runtime, the model is a causal model. This kind of modeling is valid for conventional Simulink® blocks or programming languages like C or Python.

If there is no static assignment, but the causal link relation can change during simulation runtime, the model is an acausal model. For acausal simulation there exist languages like Modelica or the Simscape library, which is an extension for Simulink®.

Figure 10 shows the difference of causal and acausal models by modeling an inductance. The models are all based on the equation $u = L\frac{\mathrm{d}i}{\mathrm{d}t}$. The first figure is a causal model realized by conventional Simulink® blocks with voltage u as input and current i as output. The second figure shows a causal model also realized by conventional Simulink® blocks

with i as input and u as output. The third figure shows the acausal model using Simscape blocks in Simulink®, in which there is no assignment.

(a) Causal inductance model u/i (b) Causal inductance model i/u (c) Acausal inductance model

Figure 10: Different models of an inductance

As Machine Learning algorithms are trained with data that is explicitely described as input and output, it is only possible for ML based algorithms to perform causal simulation.

3.2. Machine Learning Methods

3.2.1. Categorization of Machine Learning Algorithms

The number of Machine Learning algorithms is almost as big as the number of problems they solve. So for the first step different types of Machine Learning are introduced to define the category of the algorithms used to solve the problem in this thesis.

Supervised vs Unsupervised Learning

Supervised learning means that during the training of the ML model the input and the associated output of the training data is used [1]. The allocated output data within a training data set is called labeled data. In this case the ML based algorithm is used for generalizing a known behavior of the application at hand. Unsupervised learning only uses the input data for training. This is mostly used to find patterns in the data.
As the goal of this thesis is to reproduce the behavior of a dynamic system supervised learning techniques will be used.

Online vs Offline Learning

A system applying online learning constantly uses new data for training of the algorithm. Hence the behavior of the ML based system can change and adapt to new data during its usage. Offline learning, also called batch learning, however, means that the Machine Learning algorithm is trained once, and after training the structure and parameters will not be changed anymore [10]. When new data is available, it might be retrained at a specific point of time, but the behavior of the system will not change during its usage.
In this thesis offline learning is used. The goal is to reproduce the dynamic system‘s behavior to use it for simulation. If the simulation model is used to adapt controllers or investigate the system in a composition of systems, there is a need for stable, deterministic

behavior of the simulation model.

Classification vs Regression

Classification is the prediction of predefined output classes with usage of the ML based algorithm. Whereas regression is the prediction of concrete (quasi-)continuous values [10]. Due to generalization of the model the output values of a regression models can estimate output values that have not been used as labels during training.
As the goal of this thesis is the prediction of the system‘s behavior, which means predicting certain output values according the input values, the ML models must solve regression problems.

3.2.2. General Approach for the Training of a ML Based Model

After a model has been chosen, it must be trained and optimized. A very common, iterative approach for the model optimization is the gradient descent method. For this method the parameters to be optimized need to be defined and initialized randomly. Additionally a loss function $J(\boldsymbol{\theta})$ needs to be defined. The loss function is a measure for defining the error of the model‘s output w.r.t. the actual labeled data, e.g. mean squared error (MSE) or root mean squared error (RMSE). So the gradient descent method can only be used for supervised learning applications.
The gradient descent method‘s task is to minimize the model‘s error by calculating the local gradient of the loss function and subtract it from the current parameter set, s. equation 10. A very important hyperparameter of the gradient descent method is the learning rate η. It determines the step size towards the minimum.
This calculation is done for every iteration. It should be stopped when the minimum of the loss function is reached.

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \tag{10}$$

A special type of the gradient descent method is the stochastic gradient descent method (SGD). Unlike the aforementioned approach, the SGD does not apply the optimization on every data on the training set during one iteration but only on a stochastically chosen subset.

3.2.3. Over- and Underfitting

Underfitting is defined as the model's inability to minimize the loss function w.r.t. the training data [30]. This means that regardless of the hyperparameter tuning and number of training iterations the loss function of the training data will not further decrease at a specific point.
Overfitting means that the learning model is too powerful. It fits the model according to the training set very well, but it lacks for generalization [30]. This means that the loss function for the training data set can be mimimized by the model. But the loss function for the test data set, which is data, that is not used for training, starts to increase at a certain point. Possible countermeasures are early stopping of the training when the validation loss increases or using regularization techniques [10].

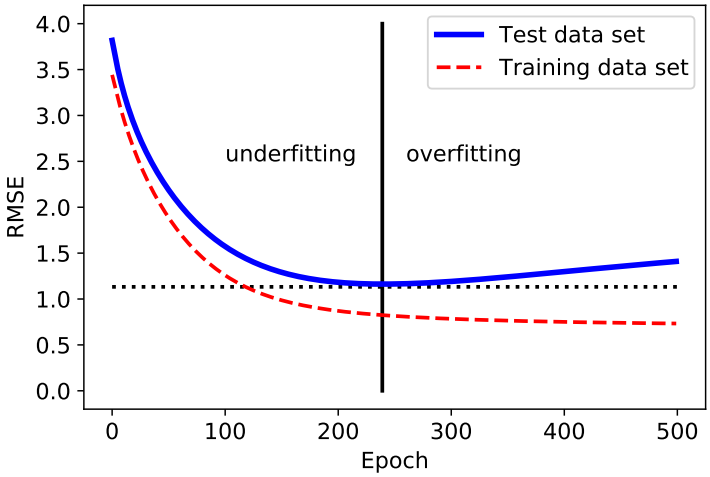


Figure 11: Over- and underfitting [10]

3.2.4. Introduction to Neural Networks

Neural Networks are mathematical entities, with the capability to learn, inspired by the neurons of the human brain [13]. The idea and implementation goes back to the perceptron introduced by Frank Rosenblatt in 1957. Since then the algorithms, also because of increasing computer performance, have been improved and adapted to fulfill different tasks. Hence it can be stated the Neural Networks today have very little resemblance to the human brain and can either be seen as a discipline of statistics [26]

3.2.4.1. Structure of Neural Networks

The basic structure of a Neural Network is the artificial neuron, also called perceptron [1], which can be seen in figure 12. It calculates the sum of the weighted inputs, takes this

result as input for the activation function and the outcome of this activation function is the output of the neuron. The neuron‘s input can be the output of neurons from the previous layer of neurons or it can be directly the training data.
Figure 13 shows a Neural Network. The weights are not explicitely drawn in this figure for the sake of simplicity. The Neural Network consists of one input layer, which only passes through the input data and additionally adds a bias neuron, and one output layer. Between these two layers one or more hidden layers can be placed. One layer consists of numerous neurons which are placed in parallel. Every layer, except the output layer, has one additional bias neuron with a static value, that is determined during the Neural Network‘s training.
The output of a complete layer is defined as:

$$\boldsymbol{Y} = \phi(\boldsymbol{X}\boldsymbol{W} + \boldsymbol{b}) \tag{11}$$

Figure 13 for instance can process 3 input values and 2 output values.

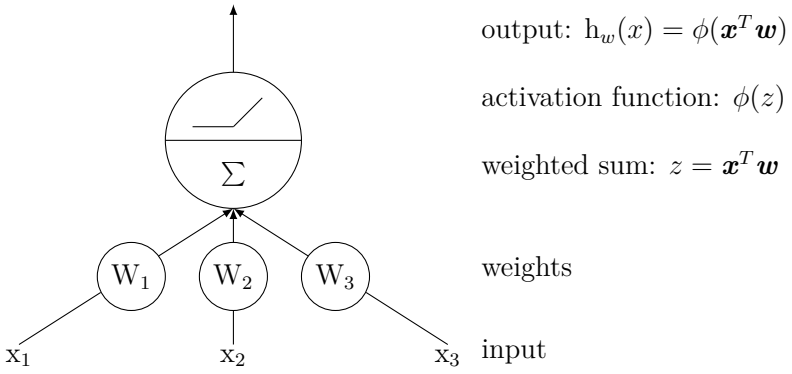


Figure 12: Structure of a single perceptron

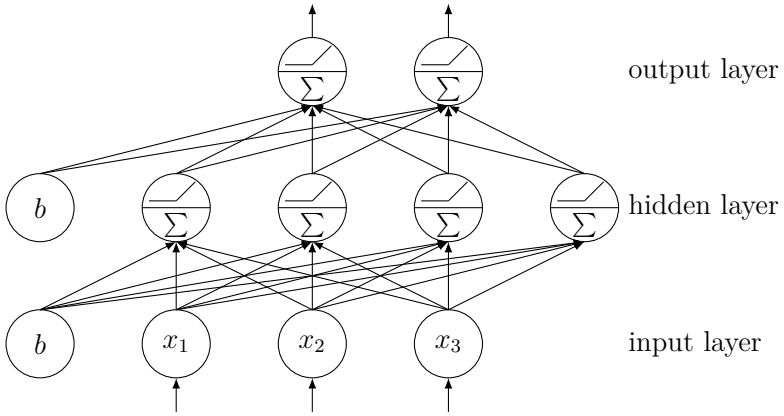


Figure 13: Fully connected Neural Network

The activation function of neurons within a Neural Network needs to be non-linear. Other-

wise the result of each neuron would be linear. If numerous linear functions are added the result would still be linear. So the benefit of having more than one layer would be lost [10].

In the above mentioned Neural Network the data flow is in one direction only, from input to output. These are so called Feedforward Neural Networks. But there are also neurons whose input is not only the output of the previous layer but also the output of the neuron itself from the previous step, s. figure 14. These neurons are called recurrrent neurons. Every recurrent neuron has two weights. One for its input from the previous layer and one for its output of the previous step [10]. Recurrent Neural Networks can be trained on input sequences.
The output of one layer of recurrent neurons can be calculated as:

$$\boldsymbol{Y}[k] = \phi(\boldsymbol{X}[k]\boldsymbol{W}_{\mathrm{X}} + \boldsymbol{Y}[k-1]\boldsymbol{W}_{\mathrm{Y}} + \boldsymbol{b}) \tag{12}$$

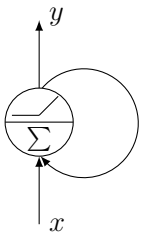


Figure 14: Recurrent neuron

3.2.4.2. Backpropagation

The backpropagation algorithm is used for the training of Neural Networks which means adjusting the weights in a manner so that the Neural Network behaves as intended. The backpropagation algorithm is based on the gradient descent method, which has been discussed in chapter 3.2.2.
Before this algorithm can be applied, the weights need to be initialized in a stochastical manner. Also, the whole batch (training data set) has to be split up in so called mini batches, e.g. batches of 512 data points [10]. The mini batches are then applied to the Neural Network. For each layer the dedicated output is calculated, saved and forwarded to the next layer. After having reached the output layer, the error of the calculated output to the labeled data is estimated [18]. After that, the backpropagation algorithm persues the Neural Network from the output layer backwards to the input layer. By applying the chain rule of calculus every connection‘s share in the total error is calculated [13]. After

having calculated the error of every connection the weights are adpated by applying the gradient descent method.
This calculation is done for every mini batch until all data points of the whole batch have been passed through the Neural Network. One run of all data point is called epoch. This can be repeated until the overall error is small enough or the model starts to overfit.

3.3. State of the Art

As pointed out above, this thesis uses Machine Learning algorithms for system identification and modeling. Especially the usage of Neural Networks for this purpose has been investigated during the last decades. In [16] it has been proven for the first time that Feedforward Neural Networks can approximate every function. After this theorem there have been publications that proof the capability of Feedforward Neural Networks to aproximate the behavior of linear and non-linear dynamic systems, [25], [24].
[22] and [21] show the capability of simple Recurrent Neural Networks to simulate linear and non-linear systems. In 1999, [23] showed that LSTM are able to approximate dynamic system behavior, not only when the LSTM is trained offline but also when it is trained online. Since then a lot of research for system identification has been conducted using LSTMs and special subtypes of LSTMs and Recurrent Neural Networks, e.g. [38], [34], [4],

The identification of the SOC is a special issue. This topic gained attention due to raising restriction on combustion engines and the new gained interest in electric vehicle. It has been shown that the coulomb counter approach failed in reliably estimating the battery‘s SOC. Also other approaches like the Extended Kalman Filter, which are more accurate than the coulomb counter, have their drawbacks due to their need for calibration. Hence, the investigation of the usage of Machine Learning algorithm for estimateing the SOC seemed to be promising. [8] has shown the ability of LSTM Neural Networks to estimate the SOC in a test bench under well defined conditions. [4] has used real driving cycle data for the SOC estimation. In this case also a LSTM Neural Network has been used and also the Neural Network outperformed the Extended Kalman Filter.

The thesis at hand takes the result of the aforementioned research and applies them explicitely on E/E components to investigate the possibility of simulating the behavior of a battery electric vehicle from the wheel to the battery. Thereby not only the ability for the simulation of one component is investigated but also the Machine Learning algorithms‘ ability to simulate a composition of components.

4. Model Conception

4.1. Selection of the Machine Learning Algorithm

In 1996 David Wolpert has proven in [36] that it is not possible to define a priori a ML algorithm that is best suitable for a given problem. He called this statement the No Free Lunch Theorem. In 1997 he extended this statement to the possible optimization methods [37]. These statements mean that the decsision whether an algorithm is suitable or not can only be made after it has been trained for the given problem and the result has been compared with other algorithms.
Nevertheless, the number of Machine Learning algorithms is manifold and a decision must be made which algorithms will be investigated during the course of this thesis. The ability of Neural Networks to simulate dynamic systems in general has been proven in multiple publications, s. also chapter 3.3. Also Neural Networks possess the ability to process more than one output, which not all Machine Learning algorithms are able to. Hence, for the further course of this thesis the upcoming investigations will be done by using Neural Networks.
However, the number of different types of Neural Networks is large. So a decision is necessary which types of Neural Networks shall be used to further evaluate the ability of simulating the E/E components described in chapter 2.
In [13] 2 fundamental architectures are distinguished, which have also been introduced in chapter 3.2.4:

- Feedforward Neural Network
- Recurrent Neural Network

According to [26] a single layer Neural Network can theoretically approximate every continuous function. Nevertheless it is advisable for non-linear functions and complex problems to have more than one layer [14]. So in this thesis Neural Networks with more than one layer will be investigated.
There are a lot of specialized sub-architectures of the aforementioned 2 fundamental types. For the Recurrent Neural Networks the LSTM and GRU are widely used. They have a longer memory than simple RNNs. Whereas at simple recurrent neurons the information of input data may get lost after some calls of the neuron, the LSTM and GRU cells have a longterm memory gate, which is independent of the number of calls of the neuron. They also have a forget gate, which decides which information shall be stored and which data has no impact and therefor it is not worthwile to be stored. GRU cells are newer than LSTMs, have a simpler structure than LSTMs and have the same learning capability as LSTMs and are even faster to train than LSTMs and so they have benefits regarding

simple recurrent neuron cells [10].
In the further course of this work simple FNNs as well GRUs[4] are used to investigate the differences between Neural Network with and without feedback and their ability to simulate E/E powertrain components.

4.2. Determination of the Data Representation

For the modeling concepts the data of interest must be determined. This means on the one hand that the input values as well as the output values of the model, which are in turn also the labels of the training set must be defined. On the other hand this also means that the data representation must be defined. This includes the defintion of the data‘s step size, the data‘s format, the norming of the data and in case of recurrent networks also the input data‘s sequence length.

Data Format
For a Neural Network it is important, that the input and output data are normed to values between 0 and 1 [10].
Furthermore the AC values, which are available between inverter and electric machine must be adapted. The three phase sinusoidal AC current must be represented in a manner that is suitable for Machine Learning algorithms. An AC value is not suitable for the correlation between input and output value. Therefore, instead of 3 sinusoidal current values, the amplitude of the currents, which is the same for all three currents, is used. The same is true for the phase voltage. The phase voltage, is a chopped PWM signal. This means that within one PWM cycle the voltage is either at battery voltage level or 0 V. However, the carrier signal is also a sinusoidal value. The sinusoidal value can be calculated by the on-time of the voltages per PWM cycle. With the knowledge of the sinusoidal carrier signal, the voltage amplitude can also be calculated. Figure 15 shows the PWM phase voltage and the calculated line-to-line voltage.
By only having the amplitude values one important information is missing. This is the phase shift between current and voltage as this varies depending on the electric machine‘s operation mode. It indicates whether the electric machine is in motor or regenerator mode or whether the electric machine is in field weakening or constant torque mode. This has an enormous impact as the higher AC current in field weakening can only be measured between inverter and electric machine but does not lead to a higher DC current consumption.
For calculating the phase shift, the instantaneous angles of the voltage and current is needed. The angle information can be calculated by using the 3 phase sinusoidal values.

[4] In the following chapters the term RNN is used, when defining and evaluating the GRU network

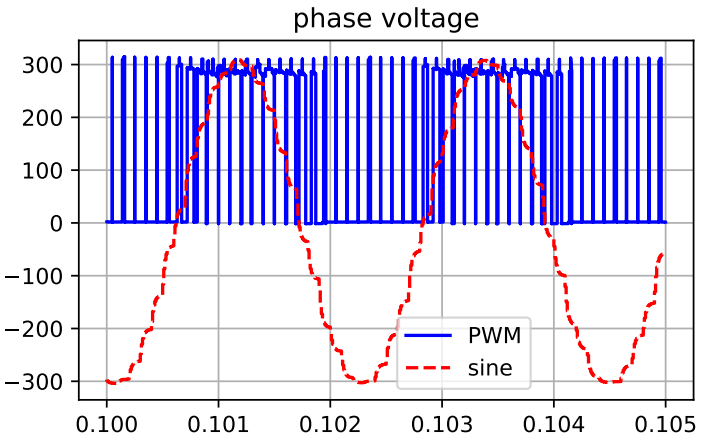


Figure 15: PWM phase voltage signal and calculated line-to-line voltage

With help of the clarke tranformation the current values can be calculated as a vector in a stator oriented frame with alpha and beta axis. With help of the the $\arctan_2$-operator the instantaneous angle of the current as well as the instantaneous angle of the voltage can be calculated by $\varphi_\mathrm{x} = \arctan_2(\frac{x_\beta}{x_\alpha})$
This calulcation must be done for the current angle φ_i as well as for the voltage angle φ_u. The phase shift φ between current and voltage is then calculated by $\varphi = \varphi_\mathrm{u} - \varphi_\mathrm{i}$.

Step Size of the Recorded Data

All the used data has been recorded during the same experiment runs. To be able to use the data for the different components the step size has been chosen the same for every component. The step size is chosen according the smallest change of the set values. In this case it is the PWM cycle of the inverter which controls the electric machine. So sample time is the same as the PWM cycle time. In the experiment at hand this constitutes 100 micro seconds.

Sequence Length of the Input Data for RNNs

Recurrent Neural Network possess the ability that their output does not only rely on the current input value but also on former input values. For Recurrent Neural Networks different input to output relations are possible, sequence-to-sequence, vector-to-sequence, sequence-to-vector. For this application the sequence-to-vector approach is used. This means that the input matrix is of the form $[batchsize, sequencelength, features]$ and the output matrix is of the form $[batchsize, features]$. Feedforward Neural Network have a input matrix of the form $[batchsize, features]$.
For the sequence-to-vector relation the available data must be adapted in a manner that the input values are arranged as sequences that must not overlap between different ex-

periment runs as this may distort the output. In picture 16 the approach of defining sequences with the sequence length of 5 steps using the data at hand is shown. The 5 input values are labeled with the output value that has the same number as the last value of the sequence.

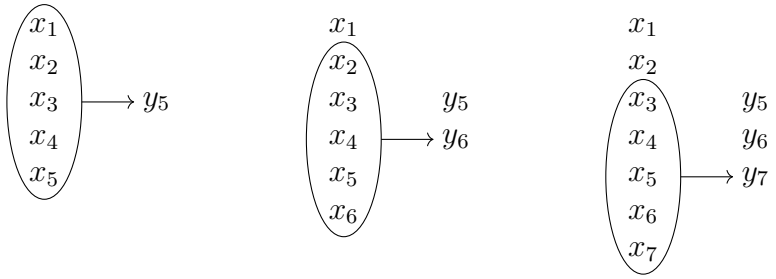


Figure 16: Sequence estimation

Code example 1 shows the implementation of the sequence estimation.

```
# define the paramters for the sequences
n_repitions = 180                          # number of calls (loops) of the simulation
n_steps = 5                                # define the number of steps within one sequence
n_max = len(Train_values) // n_repitions # the sequences consist of values of one simulation call
n_features = Train_values.shape[1:]        # define the number of features within the sequences
n_features = int(n_features[0])
print(n_max)


# estimate the Train_values
# ------------------------- X train values -------------------------
j = 1 # j defines the current simulation loop
i = 1 # i defines the current time steps

# split up in two while loops, as the the array needs to be defined and initialized in the first pass
while i <= ((j * n_max) - n_steps):
    sequence_start = i             # start the sequence at i
    sequence_end   = i + n_steps  # sequence starts at i + n_steps - 1
    aux_array      = Train_values[sequence_start:sequence_end,:]
    if i == 1:
        X_train_sequences = aux_array# in the first round of the loop create the array and initialize it
    elif i == 2:                        # first array sequence
        X_train_sequences = np.stack((X_train_sequences, aux_array)) # in the second loop stack the new
                                                                     #and the previous values
    else:
        X_train_sequences = np.concatenate((X_train_sequences, aux_array[None,:,:]))
    i = i + 1

j = 2
while j <= n_repitions:
    i = 1 + ((j-1) * n_max)
    while i <= ((n_max * j) - n_steps):
        sequence_start    = i             # start the sequence at i
        sequence_end      = i + n_steps  # sequence starts at i + n_steps - 1
        aux_array         = Train_values[sequence_start:sequence_end,:]
        X_train_sequences = np.concatenate((X_train_sequences, aux_array[None,:,:]))
        i = i + 1
    j = j + 1
```

Algorithm 1: Code example for defining the sequences

4.3. Determination of Input and Output Values

As discussed in chapter 3 Machine Learning based modeling is suitable for causal modeling only. So it must be determined which values are input and which are output values. In this thesis the behavior of the components and their composition from the mechanical values to the battery‘s SOC shall be simulated. This must be considered when defining input and output.

4.3.1. Interfaces of the Electric Machine

As described previously in this chapter the AC values must be adapted in a manner that the amplitude values of current and voltage $\hat{i}$, $\hat{u}$ and phase shift φ are used. These adapted AC values form the electric machine‘s output. The input values are the electrical torque M_{el} and the rotation speed n. In this case it does not matter whether the rotation speed is existent as rad/s or rpm, as long as the form is used persistently for the whole simulation. In this thesis the rotation speed is existent as rad/s.

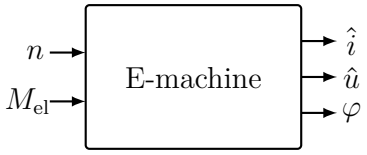


Figure 17: Inputs and outputs of the electric machine

4.3.2. Interfaces of the Inverter

The input of the inverter are the AC voltage and AC current values, whose format has been described previously in this chapter, namely $\hat{i}$, $\hat{u}$ and φ. The output values are the DC values of current I_{DC} and voltage U_{DC}.

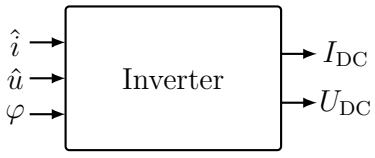


Figure 18: Inputs and outputs of the inverter

4.3.3. Interfaces of the Battery

In this tesis the impact of the temperature on the battery‘s SOC is neglected. So the input values for the battery model are the DC values of current and voltage, I_{DC} and U_{DC}. The output value of the battery model is the state of charge, SOC.

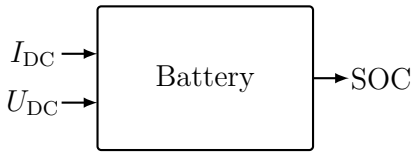


Figure 19: Inputs and outputs of the battery

4.3.4. Interfaces of the Composition of Inverter and Electric Machine

With the model which composites the electric machine and the inverter the DC values, I_{DC} and U_{DC} are directly simulated by using the mechanical values rotation speed n and electrical torque M_{el} as input.

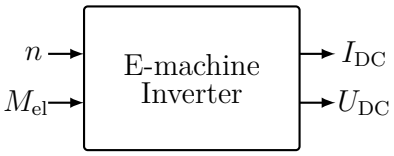


Figure 20: Inputs and outputs of the composition of electric machine and inverter

4.3.5. Interfaces of the Composite of Battery, Inverter and Electric Machine

The model of the composition of all three components estimates the SOC by using rotation speed n and electrical torque M_{el} as input values.

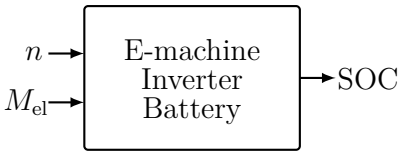


Figure 21: Inputs and outputs of the composition of electric machine, inverter and battery

5. Implementing the Machine Learning Based Simulation Models

After defining the interfaces the Neural Network models must be implemented, trained and optimized.

5.1. Code implementation

In this thesis Keras is used for implementing the Neural Networks. Keras is a high-level-deep-learning-API, which provides the possibility to access 3 open source deep learning libraries: Tensorflow, Theano or MXNET. For the further implementation Tensorflow is used as backend. By using tf.keras it is not only possible to use the Keras API but also further Tensorflow features.
In the following the code implementation of FNN and RNN by using Keras is presented.

```
model = keras.models.Sequential()
model.add(keras.layers.InputLayer(input_shape=inputshape))
layer = 1
while(layer <= n_hidden):
    model.add(keras.layers.Dense(n_units, activation=activator, kernel_initializer=initializer))
    layer = layer + 1
model.add(keras.layers.Dense(int(index_output)))
model.summary()
```

Algorithm 2: Structure for implementing a Feedforward Neural Network with Keras

```
model = keras.models.Sequential()
model.add(keras.layers.InputLayer(input_shape=inputshape))
if (n_hidden > 1):
    layer = 1
    while layer <= (n_hidden - 1):
        model.add(keras.layers.GRU(n_units, activation=activator, kernel_initializer=initializer,
                                   return_sequences=True))
        layer = layer + 1;
model.add(keras.layers.GRU(n_units, activation=activator, kernel_initializer=initializer))
model.add(keras.layers.Dense(int(index_output)))
```

Algorithm 3: Structure for implementing a Recurrent Neural Network with Keras

It can be seen, that both Neural Networks have a dedicated input layer **`keras.layers.InputLayer(input_shape=inputshape)`** and a dedicated output layer **`model.add(keras.layers.Dense(int(index_output)))`**. As described in chapter 3 the input layer defines the number of the attributes of the input matrix. It does not process any computation on the data. The output layer defines the number of attributes of the output data. Also for the recurrent Neural Network the last layer has been chosen to be of the form **`keras.layers.Dense`**, which means that there is no feedback in the neurons of this layer. This has two advantages over the usage of a recurrent layer as output layer. First, if the relation between the number of neurons between the other layers and the last layer is big, then the hidden state of the recurrent neurons will hardly be used, but still the training

runtime will take longer than a regular Dense-layer and second it also provides the ability to use other activation functions than tanh [10]. For this purpose in the last hidden layer the attribute **return_sequences** is not set to **True**. This means that the hidden state is not available for being used at the next layer. As the output layer is a normal Dense-layer this is necessary in this case.

5.2. Training of the models

After the code has been implemented. The model can be compiled and trained, s. algorithm 4. For being able to be compiled the optimizer as well as the learning rate must be defined. For a successful training the hyperparameters must be defined, which will be discussed in the following chapter.

```
1 optimizer = tf.keras.optimizers.SGD(clipnorm = 1.0)
2 optimizer.learning_rate.assign(learning_rate)
3 model.compile(loss=loss_function, optimizer=optimizer)
4 model.fit(X_train_sequences1, Y_train_sequences1, epochs=max_epochs, batch_size = batchsize)
```

Algorithm 4: Compiling and training of the model

As the RNN is more prone to exploding gradients the attribute **clipnorm = 1.0** is used. This means that the gradient is limited to 1.0 with retention of the gradient‘s direction. The usage of clipnorm has to be shown to be not necessary for FNN.

5.2.1. Training Data

The training data consists of 180 different runs of the whole powertrain. During these runs the necessary input and output data of all three components have been logged.
One part of the runs has been conducted in motor operation, with jumps of the reference torque value from zero to different final values. Following parameters have been varied and different combination of them have been used for the runs in motor operation:

- the reference torque value
- the mechanical load, which can be described as a quadratic function with an offset $a \cdot \omega^2 + n$. The offset n as well as the factor a have been varied.
- the battery‘s SOC

The other part of the runs has been conducted in regeneration mode. At the beginning of these runs the electric machine is driven with an initial speed and with a torque request of zero. After 20 ms a negative reference torque jump is conducted. These parameters have been varied during the runs in regeneration mode:

- the reference torque value
- the initial speed
- the battery‘s SOC

5.2.2. Model Optimization

In the previous code examples the hyperparameters have not been explicitely defined. The possible hyperparameters and their combination are manifold.
For finding the best combination of the hyperparameters, there are several possibilities like grid search, bayes search, random search or the usage of predefined libraries like Keras Tuner, Hyperopt, etc, [10].
The hyperparameters that are chosen to be varied are listed in table 1

hyperparameter	**varied values**	**architecture**
number of hidden layers	2, 3, 4	RNN and FNN
number of cells per layer	10, 20, 30	RNN and FNN
activation function	selu, elu	FNN
optimizer	SGD, Adam	RNN and FNN
learning rate	0.001, 0.0001	RNN and FNN
initialization	lecun normal, he normal	FNN
length of sequence	5, 10, 20	RNN

Table 1: Hyperparameters to be varied

The algorithm 5 shows the definition of the hyperparameters in the code for optimizing the FNN, the implementation for RNN is similar to the FNN. Therefore Tensorboard is used. Tensorboard is Tensorflow‘s visualization toolkit. The Hparams-plugin of Tensorboard enables definition of the hyperparameters with visualization and storage of the results.

```
1  # define the hyperparameters to be modified
2  HP_NUM_UNITS     = hp.HParam('n_units', hp.Discrete([10, 20, 30]))
3  HP_NUM_LAYERS    = hp.HParam('n_hidden', hp.Discrete([2, 3, 4]))
4  HP_LERN_RATE     = hp.HParam('learning_rate', hp.Discrete([1e-3, 1e-4]))
5  HP_ACTIVATOR     = hp.HParam('activator', hp.Discrete(['selu', 'elu']))
6  HP_INITIALZIZER = hp.HParam('initializer', hp.Discrete(['he_normal', 'lecun_normal']))
7  HP_OPTIMIZER     = hp.HParam('optimizer', hp.Discrete([1, 2])) # 1 = SGD, 2 = adam
8
9  METRIC_LOSS = 'loss'
10 METRIC_ACCURACY = 'accuracy'
11 # define the storage place of the metrics
12 with tf.summary.create_file_writer(root_logdir).as_default():
13     hp.hparams_config(
14         hparams=[HP_NUM_UNITS, HP_NUM_LAYERS, HP_LERN_RATE, HP_ACTIVATOR, HP_INITIALZIZER, HP_OPTIMIZER]
15         metrics=[hp.Metric(METRIC_LOSS, display_name='Loss'), hp.Metric(METRIC_ACCURACY,
16                   display_name='Accuracy')],)
```

Algorithm 5: Definition of the hyperparameters to be varied

In this thesis grid search has been conducted. Grid search means that all combinations of predefined hyperparameters are used for training and after training the outcoming model of each combination is evaluated against a test data set which has not been used for training. For enabling the usage and variation of the aforementioned hyperparameters the algorithm 2 needs to be adapted to algorithm 6.

```
def build_model(hparams):
    early_stopping_cb = keras.callbacks.EarlyStopping(patience = 20, restore_best_weights = True)
    # build the model
    model = keras.models.Sequential()
    model.add(keras.layers.InputLayer(input_shape=inputshape))
    layer = 1
    while(layer <= hparams[HP_NUM_LAYERS]):
        model.add(keras.layers.Dense(hparams[HP_NUM_UNITS], activation=hparams[HP_ACTIVATOR],
                  kernel_initializer=hparams[HP_INITIALZIZER]))
        layer = layer + 1
    model.add(keras.layers.Dense(int(index_output)))
    # define the optimizer
    if hparams[HP_OPTIMIZER] == 1:
        optimizer = keras.optimizers.SGD()
    elif hparams[HP_OPTIMIZER] == 2:
        optimizer = keras.optimizers.Adam()
    optimizer.learning_rate.assign(hparams[HP_LERN_RATE])
    model.summary()
    #compile the model
    model.compile(loss=loss_function, optimizer=optimizer, metrics=['accuracy'])
    # train the model
    model.fit(X_train1, Y_train1, epochs=max_epochs, batch_size = batchsize,
              validation_data=(X_valid, Y_valid), callbacks=[early_stopping_cb])
    loss, accuracy = model.evaluate(X_test, Y_test)
    return accuracy, loss
```

Algorithm 6: Adapted code for using the hyperparameters to be varied

Algorithm 7 shows the needed implementation to conduct the grid search in an automated way.

```
def run(run_dir, hparams):
    with tf.summary.create_file_writer(run_dir).as_default():
        hp.hparams(hparams)  # record the values used in this trial
        accuracy, loss = build_model(hparams)
        tf.summary.scalar(METRIC_ACCURACY, accuracy, step=1)
        tf.summary.scalar(METRIC_LOSS, loss, step=1)
# run the model with the different hyperparameter settings
session_num = 0
for n_units in HP_NUM_UNITS.domain.values:
    for n_hidden in HP_NUM_LAYERS.domain.values:
        for learning_rate in HP_LERN_RATE.domain.values:
            for activator in HP_ACTIVATOR.domain.values:
                for initializer in HP_INITIALZIZER.domain.values:
                    for optimizer in HP_OPTIMIZER.domain.values:
                        hparams = {
                            HP_NUM_UNITS     : n_units,
                            HP_NUM_LAYERS    : n_hidden,
                            HP_LERN_RATE     : learning_rate,
                            HP_ACTIVATOR     : activator,
                            HP_INITIALZIZER  : initializer,
                            HP_OPTIMIZER     : optimizer,
                        }
                        run_name = "run-%d" % session_num
                        run(root_logdir + '/' + run_name, hparams)
                        session_num += 1
```

Algorithm 7: Automated conduction of the grid search

With the defined algorithms not only the correctness of the model is ensured, but also the model's ability for generalization. This come from the definition of the the early stopping callback in Keras. The trainig data is split in training data and validation data, which is also not used for training. If the validation loss does not improve after 20 cycles the training will be stopped and the best weights are stored, **`keras.callbacks.EarlyStopping( patience = 20, restore_best_weights = True)`**. This avoids under- and overfitting.

The initialization and the activation function of the RNN is not to be varied for the grid search. The reason for this is that unlike the FNN, the RNN needs a saturating activation function to prevent the occurance of exploding gradients. So it is not possible to use the elu, selu or relu activation function in the hidden layers. According to [10] the best initialization for a saturating activation is the initialization after Glorot. Another, more practical, reason is that in Keras the GRUs are optimized for the initialization after Glorot and the tanh activation function for the processing on a GPU.
In the following some of the hyperparameters are described shortly:

Activation function
For the FNN non-saturating activation functions are used in the grid search. These activation functions are useful for the prevention of vanishing gradients. The most popular function is the so called RELU (rectified linear unit) function. But as RELU function sets the output to zero for negative values this can lead to dying neurons.

$$\mathrm{RELU}(z) = \begin{cases} 0 & \text{if } z < 0 \\ z & \text{if } z \geq 0 \end{cases} \tag{13}$$

A possibility to avoid this problem is the so called ELU (exponential linear unit) function. It has the benefit that negative values will lead to negative output values and also the derivation is unlike zero for negative values. So it will not lead to dying neurons. It also possesses the property that due to the negative values the mean of the activations is pushed closer to zero. Mean activations that are closer to zero enable faster learning [9].

$$\mathrm{ELU}(z) = \begin{cases} \alpha(\mathrm{e}^z - 1) & \text{if } z < 0 \\ z & \text{if } z \geq 0 \end{cases} \tag{14}$$

The SELU (scaled exponential linear function) improves the ELU in the way that it leads

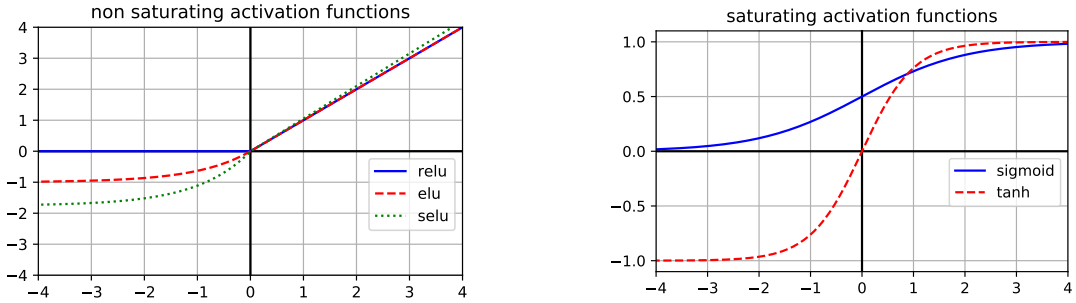


Figure 22: Comparison of different activation functions

to zero mean unit variance [20].

$$\text{SELU}(z) = \lambda \begin{cases} \alpha(e^z - 1) & \text{if } z < 0 \\ z & \text{if } z \geq 0 \end{cases} \tag{15}$$

As described above RNNs need saturating activation functions. Tanh has been proven to be the most suitable function as it allows negative output values, in contrast to the sigmoid function. Figure 22 shows the aforementioned activation functions.

Initialization

As described in chapter 3 for the backpropagation algorithm the neurons must be stochastically initialized. For a long time Feedforward Neural Network have been hard to train. This has been a result of the stochastic initialization of the weights with a mean of 0 and variance of 1. In [11] the authors prove that the variance must depend on the number of units of the layers. With n_i = number neurons of the layer i and $n_{\text{i}+1}$ = number of neurons of the following layer following variance must be applied for the weights for the layer i for the initialization after Glorot:

$$\sigma^2 = \frac{2}{n_\text{i} + n_{\text{i}+1}} \tag{16}$$

According to Kaiming He, the variance of the weight initialization must be:

$$\sigma^2 = \frac{2}{n_\text{i}} \tag{17}$$

The initialization after Lecun demands a variance of:

$$\sigma^2 = \frac{1}{n_\text{i}} \tag{18}$$

All three initializations have slightly different variances but they have all in common that a mean of 0 is required.

Optimizer

As described in chapter 3 the stochastic gradient descent (SGD) algorithm subtracts the derivative of the loss function multiplied by the learning rate from the current weight, s. equation 10. It does not care about the former gradient‘s size, which can lead to slow computation when the gradients are small [10].

The ADAM (adaptive moment estimation) optimizer also takes into account the size of the former gradients. It updates exponential moving averages of the gradient $\mathbf{m}$ and the squared gradient $\mathbf{s}$ where the hyperparameters β_1 and $\beta_2 \in [0, 1)$ control the exponential decay rates of these moving averages[19].

$$\mathbf{m} \leftarrow \beta_1 \mathbf{m} - (1 - \beta_1) \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \tag{19}$$

$$\mathbf{s} \leftarrow \beta_2 \mathbf{s} - (1 - \beta_2) \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \otimes \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \tag{20}$$

$$\hat{\mathbf{m}} \leftarrow \frac{\mathbf{m}}{1 - \beta_1^t} \tag{21}$$

$$\hat{\mathbf{s}} \leftarrow \frac{\mathbf{s}}{1 - \beta_2^t} \tag{22}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \hat{\mathbf{m}} \oslash \sqrt{\hat{\mathbf{s}} + \varepsilon} \tag{23}$$

5.2.3. Results of the Hyperparameter Optimization

In the following the best hyperparameters, that have been found during the model optimization are presented.

For every component it can be seen that the Adam Optimizer outperforms the SGD optimizer. This is a result of the adaptive first and second moment. As the step size depends on the sliding average of the last gradient values the parameters converge faster and more stable to the optimum. The SGD does not adapt the step size. Especially when the signal to noise ratio becomes smaller high gradients, that are not filtered, lead to problem when converging to the optimum.

It can also be seen, that there is a trend to use less neurons per hidden layer but more hidden layers to be able to simulate the components. Regarding the other hyperparameters it is difficult to chose one variant as superior over the others.

	units	**layers**	**learning rate**	**optimizer**[5]	**activator**	**initializer**	**seq. length**
Electric machine							
RNN	10	2	0.001	2	tanh	Glorot	10
FNN	30	2	0.001	2	elu	lecun normal	–
Inverter							
RNN	30	3	0.0001	2	tanh	Glorot	20
FNN	20	4	0.0001	2	selu	he normal	–
Battery							
RNN	30	2	0.001	2	tanh	Glorot	5
FNN	20	3	0.001	2	elu	lecun normal	–
Composition of electric machine and inverter							
RNN	10	3	0.0001	2	tanh	Glorot	5
FNN	10	4	0.001	2	selu	lecun normal	–
Composition of electric machine, inverter and battery							
RNN	10	3	0.0001	1	tanh	Glorot	5
FNN	30	4	0.0001	1	selu	he normal	–

Table 2: Best results of the hyperparameter optimization

In Appendix E the best 20 hyperparameter combinations for every component and every architecture are shown.

[5] 1 = SGD, 2 = ADAM

6. Evaluation of the Machine Learning Based Simulation Models

In the following the results of the Machine Learning based simulation models are discussed. For the evaluation the results of the FNN and RNN models, that have been trained with the hyperparameters described in chapter 5.2.3, are compared with a set of test data, that has not been used for the training of the Neural Networks.

6.1. Definition of the Error Measures

Whereas the loss function for the training and validation has been mean squared error (MSE), in this chapter the error of the final results will be displayed as root mean squared error (RMSE) and mean absolute error (MAE) and their percentage representations.
The RMSE‘s calculation is shown in equation 24. The RMSE displays the error of the predicted value in its physical unit. To enable the comparison between different models and different output values a relative RMSE is used. Due to the RMSE‘s property of weighting outliners very strong the root mean squared percentage error RMSPE can not been used. This weighting distorts the RMSPE in a way, that no clear statement can be made based on this value for the models used in this thesis. Hence, for the further investigation the relation of the RMSE to the sum of the squared original y values is used. In the following this will be called root mean squared relation error (RMSRE), s. equation 25.

$$\mathrm{RMSE}(\hat{y}) = \sqrt{\frac{1}{N}\sum_{n=1}^{N}((y(n) - \hat{y}(n))^2)} \tag{24}$$

$$\mathrm{RMSRE}^{6}(\hat{y}) = 100 \cdot \sqrt{\frac{\sum_{n=1}^{N}(y(n) - \hat{y}(n))^2}{\sum_{n=1}^{N} y(n)^2}} \tag{25}$$

As stated above a further measure used for the evaluation of the models‘ performance is MAE, shown in equation 26. MAE displays the error in the output value‘s physical unit. Also for this measure a percentage value can be described, the mean absolute percentage

[6]The multiplication of 100 for MAPE and RMSRE is needed to get a percentage value between 0 % and 100 %.

error (MAPE). The MAPE's calculation is shown in equation 27.

$$\mathrm{MAE}(\hat{y}) = \frac{1}{N}\sum_{n=1}^{N}|y(n) - \hat{y}(n)| \tag{26}$$

$$\mathrm{MAPE}(\hat{y}) = 100 \cdot \frac{1}{N}\sum_{n=1}^{N}\left|\frac{y(n) - \hat{y}(n)}{y(n)}\right| \tag{27}$$

According to [10] the RMSE should be the first choice for regression problems. This results from the fact that RMSE performs very good for errors following a normal distribution [7]. In most regression problems this error distribution is assumed.
But due to the square calculation, the RMSE weights large errors higher than the MAE does. If a large number of large outliners is expected, this higher weighting could distort the RMSE. In this case the MAE is advantageous. The MAE also performs better for uniformly distributed errors [7]. To be able to describe the deviations of the models regardless their error distribution, both metrics incl. their percentage representations will be exerted.

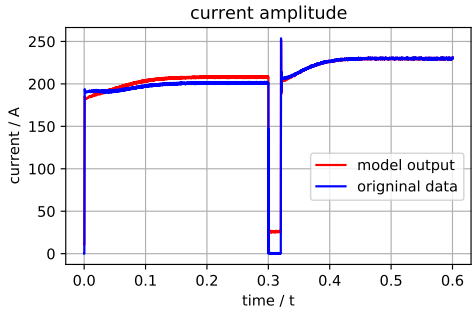
6.2. Test Data

The test data consists of 2 runs, one in motor and one in regeneration mode. The way how the runs are conducted are similar to the way the runs for the training data have been conducted.
For the run in motor operation a jump of the torque reference value from 0 to 35 Nm is conducted. This leads to an acceleration of the electric machine from standstill to 337 rad/s. The mechanical load of the electric machine during this run can be expressed as $12\mathrm{Nm} + 0.0002\mathrm{Nms}^2 \cdot \omega^2$.
For the run in regeneration mode the electric machine is initially running with a rotation speed of 600rad/s without any electrical operation. After 20 ms there is a jump of the reference torque from 0 to -40 Nm, but the rotation speed remains 600 rad/s. This leads to a regeneration power of 18 kW.

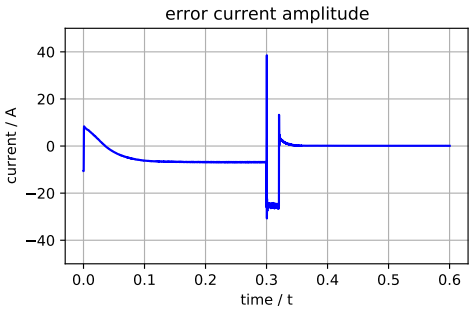
6.3. Results of the Electric Machine Models

In this chapter the results for the electric machine models are discussed. As described in chapter 4.3.1 the output of the electric machine model is the phase current's amplitude $\hat{i}$, the phase voltage's amplitude $\hat{u}$ and the phase shift φ.

The FNN shows a better performance for $\hat{i}$ and $\hat{u}$, whereas the performance for the phase shift is a bit better when using RNN. However the overall performance is better when using FNNs for simulating the electric machine. The RMSRE is 1.42 times as big when using RNN compared to FNN and the MAPE is 268 times as big.

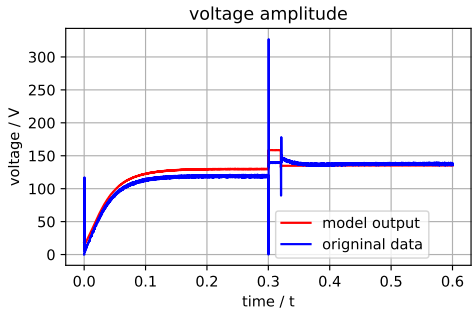


(a) Comparison of original data and model output

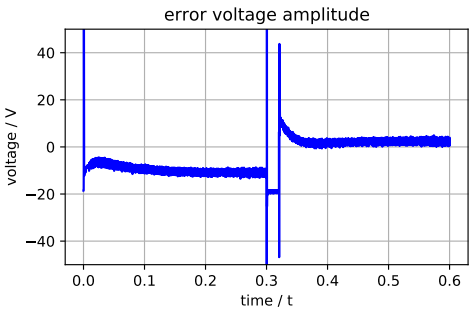


(b) Error of the modeled output

Figure 23: AC current of the electric machine modeled via a Feedforward Neural Network

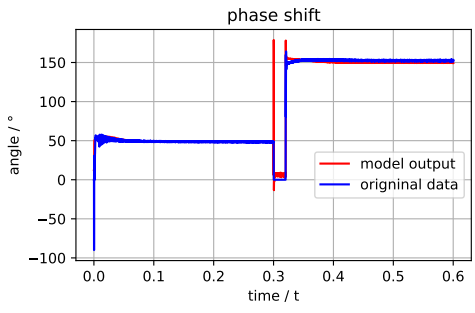


(a) Comparison of original data and model output

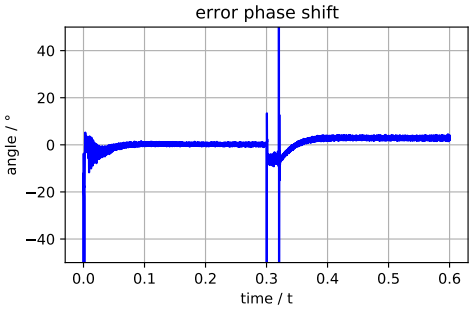


(b) Error of the modeled output

Figure 24: AC voltage of the electric machine modeled via a Feedforward Neural Network



(a) Comparison of original data and model output

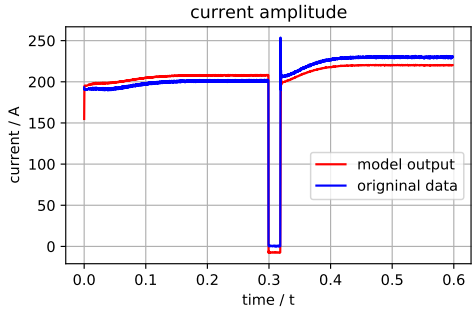


(b) Error of the modeled output

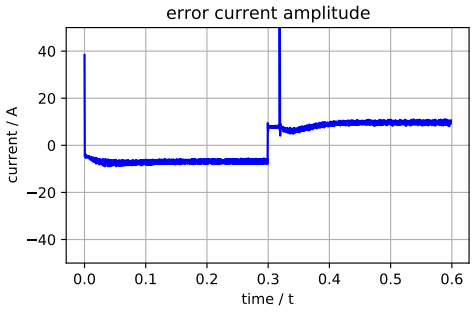
Figure 25: Phase shift of the electric machine modeled via a Feedforward Neural Network

	FNN			**RNN**		
	$\hat{i}$	$\hat{u}$	φ	$\hat{i}$	$\hat{u}$	φ
RMSE	6.38 A	9.38 V	5.23 °	9.19 A	13.14 V	4.58 °
RMSRE	3.06 %	7.57 %	4.76 %	4.40 %	10.60 %	4.16 %
Overall RMSRE	3.62%			5.23%		
MAE	3.89 A	6.74 V	2.13 °	8.08 A	12.13 V	1.32 °
MAPE	$3.18 \cdot 10^{-3}$%	$9.25 \cdot 10^{-3}$%	$3.71 \cdot 10^{-3}$%	4.25 %	$16.68 \cdot 10^{-3}$%	$1.81 \cdot 10^{-3}$%
Overall MAPE	$5.38 \cdot 10^{-3}$%			1.42%		

Table 3: Error of the electric machine simulation models

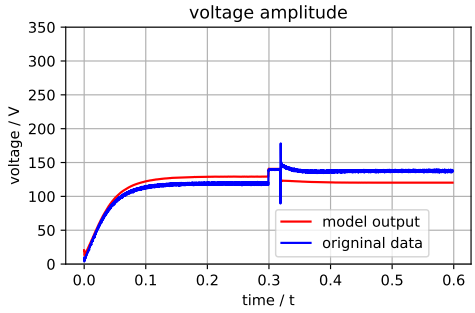


(a) Comparison of original data and model output

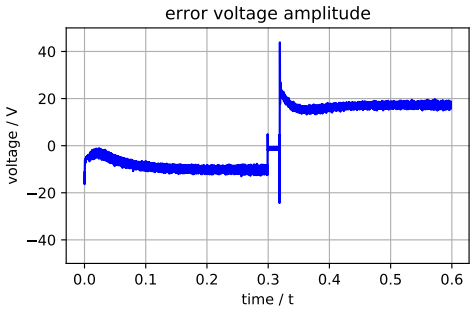


(b) Error of the modeled output

Figure 26: AC current of the electric machine modeled via a Recurrent Neural Network

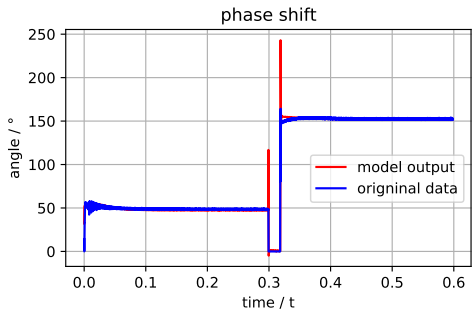


(a) Comparison of original data and model output

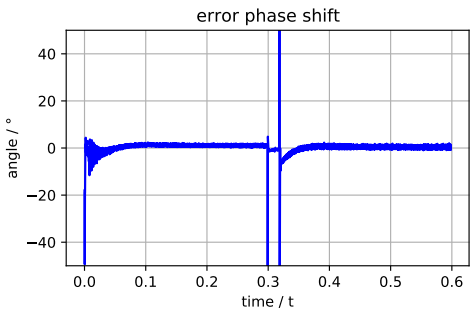


(b) Error of the modeled output

Figure 27: AC voltage of the electric machine modeled via a Recurrent Neural Network



(a) Comparison of original data and model output



(b) Error of the modeled output

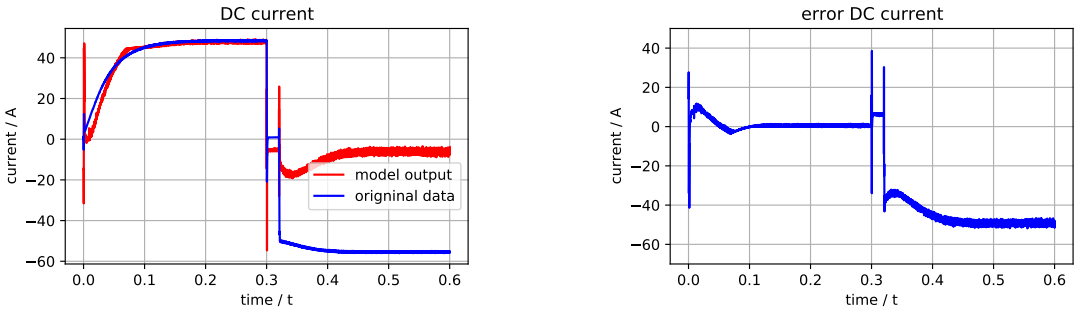
Figure 28: Phase shift of the electric machine modeled via a Recurrent Neural Network

6.4. Results of the Inverter Models

In the following the inverter models‘ results are discussed. The output of the inverter models is the DC current I_{DC} and the DC voltage U_{DC}, s. chapter 4.3.2.
The DC current, that is simulated by the FNN model during regenration, shows a big deviation from the original data. During regeneration the error is up to 50 A, s. figure 36. This can also be seen in the RMSRE of 65.19 % and the MAPE of 11 %. The DC voltage fits very well to the original data and shows only a RMSRE of 2.83 % and a MAPE of $3.87 \cdot 10^{-3}$%.
The RNN, however, is capable of simulating the inverter. The current has only a RMSRE of 6.18 % and a MAPE of 6.5 % and the DC voltage has a RMSRE of 2.05 % and a MAPE $3.41 \cdot 10^{-3}$%, which is even better than the DC voltage of the FNN.
Due to the FNN‘s inability to appropriately reproduce the DC current during regeneration, the FNN model that has been trained for this thesis fails for this application. The overall RMSRE of the model is 34.01 % and an overall MAPE of 5.7 %. The RNN model has an overall RMSRE 4.11 % and a MAPE of 3.28 %.

	FNN		**RNN**	
	I_{DC}	U_{DC}	I_{DC}	U_{DC}
RMSE	31.54 A	8.98 V	3.0 A	6.50 V
RMSRE	65.19 %	2.83 %	6.18 %	2.05 %
Overall RMSRE	34.01%		4.11%	
MAE	22.39 A	6.74 V	2.79 A	6.44 V
MAPE	11%	$3.87 \cdot 10^{-3}$%	6.5 %	$3.41 \cdot 10^{-3}$%
Overall MAPE	5.7%		3.28%	

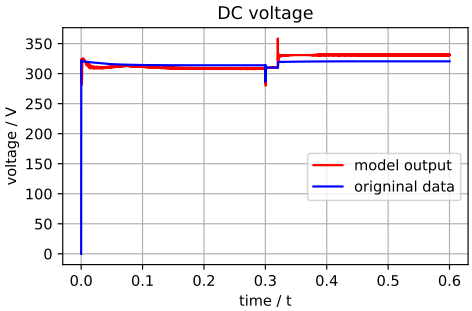
Table 4: Error of the inverter simulation models



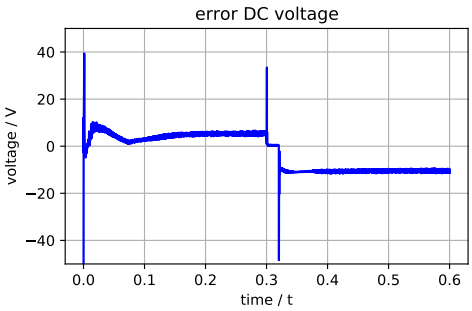
(a) Comparison of original data and model output

(b) Error of the modeled output

Figure 29: DC current of the inverter modeled via a Feedforward Neural Network

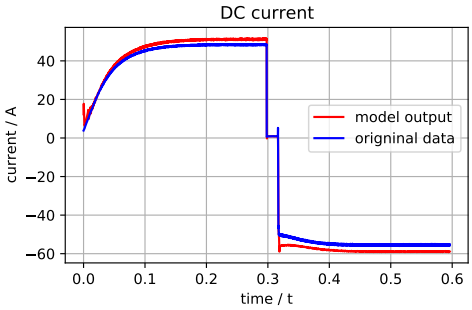


(a) Comparison of original data and model output

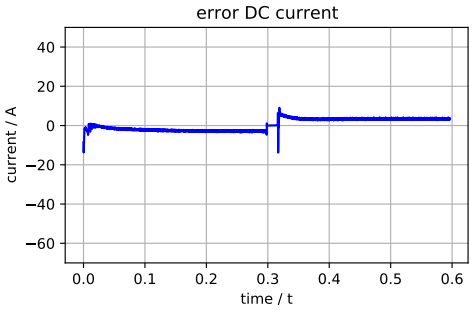


(b) Error of the modeled output

Figure 30: DC voltage of the inverter modeled via a Feedforward Neural Network

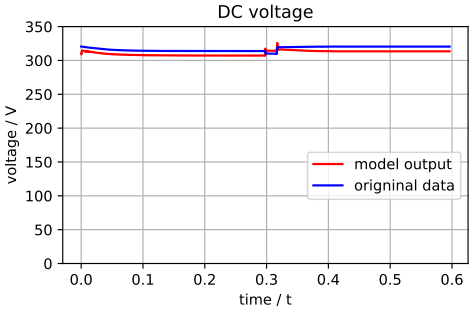


(a) Comparison of original data and model output

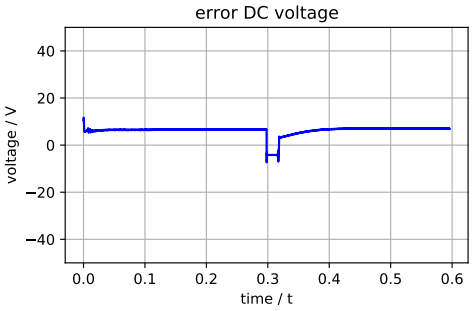


(b) Error of the modeled output

Figure 31: DC current of the inverter modeled via a Recurrent Neural Network



(a) Comparison of original data and model output



(b) Error of the modeled output

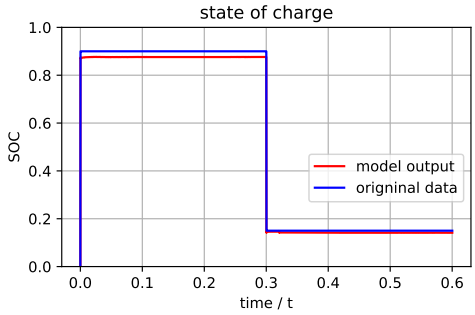
Figure 32: DC voltage of the inverter modeled via a Recurrent Neural Network

6.5. Results of the Battery Models

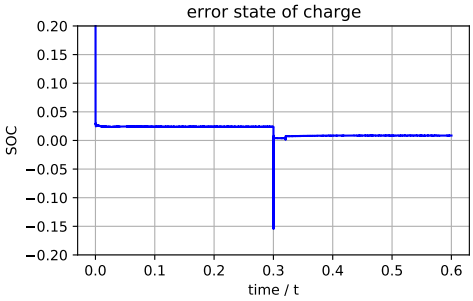
The SOC of the two battery models and their precision is discussed in this chapter. Both models, RNN as well as FNN, reproduce the original data in a very precise manner. Nevertheless the RNN's RMSRE is about 1.86 % better than the FNN's RMSRE. The MAPE is only very little better, 0.03%, when using RNN compared to FNN, but the MAE shows, as the RMSRE and the RMSE, a clearly better performance of the RNN. The FNN has a deviation up to 0.025 during the motor operation, s. figure 33.

	FNN	**RNN**
	SOC	SOC
RMSE	0.0244	0.012
RMSRE	3.78 %	1.92 %
MAE	0.016	0.010
MAPE	4.06 %	4.03 %

Table 5: Error of the battery simulation models

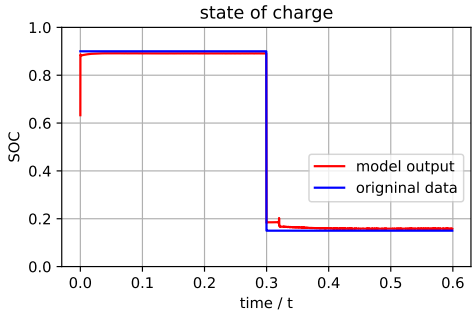


(a) Comparison of original data and model output

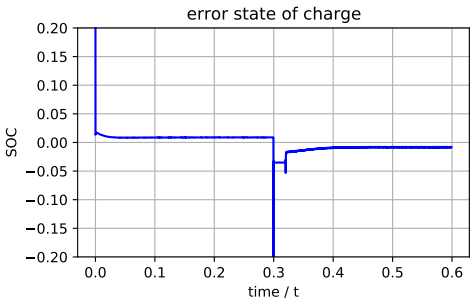


(b) Error of the modeled output

Figure 33: State of Charge value of the battery modeled via a Feedforward Neural Network



(a) Comparison of original data and model output



(b) Error of the modeled output

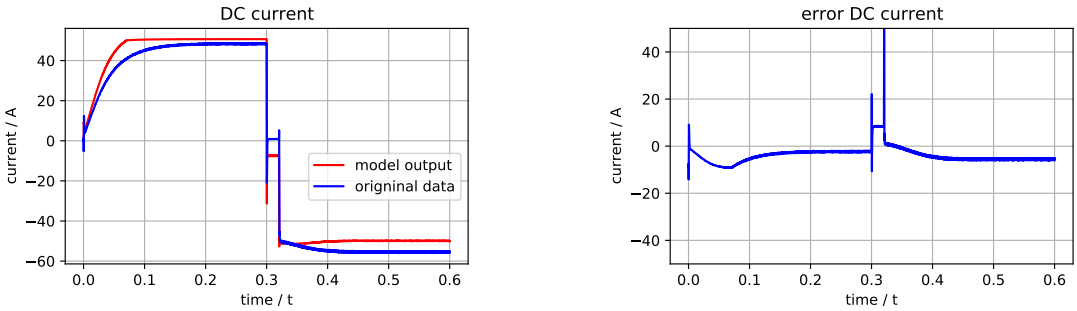
Figure 34: State of Charge value of the battery modeled via a Recurrent Neural Network

6.6. Results of the Models of the Composition of Electric Machine and Inverter

In this chapter the results for the models of the composition of the electric machine and inverter are shown. As stated in chapter 4.3.4 the output of these models is the DC current I_{DC} and the DC voltage U_{DC}. This composition is intended to deliver the electrical output values directly from the mechanical input values, rotation speed n and torque M_{el}. The FNN's DC current is closer to the original value than it has been for the plain inverter model. But still it has a RMSRE value of 10.4 %. The RNN model as well as the FNN model have an overall RMSRE greater than 7 % which is worse than the single components' RMSRE, except the FNN inverter model. The overall MAPE values, RNN 4.22 % and FNN 6.2 %, are also worse than the ones of single components
These RMSRE and MAPE values are a result of the missing sensitivity of the DC output values' relation w.r.t. the mechanical input values. For further discussion s. chapter 6.8.2.

	FNN		**RNN**	
	I_{DC}	U_{DC}	I_{DC}	U_{DC}
RMSE	5.04 A	17.11 V	6.31 A	5.33 V
RMSRE	10.4 %	5.39 %	13.03 %	1.68 %
Overall RMSRE	7.9%		7.35%	
MAE	4.43 A	12.86 V	5.03 A	5.22 V
MAPE	12.39%	$6.72 \cdot 10^{-3}$%	8.4 %	$2.75 \cdot 10^{-3}$%
Overall MAPE	6.2%		4.22%	

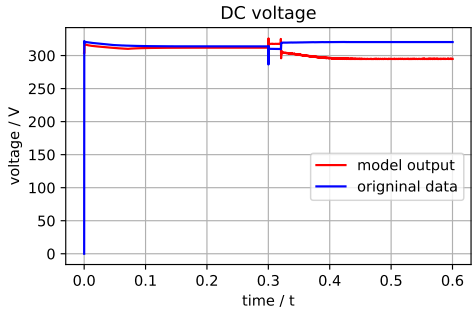
Table 6: Error of the electric machine + inverter simulation models



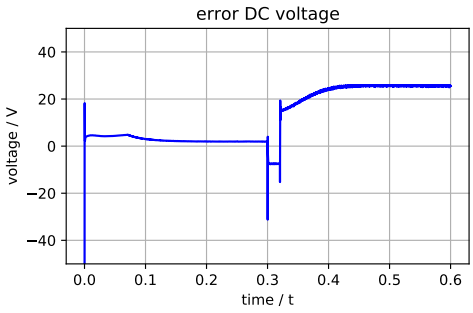
(a) Comparison of original data and model output

(b) Error of the modeled output

Figure 35: DC current of the composition of electric machine and inverter modeled via a Feed-forward Neural Network

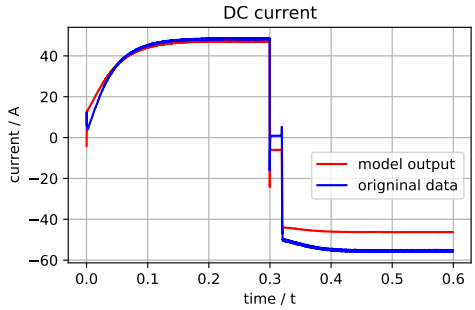


(a) Comparison of original data and model output

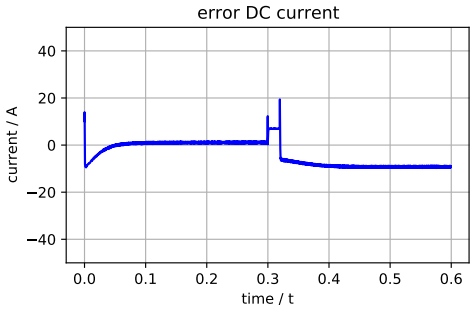


(b) Error of the modeled output

Figure 36: DC voltage of the composition of electric machine and inverter modeled via a Feed-forward Neural Network

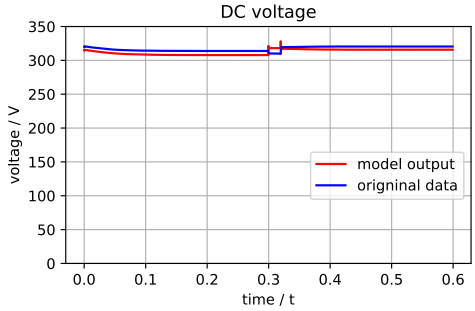


(a) Comparison of original data and model output

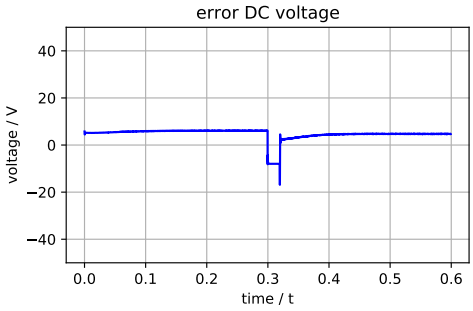


(b) Error of the modeled output

Figure 37: DC current of the composition of electric machine and inverter modeled via a Recurrent Neural Network



(a) Comparison of original data and model output



(b) Error of the modeled output

Figure 38: DC voltage of the composition of electric machine and inverter modeled via a Recurrent Neural Network

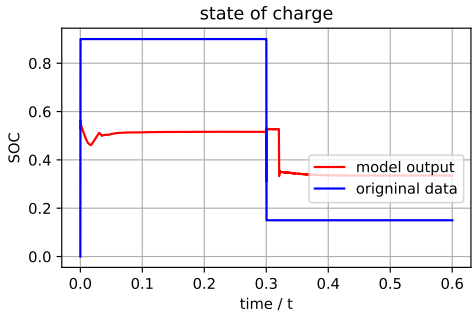
6.7. Results of the Models of the Composition of Electric Machine, Inverter and Battery

The battery‘s SOC as output of the models of the composition of electric machine, inverter and battery 4.3.5 is presented in this chapter.

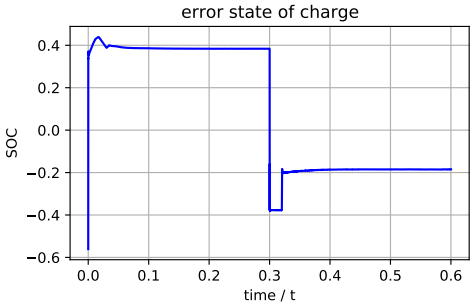
	FNN	**RNN**
	SOC	SOC
RMSE	0.31	0.35
RMSRE	48.15 %	54.62 %
MAE	0.29	0.35
MAPE	88.18 %	125 %

Table 7: Error of the electric machine + inverter + battery simulation models

The figures 39 and 40 as well as the table 7 show that the models, FNN as well as RNN, fail in simulating the correct output value for the composition of all components.
This inability of reproducing the output values of the models results from the fact that the values which the battery‘s SOC is sensitive to are missing in these models. The mechanical input values do not have the needed information for the battery‘s SOC, s. chapter 6.8.2.

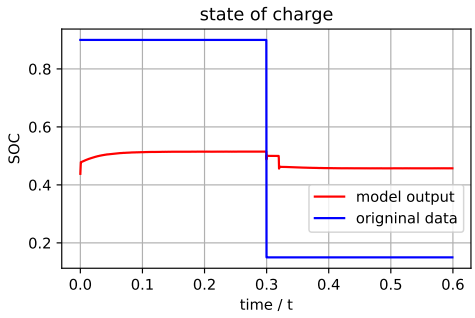


(a) Comparison of original data and model output

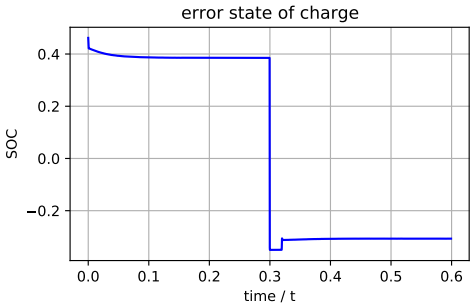


(b) Error of the modeled output

Figure 39: State of Charge value of the composition of electric machine, inverter and battery modeled via a Feedforward Neural Network



(a) Comparison of original data and model output



(b) Error of the modeled output

Figure 40: State of Charge value of the composition of electric machine, inverter and battery modeled via a Recurrent Neural Network

6.8. Discussion on the Results

6.8.1. Further Discussion on the Single Component Models

In two of three cases the RNN architecture performs better than the FNN architecture does. The reason for the better performance of the RNN architecture might come due to the input sequences. The RNN models do not only use input values at one point in time, but additionally the information of former points in time, which is according to [35] better suited for dynamic systems.

The better performance of the FNN for the electric machine simulation does not seem to be systematic, as the models of the other single components seem to work better using the RNN architecture. It is more likely that in this case the RNN overfits on the training data. Which means in this case that the 2 layers, s. table 2, worked perfectly for the training data but are not powerful enough to generalize. Countermeasures could be the further training and testing using a more powerful RNN setup.

The erroneous behavior of the FNN for the simulation of the inverter can possibly result from the network‘s inability to generalize between two operation modes, motor mode and generator mode. Although there is actually a negative DC current, which means the electric drive is in generator mode, the FNN simulates a positive current. As the FNN consists of 4 layers, s. table 2, it is unlikely that these wrong output values are a result of underfitting. It is more likely that the models have overfitted during training to motor operation modes. Hence, possible countermeasures can be regularization of the model and using training data with more regeneration runs.

6.8.2. Further Discussion on the Composition Models

The problem at hand for the composition models is the missing information in the input values needed for correct reproduction of the output values.

For the electric machine + inverter this means that according to physical laws the mechanical and electrical power must be the same, when neglecting the power losses. But the relation of DC current to DC voltage cannot properly be mapped from the mechnical input values only. The DC values and also their exact relation are not sensitive to the mechanical input values. Due to this lacking information the DC values cannot be properly reproduced with the used input values.

The same is true for the composition of all three components. As described in chapter 2.1 the SOC has influence on the battery‘s internal resistance R_i as well as on the open circuit voltage U_OC. These two values have an impact on the relation of DC voltage and DC current. This relation of the DC values in turn allows the reproduction of the battery‘s SOC. But this information gets lost when using the mechanical values as input.

As stated in [12], the performance of a Machine Learning system relies stronger on the provided input data than on the Machine Learning algorithm itself. Accordingly there is no possibility to optimize the Neural Networks, FNN as well as RNN, to simulate the battery‘s SOC directly out of the rotation speed n and electrical torque M_el without finding additional input data the battery‘s SOC is sensitive to. The same is true for the correct DC values and their relation for the composition of electric machine and inverter.

7. Usability and Application of the Machine Learning Based Simulation Models

As stated in chapter 3.1 the Machine Learning based simulation models belong to the group of black box models. This means only the behavior at the system boundary is known, but not the system‘s internal states.
As a result the ML based models can in general be used in applications where the system‘s behavior at the system boundary is of interest. However the implementation requires the existence of a real physical system. The physical system is needed to conduct several experiment runs at different conditions. This data can then be used for the training and evaluation of the model. This approach has been presented in the previous chapters.
After the models are finally ready to be used they can be deployed in different applications with different modeling goals.
One possibility is the usage for applications like software in the loop (SIL), processor in the loop (PIL) and hardware in the loop (HIL) systems. These systems enable faster and less time consuming software and system qualification tests. The reason for their advantages is the easier automation of tests, the possibility to run tests at the system‘s limit without the risk of damaging the system and the faster processing of the test runs compared to the usage of a physical system.
These models can also be used not only on the right side of the V-model but also during earlier stages of product development. During system design they can be the basis for system design decisions, provided real physical systems are available to produce training and test data. These models can be used for the simulation of the vehicle‘s efficiency and range under different conditions without having a fully build up vehicle at hand. These results can provide information regarding which electric machine, which inverter, which battery etc. fits the vehicle‘s range requirements the best. Furthermore such models can be used for defining the operation strategy in hybrid electric vehicles regarding minimizing the vehicle‘s overall energy demand.
The introduced models can also be applied in real-time systems themselves, not only for the simulation of them. Applications where the Neural Networks are embedded in the system can be model predictive control of the components, battery SOC estimation for the usage in battery management systems, DC estimation of an electric drive when no DC sensor is available, etc.
Neural Networks consist of many neurons, which are simple mathematical processing units [27]. Due to the simple mathematical processing, the runtime of the models can be lower than the runtime of difference equation based models. This is an advantage for the usage in systems with strong time constraints.

8. Summary and Outlook

In this thesis the three following E/E components have been modeled with use of two different Neural Network architectures:

- Surface mounted permanent magnet synchronous machine
- Two-level three-phase inverter
- Lithium-ion battery

For the single components simulation models with a certain degree of accuracy could be realized via Recurrent and Feedforward Neural Networks. The RMSRE values of these models are between 1.9 % up to 5.23 % for the RNN models and between 3.6 % up to 34 % for the FNN models. The overall MAPE value of the RNN is between 1.42 % and 4.03 %. The FNN‘s MAPE value lies between $5.38 \cdot 10^{-3}$% and 5.7 % . For those component models there is potential of improving the Neural Networks‘ performance by conducting further model optimization. This is especially true for the RNN models as they have performed better in most cases, except when modeling the electric machine.

The modeling of composition of components via Neural Networks has also been investigated. These simulation models have performed worse than the single component models. The composition models of all three aforementioned components have even completely failed. For the composition models a model optimization or the investigation of further different ML algorithms is not advisable. Only expanding the input values with values to which the requested output values are sensitive can optimize the models‘ performance.

The usage of the ML based models also depends on the accuracy. HIL, SIL and PIL for instance need precise black box models. If the precision can be improved then the usage can be expanded to application that have higher requirements regarding the models‘ performance. Nevertheless the simple processing units are a benfit of the Neural Networks.

A possible course of action for improving the existent Neural Networks is:

- Generate more training and test data
- Take more features into account, e.g. different temperatures
- Vary further hyperparameters, like batch size for the backpropagation, loss function or max number of epochs during training

These actions can help to further decrease the error values of the Neural Network models. This will increase the models‘ precision and in turn increase their acceptance for further simulations using Machine Learning based models.

References

[1] Awad M. ; Khanna R. *Efficient Learning Machines: Theories, Concepts, and Applications for Engineers and System Designers*. Apress Media, 2015. ISBN: 978-1-4302-5989-3.

[2] Basler, A. "Eine modulare Funktionsarchitektur zur Umsetzung einer gesamtheitlichen Betriebsstrategie für Elektrofahrzeuge". PhD thesis. Karlsruher Instituts für Technologie (KIT), 2015.

[3] Binder, A. *Elektrische Maschinen und Antriebe*. Berlin; Heidelberg: Springer, 2012. ISBN: 978-3-540-71849-9.

[4] Bockrath S. et al. "State of Charge Estimation using Recurrent Neural Networks with Long Short-Term Memory for Lithium-Ion Batteries". In: *45th Annual Conference of the IEEE Industrial Electronics Society*. 2019, pp. 2507–2511.

[5] Bohn, C. ; Unbehauen H. *Identifikation dynamischer Systeme*. Wiesbaden: Springer Vieweg, 2016. ISBN: 978-3-8348-2197-3.

[6] Bungartz, H-J. et al. *Modellbildung und Simulation*. 2. Auflage. Berlin Heidelberg: Springer, 2013. ISBN: 978-3-642-37656-6.

[7] Chai T. ; Draxler R. R. "Root mean square error (RMSE) or mean absolute error (MAE)? – Arguments against avoiding RMSE in the literature". In: *Geoscientific Model Development Discussions*. 2014, pp. 1247–1250.

[8] Chemali E. et al. "Long Short-Term Memory-Networks for Accurate State of Charge Estimation of Li-ion Batteries". In: *IEEE Transactions on Industrial Electronics*. 2017, pp. 6730–6739.

[9] Clevert, D. ; Unterthiner, D. ; Hochreiter, S. "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)". In: *4th International Conference on Learning Representations, ICLR 2016, Conference Proceedings*. 2016.

[10] Géron, A. *Praxiseinstieg Machine Learning*. 2. Auflage. Heidelberg: O'Reilly, 2020. ISBN: 978-3-96009-124-0.

[11] Glorot, X., Bengio, Y. "Understanding the Difficulty of Training Deep Forward Neural Networks". In: *Proceedings of the 13th International Conferenceon Artificial Intelligence and Statistics*. 2010, pp. 249–256.

[12] Halevy, A. ; Norvig, P. ; Pereira, F. "The Unreasonable Effectiveness of Data". In: *IEEE Intelligent Systems*. 2009, pp. 8–12.

[13] Haykin, S. *Neural Networks and Learning Machines*. 3rd edition. New Jersey: Pearson Education, 2009. ISBN: 978-0-13-147139-9.

[14] Heaton, J. *The Number of Hidden Layers*. June 2017. URL: `https://www.heatonresearch.com/2017/06/01/hidden-layers.html`.

[15] Holbach, S. "Modellbasierte Softwareentwicklung zur sensorlosen feldorientierten Regelung einer permanentmagneterregten Synchronmaschine für den Einsatz in elektrisch unterstützten Aufladesystemen". Master's Thesis. Hochschule Kaiserslautern, 2016.

[16] Hornik K. et al. "Multilayer Feedforward Networks are Universal Approximators". In: *Neural Networks, Vol. 2*. 1989, pp. 359–366.

[17] Jenni, F. ; Wuest, D. *Steuerverfahren für selbstgeführte Stromrichter*. Zürich; Stuttgart: vdf Hochschulverlag AG der ETH Zürich, Teubner, 1995. ISBN: 3-519-06176-7.

[18] Joshi A. *Machine Learning and Artificial Intelligence*. Springer, 2020. ISBN: 978-3-030-26621-9.

[19] Kingma, D., Ba, J. "ADAM: A Method for Stochastic Optimization". In: *International Conference on Learning Representations*. 2015.

[20] Klambauer, G., Unterthiner, T., Mayr, A., Hochreiter, S. "Self-Normalizing Neural Networks". In: *Advances in Neural Information Processing Systems (NIPS)*. 2017.

[21] Li, X. ; Yu, W. "Dynamic system identification via recurrent multilayer perceptrons". In: *Information Sciences 147*. 2002, pp. 45–63.

[22] Lo J. T. "Dynamical system identification by recurrent multilayer perceptron". In: *Proceedings of the 1993 World Congress on Neural Networks*. 1993.

[23] Lo J. T. ; Bassu D. "Mathematical Justification of Recurrent Neural Networks with Long- and Short-Term Memories". In: *International Joint Conference on Neural Networks*. 1999, pp. 364–369.

[24] Lu S. ; Basar T. "Robust Nonlinear System Identification Using Neural-Network Models". In: *IEEE Transactions on Neural Networks Vol. 9*. 1998, pp. 407–429.

[25] Murray-Smith, R. et al. "Neural Networks for Modeling and control of a non-linear dynamic system". In: *IEEE International Symposium on Intelligent Control*. 1992, pp. 404–409.

[26] Norgaard M. et al. *Neural Networks for Modelling and Control of Dynamic Systems*. London: Springer, 2000. ISBN: 978-1-85233-227-3.

[27] Schmidhuber, J. "Deep learning in neural networks: An overview". In: *Neural Networks* (2015), pp. 85–117.

[28] Schmitt, T. ; Andres, M. *Methoden zur Modellbildung und Simulation mechatronischer Systeme*. Wiesbaden: Springer Vieweg, 2019. ISBN: 978-3-658-25089-8.

[29] Schröder, D. ; Böcker J. *Elektrische Antriebe - Regelung von Antriebssystemen*. 5. Auflage. Berlin: Springer Vieweg, 2021. ISBN: 978-3-662-62699-3.

[30] Smith, L. N. *A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay*. 2018.

[31] Suchaneck, A. "Energiemanagement-Strategien für batterieelektrische Fahrzeuge". PhD thesis. Karlsruher Instituts für Technologie (KIT), 2018.

[32] Sunden, B. *Hydrogen, Batteries and Fuel Cells*. Academic Press, 2019. ISBN: 978-0-12-816950-6.

[33] Tschöke, H. et al. *Elektrifizierung des Antriebsstrangs*. Berlin: Springer Vieweg, 2019. ISBN: 3-519-06176-7.

[34] Wang Y. "A New Concept using LSTM Neural Networks for Dynamic System Identification". In: *American Control Conference*. 2017, pp. 5324–5329.

[35] Wang, C-H. et al. "A Dynamic Neural Network Model for Nonlinear System Identification". In: *IEE Information Reuse and Integration for Data Science*. 2009, pp. 440–441.

[36] Wolpert D. H. "The Lack of A Priori Distinctions Between Learning Algorithms". In: *Neural Computation 8*. 1996, pp. 1341–1390.

[37] Wolpert D. H. ; Macready W. G. "No Free Lunch Theorems for Optimization". In: *IEEE Transactions on Evolutionary Computation Vol.1*. 1997, pp. 67–82.

[38] Zhang W. "On Training Optimization of the Generalized ADLINE Neural Network for Time Varying System Identification". In: *Chinese Control and Decision Conference*. 2009, pp. 775–780.

Appendix

A. Clarke Transformation

The clarke transformation transforms the physical value x in a 3 phase coordinate frame into a two phase coordinate frame by splitting the resulting complex vector in a real and a imaginary part. The resulting values are still AC values. As the clarke transformation can be used for any physical value we take the symbol x for a value to be transformed. The input are the 3 time and locus dependent values:

$$x_{\mathrm{U}} = \hat{x} \cdot \cos(\omega t) \cdot \tag{A.28}$$

$$x_{\mathrm{V}} = \hat{x} \cdot \cos(\omega t - \frac{2}{3}\pi) \cdot \cos(\frac{2}{3}\pi) \tag{A.29}$$

$$x_{\mathrm{W}} = \hat{x} \cdot \cos(\omega t - \frac{4}{3}\pi) \cdot \cos(\frac{4}{3}\pi) \tag{A.30}$$

Out of the three time and locus dependent sinusoidal values in a three phase AC machine a single complex space vector can be described.

$$\underline{\vec{x}}^{\,\mathrm{S}} = \frac{2}{3} \cdot (x_{\mathrm{U}}(t) + x_{\mathrm{V}}(t) \cdot e^{\mathrm{j} \cdot \frac{2\pi}{3}} + x_{\mathrm{W}}(t) \cdot e^{\mathrm{j} \cdot \frac{4\pi}{3}}) \tag{A.31}$$

with:

$$e^{\mathrm{j} \cdot \frac{2\pi}{3}} = -\frac{1}{2} + \mathrm{j} \cdot \frac{\sqrt{3}}{2} \tag{A.32}$$

$$e^{\mathrm{j} \cdot \frac{4\pi}{3}} = -\frac{1}{2} - \mathrm{j} \cdot \frac{\sqrt{3}}{2} \tag{A.33}$$

The real part of this complex vector can be expressed as:

$$x_{\alpha} = \Re\{\underline{\vec{x}}^{\,\mathrm{S}}\} \tag{A.34}$$

$$x_{\alpha} = \frac{2}{3} \cdot (x_{\mathrm{U}}(t) - \frac{1}{2} \cdot x_{\mathrm{V}}(t) - \frac{1}{2} \cdot x_{\mathrm{W}}(t)) \tag{A.35}$$

The imaginary part of this complex vector can be expressed as:

$$x_{\beta} = \Im\{\underline{\vec{x}}^{\,\mathrm{S}}\} \tag{A.36}$$

$$x_{\beta} = \frac{2}{3} \cdot (x_{\mathrm{U}}(t) + \frac{\sqrt{3}}{2} \cdot x_{\mathrm{V}}(t) - \frac{\sqrt{3}}{2} \cdot x_{\mathrm{W}}(t)) \tag{A.37}$$

B. Park Transformation

The park transformation takes the output of the clarke transformation and transforms it into a rotor oriented frame. Therefore the angle of the rotor must be known. The input vector is turned according to the rotor angle ϑ, s. equation B.1. As the rotor oriented frame turns togehther with the rotor itself, the output values are DC values.
The real part of this vector is the so called d-value and the imaginary part is the so-called q-value.

$$\underline{\vec{x}}^{\,\mathrm{R}} = \underline{\vec{x}}^{\,\mathrm{S}} \cdot e^{-\mathrm{j}\vartheta} \tag{B.1}$$

Using the trigonometric representation of the vector turning $e^{-\mathrm{j}\vartheta}$ and also using the α and β values, the rotor oriented vector can be expressed as follows

$$\underline{\vec{x}}^{\,\mathrm{R}} = (x_\alpha + \mathrm{j} \cdot x_\beta) \cdot (\cos(\vartheta) - \mathrm{j} \cdot \sin(\vartheta)) \tag{B.2}$$

$$\underline{\vec{x}}^{\,\mathrm{R}} = x_\alpha \cdot \cos(\vartheta) - \mathrm{j} \cdot x_\alpha \cdot \sin(\vartheta) + \mathrm{j} \cdot x_\beta \cdot \cos(\vartheta) + x_\beta \cdot \sin(\vartheta) \tag{B.3}$$

$$\underline{\vec{x}}^{\,\mathrm{R}} = x_\alpha \cdot \cos(\vartheta) + x_\beta \cdot \sin(\vartheta) + \mathrm{j} \cdot (-x_\alpha \cdot \sin(\vartheta) + x_\beta \cdot \cos(\vartheta)) \tag{B.4}$$

So the real and imaginary part of the rotor oriented vector look like this:

$$x_\mathrm{d} = x_\alpha \cdot \cos(\vartheta) + x_\beta \cdot \sin(\vartheta) \tag{B.5}$$

$$x_\mathrm{q} = -x_\alpha \cdot \sin(\vartheta) + x_\beta \cdot \cos(\vartheta) \tag{B.6}$$

C. Parameters of the used Electric Machine

connection	wye	
pol pairs p	4	
nominal voltage U_N	225	$\mathrm{V_{eff}}$
nominal current I_N	175	$\mathrm{A_{eff}}$
phase resistance R_s	4.25	mΩ
phase inductance L_s	90	µH
flux linkage Ψ_r	29	mVs

Table C.1: Parameters electric machine

D. Parameters of the used Battery

nominalc cell voltage U_N	3.3	V
max cell voltage U_max	3.6	V
min cell voltage U_max	2.5	V
cell capacity C	20	Ah
number of serial cells	100	
number of parallel cells	4	

Table D.1: Parameters battery

E. Results of the Hyperparameter Optimization

E.1. Hyperparameters for the Electric Machine Models

units	layers	learning rate	optimizer	activator	initializer	seq. length	Loss
10	2	0.001	2	tanh	Glorot	10	0.0007068
10	4	0.001	2	tanh	Glorot	20	0.0010001
10	4	0.0001	2	tanh	Glorot	20	0.0010825
30	2	0.001	2	tanh	Glorot	10	0.0011857
30	3	0.0001	2	tanh	Glorot	20	0.0012350
30	2	0.001	2	tanh	Glorot	5	0.0012386
20	2	0.0001	2	tanh	Glorot	5	0.0013692
20	4	0.0001	2	tanh	Glorot	20	0.0013716
10	3	0.0001	2	tanh	Glorot	20	0.0014877
30	2	0.0001	2	tanh	Glorot	20	0.0014961
30	4	0.001	2	tanh	Glorot	5	0.0015173
30	2	0.001	2	tanh	Glorot	20	0.0015212
30	2	0.0001	2	tanh	Glorot	5	0.0015323
20	3	0.001	2	tanh	Glorot	10	0.0015574
30	3	0.0001	2	tanh	Glorot	5	0.0015912
10	3	0.001	2	tanh	Glorot	5	0.0016144
20	2	0.001	2	tanh	Glorot	20	0.0016588
30	3	0.0001	2	tanh	Glorot	10	0.0016611
30	4	0.0001	2	tanh	Glorot	20	0.0016971

Table E.1: Hyperparameters for modeling an electric machine using Recurrent Neural Network

units	layers	learning rate	optimizer	activator	initializer	seq. length	Loss
30	2	0.001	2	elu	lecun normal	–	0.00050961
20	3	0.001	2	elu	he normal	–	0.00054814
20	4	0.001	2	elu	he normal	–	0.00068441
10	3	0.001	2	elu	lecun normal	–	0.00075791
20	4	0.001	2	elu	lecun normal	–	0.00099378
10	4	0.001	2	elu	lecun normal	–	0.0010686
10	2	0.001	2	elu	lecun normal	–	0.0010872
20	2	0.001	2	elu	lecun normal	–	0.0011359
10	4	0.001	2	elu	he normal	–	0.0013704
30	2	0.0001	2	selu	he normal	–	0.0013977
30	4	0.001	2	elu	lecun normal	–	0.0014227
20	3	0.001	2	elu	lecun normal	–	0.0014518
10	2	0.001	2	elu	he normal	–	0.0015705
20	2	0.0001	2	selu	he normal	–	0.0015856
10	3	0.001	2	elu	he normal	–	0.0019148
30	3	0.0001	2	elu	lecun normal	–	0.0019613
20	2	0.001	2	elu	he normal	–	0.0021296
10	4	0.0001	2	elu	he normal	–	0.0022703
30	2	0.0001	2	elu	lecun normal	–	0.0023016
30	2	0.0001	2	selu	lecun normal	–	0.0024474

Table E.2: Hyperparameters for modeling an electric machine using Feedforward Neural Network

E.2. Hyperparameters for the Inverter Models

units	layers	learning rate	optimizer	activator	initializer	seq. length	Loss
30	3	0.0001	2	tanh	Glorot	20	0.00008079
30	4	0.001	2	tanh	Glorot	20	0.00012876
30	2	0.0001	2	tanh	Glorot	20	0.00013545
30	3	0.001	2	tanh	Glorot	10	0.00013833
10	4	0.0001	2	tanh	Glorot	20	0.00013964
30	4	0.0001	2	tanh	Glorot	5	0.00015935
10	4	0.001	2	tanh	Glorot	20	0.00017523
20	3	0.0001	2	tanh	Glorot	20	0.00019896
20	3	0.0001	2	tanh	Glorot	10	0.00021894
20	4	0.0001	2	tanh	Glorot	5	0.00024683
20	2	0.001	2	tanh	Glorot	10	0.00035385
10	3	0.001	2	tanh	Glorot	10	0.00035974
20	4	0.0001	2	tanh	Glorot	10	0.00038902
30	2	0.001	2	tanh	Glorot	20	0.00050785
30	3	0.0001	2	tanh	Glorot	10	0.00053150
10	2	0.001	2	tanh	Glorot	10	0.00053692
20	3	0.001	2	tanh	Glorot	5	0.00055938
10	3	0.0001	2	tanh	Glorot	20	0.00059560
30	4	0.001	2	tanh	Glorot	10	0.00066952
10	2	0.0001	2	tanh	Glorot	10	0.00067086

Table E.3: Hyperparameters for modeling an inverter using Recurrent Neural Network

units	layers	learning rate	optimizer	activator	initializer	seq. length	Loss
20	4	0.0001	2	selu	he normal	–	0.0024236
30	4	0.001	1	selu	he normal	–	0.0025651
10	4	0.001	2	elu	he normal	–	0.0030968
20	3	0.001	2	elu	he normal	–	0.0037526
10	4	0.0001	2	selu	he normal	–	0.0039646
30	4	0.001	2	elu	lecun normal	–	0.0040520
20	4	0.001	2	elu	lecun normal	–	0.0040752
20	3	0.001	2	elu	lecun normal	–	0.0041693
30	2	0.001	2	elu	lecun normal	–	0.0041857
30	4	0.001	2	elu	he normal	–	0.0042898
10	3	0.001	2	selu	he normal	–	0.0044649
20	4	0.0001	2	elu	he normal	–	0.0045019
10	4	0.0001	2	elu	lecun normal	–	0.0045470
20	4	0.001	2	elu	he normal	–	0.0047760
10	2	0.001	2	selu	lecun normal	–	0.0048637
30	2	0.0001	2	elu	he normal	–	0.0048899
30	4	0.0001	2	selu	he normal	–	0.0050421
20	2	0.001	2	elu	he normal	–	0.0051255
30	3	0.001	2	elu	he normal	–	0.0051669
30	3	0.001	2	selu	he normal	–	0.0053494

Table E.4: Hyperparameters for modeling an inverter using Feedforward Neural Network

E.3. Hyperparameters for the Battery Models

units	layers	learning rate	optimizer	activator	initializer	seq. length	Loss
30	2	0.001	2	tanh	Glorot	5	0.00009699
20	3	0.0001	2	tanh	Glorot	10	0.00011977
10	4	0.0001	2	tanh	Glorot	10	0.00012041
10	2	0.001	2	tanh	Glorot	10	0.00012490
30	4	0.0001	2	tanh	Glorot	10	0.00013877
20	4	0.0001	2	tanh	Glorot	20	0.00014530
20	4	0.0001	2	tanh	Glorot	10	0.00015569
20	3	0.0001	2	tanh	Glorot	20	0.00015687
30	2	0.0001	2	tanh	Glorot	10	0.00015809
30	3	0.0001	2	tanh	Glorot	20	0.00017452
30	2	0.0001	2	tanh	Glorot	20	0.00022281
10	4	0.0001	2	tanh	Glorot	20	0.00025367
10	3	0.0001	2	tanh	Glorot	20	0.00026478
10	3	0.001	2	tanh	Glorot	5	0.00028599
10	3	.0001	2	tanh	Glorot	5	0.00029598
20	4	0.001	2	tanh	Glorot	20	0.00030824
30	4	0.0001	2	tanh	Glorot	20	0.00033606
10	3	0.001	2	tanh	Glorot	20	0.00035483
10	2	0.0001	2	tanh	Glorot	20	0.00037559
30	3	0.0001	2	tanh	Glorot	5	0.00041196

Table E.5: Hyperparameters for modeling a battery using Recurrent Neural Network

units	layers	learning rate	optimizer	activator	initializer	seq. length	Loss
20	3	0.001	2	elu	lecun normal	–	0.0012772
20	4	0.001	2	elu	he normal	–	0.0017765
30	4	0.001	2	elu	lecun normal	–	0.0018561
20	4	0.001	2	elu	lecun normal	–	0.0030858
30	4	0.001	2	selu	lecun normal	–	0.0034686
20	3	0.001	2	selu	he normal	–	0.0035035
30	4	0.001	2	selu	he normal	–	0.0042040
30	3	0.001	2	elu	lecun normal	–	0.0046374
10	4	0.001	2	selu	he normal	–	0.0048506
20	4	0.001	2	selu	he normal	–	0.0050979
10	4	0.001	2	selu	lecun normal	–	0.0058285
30	3	0.001	2	selu	he normal	–	0.0061292
20	4	0.0001	2	selu	lecun normal	–	0.0077423
30	3	0.0001	2	selu	lecun normal	–	0.0078089
10	4	0.001	2	elu	lecun normal	–	0.0083906
20	3	0.001	2	selu	lecun normal	–	0.0094561
20	4	0.001	2	selu	lecun normal	–	0.010603
30	2	0.001	2	elu	he normal	–	0.011193
10	2	0.001	2	selu	he normal	–	0.011562
30	3	0.0001	2	selu	he normal	–	0.012371

Table E.6: Hyperparameters for modeling a battery using Feedforward Neural Network

E.4. Hyperparameters for the Models of the Composition of Electric Machine and Inverter

units	layers	learning rate	optimizer	activator	initializer	seq. length	Loss
10	3	0.0001	2	tanh	Glorot	5	0.00040417
10	3	0.0001	2	tanh	Glorot	20	0.00043714
30	3	0.0001	2	tanh	Glorot	10	0.00052782
10	3	0.001	2	tanh	Glorot	10	0.00054376
20	3	0.0001	2	tanh	Glorot	5	0.00058867
20	4	0.0001	2	tanh	Glorot	20	0.00060523
20	3	0.001	2	tanh	Glorot	20	0.00062242
20	3	0.0001	2	tanh	Glorot	10	0.00064148
10	4	0.0001	2	tanh	Glorot	10	0.00064971
20	4	0.001	2	tanh	Glorot	20	0.00067386
30	2	0.0001	2	tanh	Glorot	5	0.00069038
30	2	0.001	2	tanh	Glorot	5	0.00072235
10	2	0.0001	2	tanh	Glorot	10	0.00072308
30	2	0.0001	2	tanh	Glorot	10	0.00072651
10	2	0.0001	2	tanh	Glorot	20	0.00072774
30	2	0.001	2	tanh	Glorot	10	0.00075190
10	3	0.001	2	tanh	Glorot	20	0.00076219
20	3	0.001	2	tanh	Glorot	5	0.00078347
30	4	0.0001	2	tanh	Glorot	20	0.00081651
10	4	0.0001	2	tanh	Glorot	5	0.00082800

Table E.7: Hyperparameters for modeling the composition of electric machine and inverter using Recurrent Neural Network

units	layers	learning rate	optimizer	activator	initializer	seq. length	Loss
10	4	0.001	2	selu	lecun normal	–	0.00046011
10	2	0.001	2	elu	lecun normal	–	0.00053527
10	3	0.0001	2	elu	lecun normal	–	0.00062879
10	4	0.0001	2	selu	he normal	–	0.00073784
30	4	0.001	2	elu	lecun normal	–	0.00075276
30	3	0.001	2	elu	lecun normal	–	0.00078126
30	2	0.0001	2	elu	lecun normal	–	0.00078786
20	2	0.001	2	elu	lecun normal	–	0.00084607
30	2	0.0001	2	selu	lecun normal	–	0.00086813
10	4	0.0001	2	elu	he normal	–	0.00091375
10	2	0.001	2	elu	he normal	–	0.00091640
20	3	0.001	2	selu	lecun normal	–	0.0010131
30	2	0.001	2	elu	he normal	–	0.0010136
30	2	0.0001	2	elu	he normal	–	0.0010371
20	3	0.0001	2	elu	he normal	–	0.0010390
10	4	0.001	2	elu	lecun normal	–	0.0010470
20	2	0.0001	2	selu	he normal	–	0.0010569
20	2	0.001	2	elu	he normal	–	0.0010891
10	3	0.0001	2	selu	lecun normal	–	0.0010966
30	4	0.0001	2	elu	he normal	–	0.0011123

Table E.8: Hyperparameters for modeling the composition of electric machine and inverter using Feedforward Neural Network

E.5. Hyperparameters for the Models of the Composition of Electric Machine, Inverter and Battery

units	layers	learning rate	optimizer	activator	initializer	seq. length	Loss
10	3	0.0001	1	tanh	Glorot	5	0.097418
20	2	0.0001	1	tanh	Glorot	5	0.10134
10	4	0.0001	1	tanh	Glorot	10	0.10554
20	2	0.0001	1	tanh	Glorot	10	0.11406
30	2	0.0001	1	tanh	Glorot	5	0.12005
20	3	0.0001	1	tanh	Glorot	5	0.12545
30	3	0.0001	1	tanh	Glorot	5	0.12762
30	4	0.0001	1	tanh	Glorot	10	0.12944
30	4	0.0001	1	tanh	Glorot	20	0.13111
30	4	0.0001	1	tanh	Glorot	5	0.13243
20	4	0.0001	1	tanh	Glorot	20	0.13383
20	3	0.001	2	tanh	Glorot	10	0.13482
30	3	0.001	2	tanh	Glorot	10	0.13571
30	2	0.0001	1	tanh	Glorot	20	0.13619
20	4	0.001	2	tanh	Glorot	20	0.13637
30	2	0.001	2	tanh	Glorot	10	0.13701
20	2	0.001	2	tanh	Glorot	20	0.13727
10	2	0.001	2	tanh	Glorot	20	0.13745
20	2	0.0001	2	tanh	Glorot	20	0.13751
10	3	0.001	2	tanh	Glorot	10	0.13810

Table E.9: Hyperparameters for modeling the composition of electric machine, inverter and battery using Recurrent Neural Network

units	layers	learning rate	optimizer	activator	initializer	seq. length	Loss
30	4	0.0001	1	selu	he normal	–	0.075711
10	3	0.001	1	selu	he normal	–	0.079244
10	3	0.001	2	selu	he normal	–	0.090691
30	3	0.001	2	selu	he normal	–	0.092600
30	3	0.0001	2	selu	he normal	–	0.094740
10	4	0.001	1	selu	he normal	–	0.095412
30	4	0.0001	1	selu	lecun normal	–	0.096456
10	4	0.0001	1	selu	he normal	–	0.096735
10	4	0.0001	2	selu	lecun normal	–	0.097048
10	3	0.0001	2	elu	he normal	–	0.097065
20	4	0.001	2	selu	lecun normal	–	0.099736
30	4	0.0001	2	selu	he normal	–	0.10394
30	2	0.001	2	elu	he normal	–	0.10484
10	3	0.0001	2	selu	he normal	–	0.10616
20	4	0.001	1	selu	lecun normal	–	0.10623
30	3	0.001	2	elu	he normal	–	0.10649
20	2	0.0001	1	elu	he normal	–	0.10699
20	3	0.0001	2	selu	lecun normal	–	0.10724
30	3	0.001	2	selu	lecun normal	–	0.10871
30	2	0.001	2	selu	he normal	–	0.10945

Table E.10: Hyperparameters for modeling the composition of electric machine, inverter and battery using Feedforward Neural Network