A Method for Learning Automated Driving Systems and Data Consistency Analysis

Master's thesis in Complex Adaptive Systems

ERIK SETTERSTÅL

DEPARTMENT OF PHYSICS

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025
www.chalmers.se

Master's thesis 2025

A Method for Learning Automated Driving Systems and Data Consistency Analysis

ERIK SETTERSTÅL

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Physics
Chalmers University of Technology
Gothenburg, Sweden 2025

A Method for Learning Automated Driving Systems and Data Consistency Analysis
Erik Setterstål

Industrial supervisor:
PhD Giuseppe Giordano,
Zenseact

Academic supervisor and examiner:
Associate Senior Lecturer Mohsen Mirkhalaf,
Department of Physics, University of Gothenburg

Master's Thesis 2025
Department of Physics
Mechanics and Physics of Materials
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Acknowledgements, dedications, and similar personal statements in this thesis, reflect the author's own views.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2025

A Method for Learning Automated Driving Systems and Data Consistency Analysis

Erik Setterstål
Department of Physics
Chalmers University of Technology

Abstract

In this thesis a method for developing a deep learning network to predict signals in automated driving systems is presented. The goal is to incorporate the network in a tool aimed to find discrepancies in vehicle testing data. To this end, a neural network was constructed and trained on simulations of vehicle dynamics with the purpose of predicting the dynamics of a real test vehicle. A parallel goal was to compare the properties of the network to a software for simulating vehicle dynamics. The thesis was made in collaboration with Zenseact, an AI and software company developing the complete software stack for automated driving and advanced driver-assistance systems (ADS and ADAS). The network did predictions replicating a simulation software's Controllers, Vehicle model and the two combined. To evaluate which type of training data gave the best result, three datasets with different driving schemes were generated by the simulation software. On a test set constructed from 21 log files from real test drives, the network got the best results predicting the Controllers and Vehicle model combined. In this case, for only predicting a vehicle's lateral motion, a training set consisting of constant longitudinal velocity generated the best results. If both lateral and longitudinal quantities are desired, a dataset with sinusoidally changing acceleration performed best overall. The other purpose was to compare the time aspects of the network to the simulation software. This was done by evaluating the time for data generation, training and prediction on the "Constant" dataset used in this thesis. This dataset is comprised by 84,528 time steps of ADS signals. Data generation took 8.5 to 9 minutes over all tests. The training time was consistently 61 to 62 seconds in total and the prediction time on the 84,495 batches was between 6 to 7 seconds, or an average of 75.9 µs per batch. It was concluded that, apart for the intended goal of being a core in a data consistency tool, the main use case for the network could be in applications where fast data generation times are crucial, for example in near real time predictions during test drives.

Keywords: Automated driving systems, Bicycle model, Data consistency analysis, Long Short-Term Memory, Neural networks, Vehicle dynamics.

Acknowledgements

I would like to thank my industrial supervisor PhD Giuseppe Giordano for his support and presence during the entire thesis. Having desk beside you has been invaluable for guidance in the project. Thank you to all the other guys in Vehicle control for always being available when I get stuck and to all the other great people I have had the pleasure to meet during this spring at Zenseact. I also want to express gratitude towards my academic supervisor and examiner Professor Mohsen Mirkhalaf for his advices, always being positive and supportive. Lastly to my parents, a thank you is not enough. This journey would never have been possible without you.

Erik Setterstål, Gothenburg, May 2025

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

Adam	Adaptive Moment Estimation
ADS	Automated Driving Systems
API	Application Programming Interface
FF-layer	Feed-forward layer
FFNN	Feed-forward neural network
GPU	Graphical Processing Unit
HDF	Hierarchical Data Format
KPI	Key Performance Indicators
LSTM	Long Short-Term Memory
MAE	Mean Absolute Error
MSE	Mean Squared Error
NMAE	Normalized Mean Absolute Error
NMSE	Normalized Mean Squared Error
ReLU	Rectified Linear Unit
RMS	Root-Mean Square
RMSProp	Root-Mean Square Propagation
RNN	Recurrent neural network
SGD	Stochastic Gradient Descent

Nomenclature

Below is the nomenclature of indices, sets, parameters, and variables that have been used throughout this thesis.

Indices

i,j	Indices for input and output to neural network
t	Index for time step

Variables - Machine learning

$g(x)$	Activation function of a neuron
lr, lr_0	Learning rate and Initial learning rate
n	Number of datapoints in prediction
O, O_j	Output (j) of a neural network or neuron
w_i	Weight from input i to a neuron
$w_{i,j}$	Weight between input i and neuron j
$x, x_i, x_{i,j}$	Input to a neural network (or neuron i) (to neuron j)
y_i	Target i
$\hat{y}_i$	Prediction on y_i
β, β_j	Bias term in a neuron or neuron j

Variables - Vehicle dynamics

a_x	Longitudinal acceleration
a_y	Lateral acceleration
v_x	Longitudinal velocity
v_y	Lateral velocity

x	Longitudinal direction in ego frame
X	Longitudinal direction in global frame
y	Lateral direction in ego frame
Y	Lateral direction in global frame
ψ	Yaw angle or front wheel angle
$\dot{\psi}$	Yaw rate

Parameters

ε	Small positive number, $1 \gg \varepsilon > 0$
$\bar{y}$	The mean of y

Contents

List of Figures

Note: If this thesis is printed in greyscale, the legends in all figures are in the same order as the graphs on the right in each figure. For the prediction figures in chapter 4, the otherwise red prediction graphs will be grey and the target graphs will be close to black.

List of Tables

1

Introduction

Zenseact is one of the companies within the fast moving world of autonomous vehicles and ADS, or *automated driving systems*. One important part of evolving the current capabilities in ADS is advanced simulation of vehicle dynamics [1]. To this end, this thesis will focus on developing a deep learning network which will be the core of a tool to help the evaluation of new software for automated driving. By comparing real vehicle test data with modelled behavior, inconsistencies and faults in the tested model can be identified. Finding these errors early in the development process is crucial to improve system performance and make the workflow more effective. This comparison is currently mostly manual, where the developers are doing a combination of evaluating KPIs, or *key performance indicators*, and generating graphs where important aspects of the data can be examined.

Previous classical strategies for fault analysis have been investigated, for example Dulmage-Mendelsohn decompositions [2] used in a toolbox [3] to calculate fault isolability matrices. A fault isolability matrix is a table showing the relations between faults in a system [4]. With this method the equations of a system can be organized depending on if they are under-, exactly-, or overdetermined [3]. This toolbox has been used in [5] and to find *Minimal Structurally Overdetermined* sets (MSO), which only contain the overdetermined part and can be used to find *residual generators* [6]. A residual generator is a function whose output only should change if a fault arises [7]. MSO calculations were put in practice by [8] and [9], where the latter utilized *Fault Driven Minimal Structurally Overdetermined* sets (FMSO). The main advantage with FMSO over MSO is that FMSO does not contain any empty faults, which then reduces the number of sets to analyze and in turn makes it more effective [10].

Methods leaning more towards data analysis and machine learning include use Gaussian mixture models and support vector machines, both discussed for driving pattern identification in [11], where neural networks are given as another option. Machine learning is also brought up for fault detection and analysis of the isolability of a system by open-set classification. This approach, utilizing Kullback-Leibler divergence as a measure for dissimilarity, has been explored in [12]. The direction this thesis will investigate is to train a neural network to predict signals in automated driving systems. These predictions can then be compared to real test data to calculate KPIs and evaluate performances in development faster than running and examining the output of a simulation software.

1.1 Research questions

The main goal with this thesis is to answer the question ”What are the advantages and disadvantages of instead of using a vehicle dynamics simulation software based on classical methods, let a deep neural network predict the output of the important parts of the software?”. These parts are 1) Controllers, 2) Vehicle dynamics model and 3) the entire Closed loop. In practice 1) implies the task of predicting one torque and one steering signal from a simulated road and desired acceleration, 2) use the output of the Controllers to predict the physical dynamic properties of the vehicle and lastly 3) both of them combined. Achieving this, especially 1) or 3), would then make the network a central part in new software applications aimed to detect discrepancies in vehicle dynamics between real test data and simulation.

One aspect of the question above to be investigated more thoroughly is for which type of training data the neural network gives the best fitted predictions on a fixed test set. Three different types are examined, all collections of log files generated by the simulation software. The three training sets will contain different acceleration and road profiles in order to examine which combinations allow for best performance on a test set with dynamics from a real vehicle. By training on simulation data, the network will learn the ideal, noise free behavior of the Closed loop. Its predictions can then be compared to the real test data, which contain the faults to be identified.

1.2 Purpose of the study

There are two parallel purposes of this study. Firstly, to conduct research to answer the associated research questions in section 1.1. Secondly, to develop a software for Zenseact which, by near real time predictions of signals in automated driving systems, can open opportunities for new tools. The two examples initially planned for are to ease and hasten the current laborious task of finding discrepancies in a large set of log files. The second is when doing real test drives. Meanwhile the vehicle is controlled by new software, the network could run in parallel and alert when it predicts that the simulation software would have behaved differently than the vehicle. This in turn could make debugging and development of software more efficient, enabling the company to reach their goal of zero traffic accidents faster.

1.3 Limitations

To predict the ADS signals, the neural network was trained and validated on datasets generated by the simulation software during the thesis. This was done for two reasons. Firstly, the real test data should be compared with noise- and fault-free data to find the discrepancies between real world driving and simulation. Secondly, to both ensure full replicability and complete knowledge of the used inputs. These datasets had the following limitations. Parametrically, the longitudinal acceleration was limited to constant or sinusoidal and the shape of the road to sinusoidal or as a chirp wave. It was also decided that the size of the datasets and network would

not be too large to prevent training locally.

Another limitation is the aspect of the thesis connected to data correctness. This has two sides. The modelling of vehicle dynamics in the simulation software gives a good approximation of how the real vehicle will behave from the inputs. However, simulating vehicle dynamics is a hard problem (otherwise this thesis would never have been written) and there are always a risk of errors. The same is valid for neural networks. Training is stochastic and, especially in the case of the type of black box model used in this thesis, the reasons for the outputs are unknown.

1.4 Ethical aspects

The Thesis' main impact will be enabling the developers at Zenseact to automatically analyze the log files, which will save a lot of time and make the development process more efficient. This will faster advance the company towards its goal of zero traffic collisions.

Recent studies [13] suggest that autonomous vehicles are involved in less accidents than human driven ones. Because of the large amount of vehicles that will run Zenseact's software, less, or hopefully none of them, being in a collision could significantly reduce the total number of traffic collisions [14]. This of course has many positive effects, with the most direct one being that it will lead to a safer and more efficient society. Even that there are beliefs of autonomous driving not being the solution to reach "Vision Zero" [15], it is still thought to be one way to reduce the numbers of accidents [14]. Less collisions will, apart from less injuries, also imply less traffic jams and less damage on roads and vehicles. These are all positive effects that will consequently reduce emissions and environmental impacts, both in terms of repairing damaged infrastructure and less need for new cars.

Except a safer society, the main ethical effect will be for the developers, not needing to do the laborious task of analyzing the data manually. Although there still is a need to use and interpret the tool, the aim is to give the users an understandable overview and interpretation of the data.

There are however also possible negative aspects. All entities that are connected and have computers, including vehicles, are potential targets of cyber attacks. Improved software for autonomous driving will both increase the number of self driving vehicles and imply that the software have access to controlling more vital parts of the car. These are two aspects raising the incentive to develop new attacks and therefore constitute a larger threat [16]. With more computers and more data also follows more data management. A recent study [17] shows that the increased data management in autonomous vehicles could imply considerable energy consumption.

2

Theory

In order to describe and motivate the used methods and the obtained results, some background information will first be presented. Starting with the underlying concepts, used tools and common techniques in machine learning, section 2.1 gives the reader all knowledge needed to follow the choices made in 3, the method chapter. Afterwards section 2.2 provides information about the chosen programming tools, before lastly explaining all knowledge needed for the parts of the thesis connected to simulation of vehicle dynamics in section 2.3. Common concepts and the structure of the simulation software will be described. This will be important to understand what functions the neural network is supposed to replicate.

2.1 Neural networks

This section will summarize the basics, different cell types and techniques used in this thesis within the broad field of neural networks and machine learning. Viewing definitions of these topics in other publications, it was noted that many different ones exist. Shaik et al. [18] explains it as a *group of algorithms that simulates the way the human brain works* and Vyas et al. [19] gave the summary *a neural population constitutes a dynamical system that, through its temporal evolution, performs a computation*. Finally, closest to the author's view of the topic and first seen by the author in [20], is figure 2.1. It shows that neural networks in their core just are equations and linear algebra on large scales fitted to even larger sets of data. To give a more direct and specific definition, the following sections will explain how the used types of neural network layers work.

Figure 2.1: From [21], first seen in [20] and used under a Creative Commons Attribution-Non Commercial 2.5 License. The figure shows a satirical point of view of neural networks and machine learning, with the point that despite how complex and black-box the solution might be, it is just a large system of equations accompanied by an even larger amount of data.

2.1.1 Feed-forward neural network

An artificial neuron, or McCulloch-Pitts neuron [22] is the foundation of artificial neural networks. To explain this building block, a schematic of it can be seen in figure 2.2 and the way its output is calculated will be made clear now. A single neuron, also called a *perceptron*, from n inputs $x_1, ..., x_n$ calculates the output O as

$$O = g\left(\sum_{i\in[1,n]} x_i w_i + \beta\right),$$

where g is the *activation function* and w_i are the weights [20]. β is a number called *bias* which, together with the weights, are the parameters that are changed when the network is adjusted to a certain task [20]. Combining more neurons in parallel creates a *layer* and different structures of combined neurons are called a *network* [20]. The output of neuron j in a layer is calculated as [20]

$$O_j = g\left(\sum_{i\in[1,n]} x_{i,j} w_{i,j} + \beta_j\right). \tag{2.1}$$

Note that $i \in [1, n]$, but does not necessarily have to include all values. A neuron or layer which inputs all outputs from the previous layer ($i = 1, ..., n$) is called a *dense* layer [23]. A *feed-forward neural network*, or FFNN, is a network characterized by only passing the data to later layers [24]. An example of a FFNN with two layers between the input and output (called *hidden layers*) can be seen in figure 2.3. A common use case for FFNN is classification tasks, where one input directly corresponds to one output and the prediction does not have a connection to the next, nor previous, input-output pair. As an example of this, the MNIST dataset [25] consists of images of handwritten digits. Here one image (input) corresponds to the number written on the image (output). A classification task for a network could be to predict the number from the image [26]. Which number is written in the image does not have any correlation to the next, or previous, image since all input-output pairs are isolated.

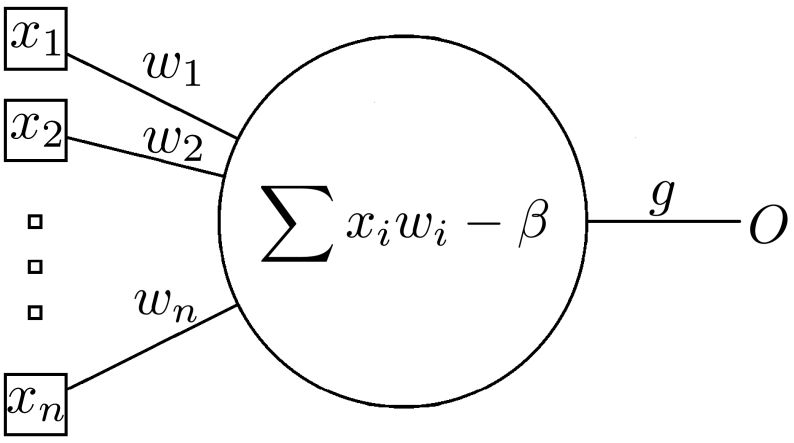

Figure 2.2: An artificial neuron with n inputs and one output. x_i are inputs, w_i are corresponding weights, β is the bias term, g is the activation function and O is the output. The activation function inputs the value of the expression in the neuron.

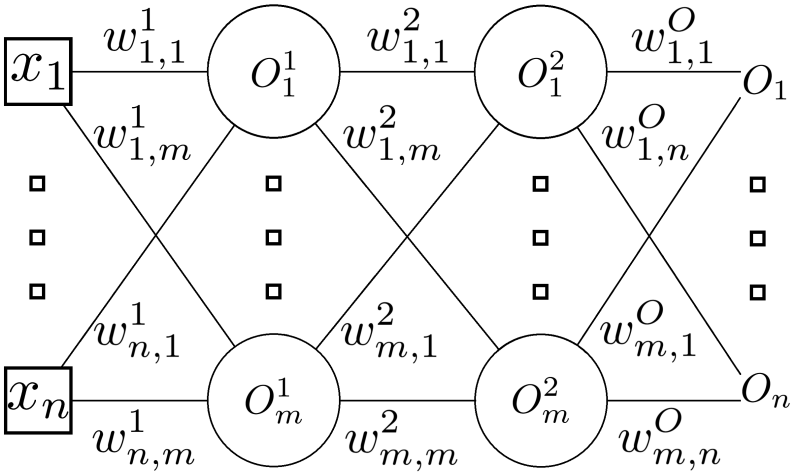

Figure 2.3: Two layers of m artificial neurons, as the one in figure 2.2. $x_1, ..., x_n$ are the inputs, $w^i_{j,k}$ are the weight in layer i between input i and neuron j. Both hidden layers have m neurons, all with individual bias terms β^i_j, where i is the layer and j is the neuron. In each calculation of a neuron's output, an activation function g is applied. Note that the activation functions not necessarily have to be the same for all neurons in a layer, or in all layers. There are n output neurons, $O_1, ..., O_n$. Their values are calculated in the same fashion as the neurons in the hidden layers, with a weighted sum, bias term and activation function.

Training a network can be seen as a form of function fitting. A training session is divided into *epochs* [20]. In each epoch the training set, or parts of it, is fed into the network and the outputs are calculated. If the output does not match the target associated with that input, the difference between the output and the target is used to adjust the weights and bias term. This is what is called *training* the network, since it each epoch should train the parameters to match the outputs to the targets.

The purpose of the activation function is to introduce non-linearity to the network, which is crucial for it to learn more intricate patterns [23]. It is chosen depending on the type of data that will be used in the network. Common choices, which also will be used in this thesis, are the hyperbolic tangent function [27],

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}},$$

the sigmoid function [28]

$$\sigma(x) = \frac{1}{1 + e^{-x}},$$

the rectified linear unit [27] and leaky rectified linear unit [28],

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

$$\text{Leaky ReLU}(x) = \begin{cases} \alpha x, & x < 0 \\ x, & x \geq 0. \end{cases}$$

An illustrative comparison of these can be seen in figure 2.4.

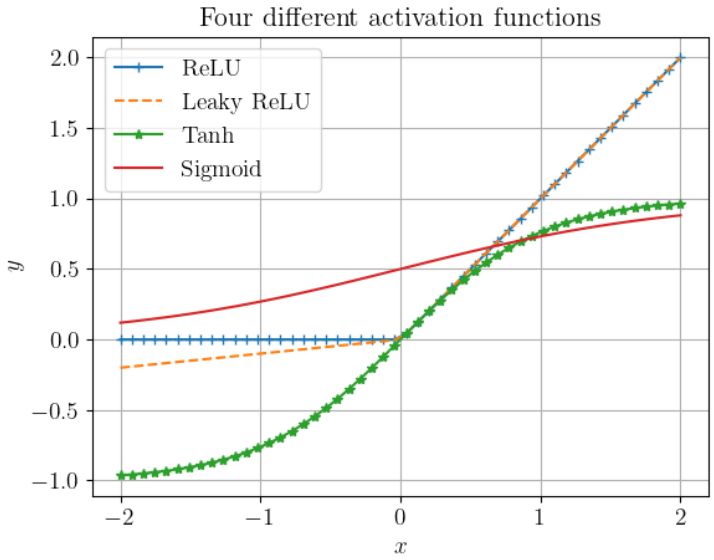

Figure 2.4: Difference between the four activation functions described above for input values $x \in [-2, 2]$. In this case $\alpha = 0.1$ for the leaky ReLU.

2.1.2 Recurrent neural network

As opposed to the FFNN, in a *recurrent neural network*, or RNN, the data is not only fed to latter layers, but also to previous ones, or back to the current one [20]. The most simple version of a single recurrent neuron is visualized in figure 2.5. For this kind of a single-input-single-output RNN, the equation for the output is [20]

$$O_t = g\left(x_t w_1 + O_{t-1} w_{fb} - \beta\right). \tag{2.2}$$

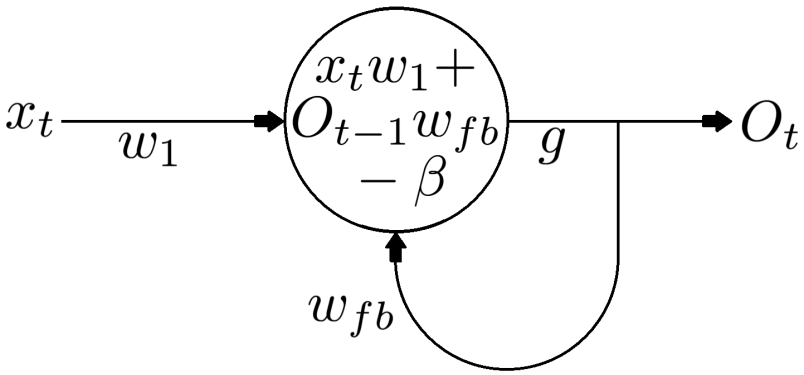

Figure 2.5: Structure of a simple recurrent neuron with one input, x_t, one output O_t and one feedback connection from the last output with weight w_{fb}.

One common use case where RNNs excel is time series data, where each datapoint corresponds to a time step and the relation between data points are important [29]. The RNN could then feed back its output from one time step and use that as an extra input for the next one, giving it temporal information valuable for these kinds of tasks [30]. An example is Kumari et al. [31] who leveraged the same types of RNNs

as in this report for weather forecasting. The RNNs use of the current state gives important information about the time evolution of the system. In a similar way to how the forecast is dependent on previous weather, a vehicle cannot suddenly have a heading, velocity or acceleration in whatever direction, it must evolve continuously from earlier states. There are many different variations of a RNN, one common that will be used in this report is *Long Short-Term Memory*, or LSTM for short.

One problem with the simple RNN is the *vanishing gradient problem*, which can be understood from equation 2.2. The gradient of the weights (the derivative of the weights during training) will, because of the chain rule, depend on all derivatives in later layers. When minimizing the loss function with gradient descent, the weights in the first layers will therefore hardly change [20]. The LSTM was created to be more resistant to this problem [20] by learning longer term properties of the current system dynamic [32]. A schematic of a LSTM cell can be seen in figure 2.6.

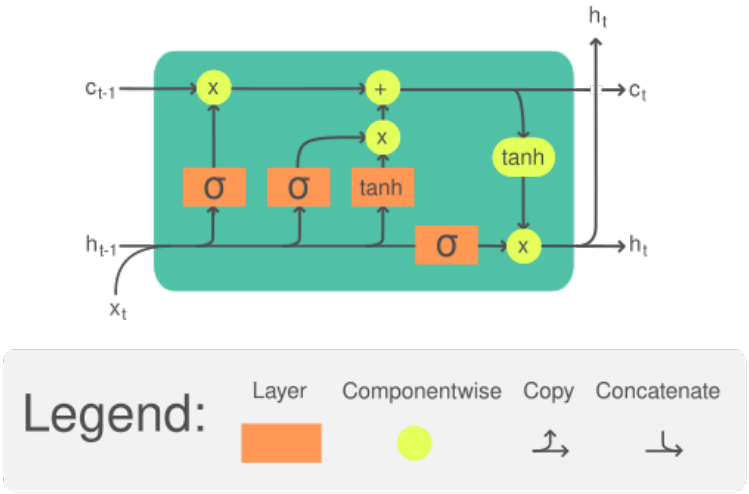

Figure 2.6: Structure of a LSTM cell. The c_t:s are the memory gates, h doubles as a hidden state and output vector of the cell, x is the input vector and σ is a sigmoid activation function. CC BY-SA 4.0, [33].

2.1.3 Over- and underfitting

Two common problems when training neural networks are over- and underfitting [34]. If a neural network is underfit, it has not learnt enough of the patterns in the training data to give good predictions, neither for training nor validation data [34]. A common cause of underfit predictions can be an incorrectly set up network, according to [35]. Overfitting is the opposite problem, where the network has started to learn the details in the training set rather than the trends shared with the validation set. This can be noticed as a low training loss but high validation loss [36]. Instead of an incorrectly set up network, overfitting can be caused by the network having too many parameters to tune in relation to the amount of data [36]. A visual representation of under-, good, and overfit predictions can be seen in figure 2.7. Two techniques to prevent overfitting will be described below.

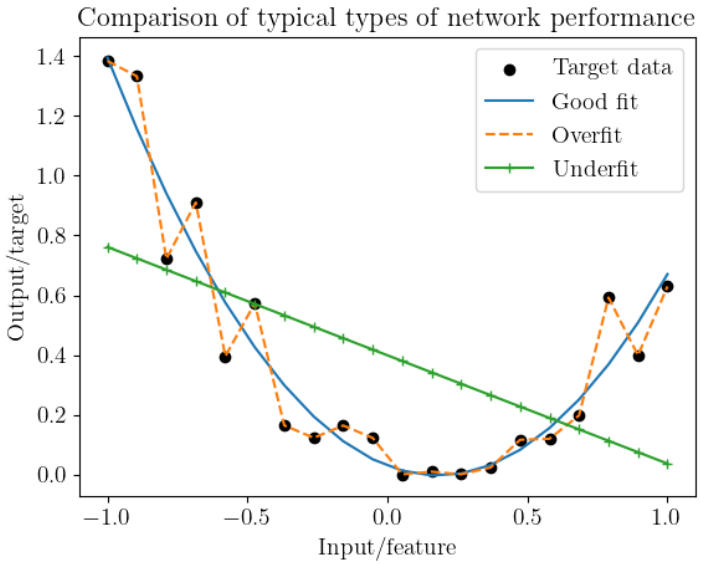

Figure 2.7: Illustrative difference between a network being under-, good or overfit to the data. The graphs show examples how the three cases could look on training data.

2.1.4 Drop out layer

As mentioned about equation 2.1 in section 2.1.1, not all neurons in a layer (or the network output) must have connections to all neurons in the previous layer. Removing connections is another strategy to avoid overfitting and common way to implement it is with a *drop out layer* [20]. Inserting a drop out layer will choose a set fraction of the neurons in the previous layer to not be used during training. These neurons are not included in either prediction or weight updates, but each epoch the neurons are randomly selected [20]. At the end of the training, all neurons are used [20]. Reducing the number of trainable parameters are also in line with the common problem with overfitting stated by Ali et al. in [35].

2.1.5 Early stopping

When to stop the training of a network is also a crucial problem to prevent overfitting [37]. To know when to stop training, the validation loss is tracked and the training is aborted when the validation loss increases. This indicates that the network has started to learn the specifics of the training set rather than the general trends shared by the training and validation set [20]. A common behavior of the training and validation loss can be seen in figure 2.8. Here the training loss keeps decreasing even when the validation loss starts to increase. This is when the model will overfit if the training is not aborted.

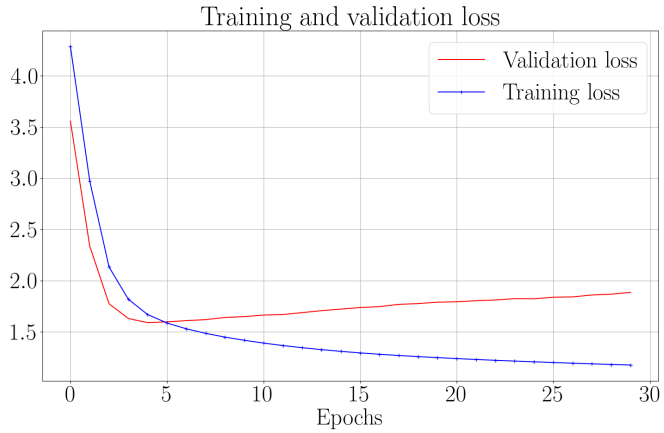

Figure 2.8: Graph of how the training set and validation set losses commonly behave during training. Overfitting will occur when the validation loss starts to increase at the same time as the training loss keeps decreasing.

2.2 Programming languages, frameworks and formats

This section gives a brief background for the most important systems used in the thesis connected to programming and development.

2.2.1 Python

Python is a high-level programming language [38] and will be the main language used in this thesis. Python has many advantages for these kinds of projects. Because of its relatively simple syntax and ability to import modules for specific use cases, it is faster to develop in than lower level languages, such as C++. For example, one way to decrease the training time for a neural network is to move the training to the GPU, or graphical processing unit [39]. This can be done with the module Tensorflow, which will be described more below.

2.2.2 Tensorflow

Compatible with Python is the machine learning ecosystem Tensorflow [40]. It uses the high-level Keras API, which is supposed to be simple, flexible and powerful [41]. Being built in Keras, Tensorflow enables fast development and the ability to change large parts of the program, but at the same time also take control over details if one would like [42]. Many useful tools often used with neural networks are already implemented in Tensorflow, for example all the layers explained in section 2.1 [43]. Tensorflow also gives the possibility to save a trained model to be used for predictions (or continue training) later [44].

2.2.3 File formats

HDF5 is a file format enabling fast processing of structured data [45]. All log files used in this thesis are .HDF5-files, both those acquired from the simulation software described in section 2.3 and from real test drives. In Python .HDF5-files can be read with the h5py module [46].

2.3 Simulating vehicle dynamics

This section will provide the important information about the simulation software central to this thesis. The software is a closed-loop vehicle dynamics simulation tool used by Zenseact. It consists of multiple parts, three of which are of interest for this thesis and will be explained later. Here neither the full functionality nor complete lists of inputs and output to each component will be given, but only what is important for the thesis. What to investigate in this thesis is how well the network can predict the output from 1) the Controllers, 2) the Vehicle model and 3) the two combined, essentially predicting the output of the Closed loop from its inputs. A schematic of the important parts of the software can be seen in figure 2.9.

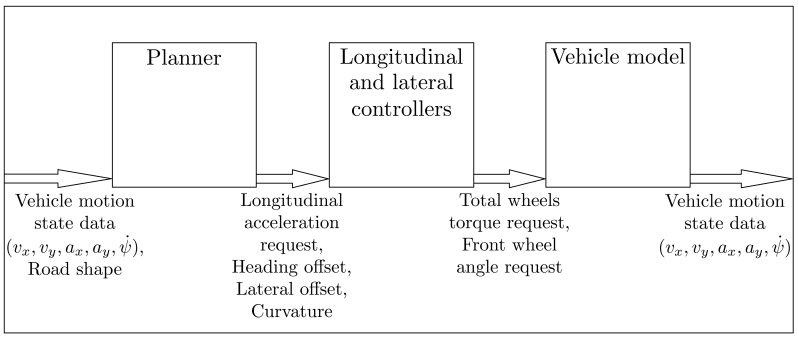

Figure 2.9: The important parts of the simulation software for this thesis, namely the Planner, Controllers and Vehicle model.

When running a simulation, a user can control the behavior of the vehicle and shape of the road from a set of configuration files. For example, the road can be set to have the shape of a sine curve with a certain amplitude and frequency. In the way the software will be used in this thesis, the relevant longitudinal parameters are initial velocity and a request of how the acceleration should change during the simulation. Laterally, the controls will be a path or simulated road that the Controllers will try to keep the vehicle in the center of. After running the simulation, important data as the inputs and outputs to the different parts of the software is stored in a .HDF5 file. This gives it the same structure as log files generated in real test drives, making it seamless to switch between analyzing generated and real data.

2.3.1 Ego and global frame

When choosing coordinate system for objects in simulation, there are multiple choices. Two of them, *ego frame* and *global frame*, will be explained here. Ego

frame is defined by a positive x-axis parallel to the heading of the vehicle (the *longitudinal* direction) [47]. The y-axis is orthogonal to the x-axis (the *lateral* direction) so that the origin is at the vehicle's rear axle. For the purpose of this thesis, only two dimensions will be considered. While the position and heading of the vehicle updates during the simulation, a coordinate system in ego frame moves along with it. In ego frame the vehicle's coordinates are therefore always $(0, 0)$. *Global frame* is defined as a coordinate system fixed at a static origin [48]. Positive X-direction is kept at the direction equaling the initial heading of the vehicle and positive Y-direction orthogonal to the left, in order to keep positive orientation. When the vehicle is simulated driving, the road keeps its coordinates, but both the coordinates and heading of the vehicle changes. A visual comparison between global and ego frame can be seen in figure 2.10.

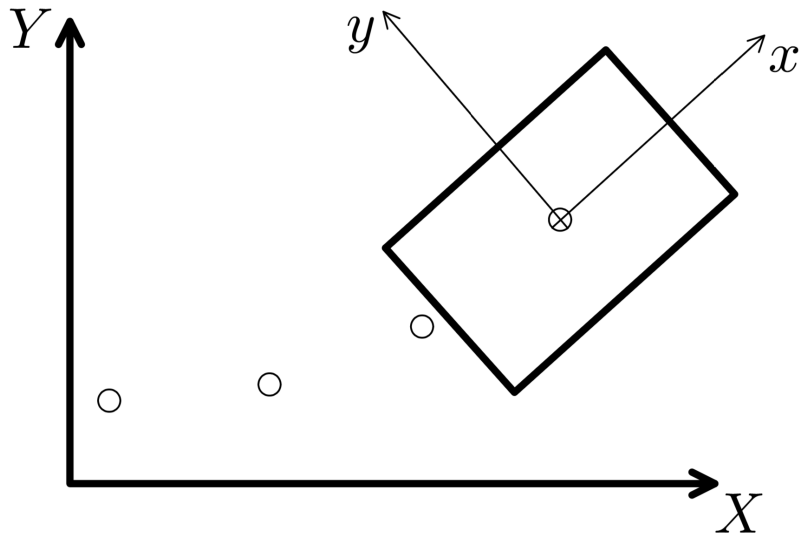

Figure 2.10: Difference between ego frame and global frame. The smaller circles is the coordinates of the vehicle's center of mass over time. The global coordinate system is constant in time, with the X- and Y-axes fixed and the vehicle's coordinates changing. Ego frame is the coordinate system fixed on the vehicle's center of mass, with positive x-direction pointing at the heading and positive y-axis orthogonal to the left.

2.3.2 Planner

The first part of the software is the planner, which as the name suggests, calculates the plan of how the vehicle should behave based on how it should drive in the near future. To do this, it inputs the shape of the road ahead and the current vehicle motion state data (the current dynamics of the vehicle, $v_x, v_y, a_x, a_y, \dot{\psi}$). The output is a longitudinal acceleration request and a list of quantities defining the vehicle's position in global frame. In this thesis, these are lateral offset, heading offset and curvature. Lateral offset in global frame is defined as the distance in the y-direction of the global coordinate system between the vehicle and road. Heading offset is the angle between the heading of the vehicle and the road and lastly the curvature defines how much the road currently is turning.

2.3.3 Controllers

After the planner there are two Controllers, one longitudinal and one lateral. The purpose with these are to convert the requests sent from the planner, together with how the vehicle moves, to output the signals used by the next part, the Vehicle model. One of the tasks for the network will be to replicate the output of the Controllers from their inputs. Since the Controllers are after the planner, they input the outputs from the Planner, i.e. the longitudinal acceleration request, curvature, heading offset and lateral offset. The Controllers outputs a request of angle for the front wheel (see section 2.3.4) and longitudinal torque request.

2.3.4 Vehicle model

There exist many models for simulating vehicle dynamics, one type of them being *Bicycle models* [49]. Although there are many versions of bicycle models, the shared property is the approximation of concatenating the front and rear wheels to instead simulate, in practice, a bicycle [50]. An example of this can be seen in figure 2.11. The two input signals deciding the time evolution of the model are the longitudinal tire forces and front wheel angle [48]. In this report one type, the linear bicycle model, will be used. Except for the shared approximation, the linear bicycle model also assumes constant longitudinal velocity, or at least quasi-stationary velocity, and small slip angles [51]. The equations for the velocities, accelerations, yaw rate and β, the angle between the x-axis and v, are

$$\begin{aligned} v_x &= v\cos\beta, \\ v_y &= v\sin\beta, \\ a &= \dot{v}, \\ \dot{\psi} &= \frac{v}{l_r}\sin\beta, \\ \beta &= \arctan\left(\frac{l_r}{l_r + l_f}\tan\psi\right), \end{aligned}$$

where ψ is the yaw angle, l_r and l_f are the distances from the center of gravity to the rear and front axle, respectively [52],[53].

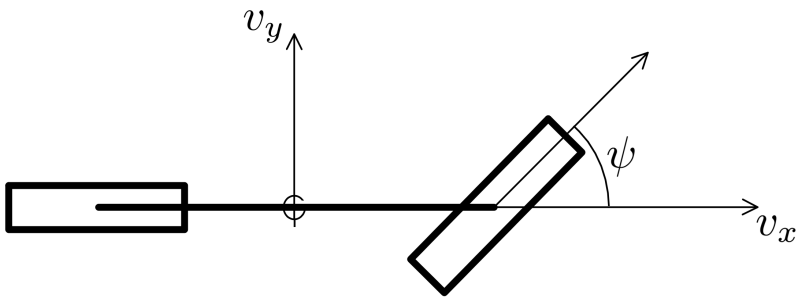

Figure 2.11: Approximation of a four-wheeled vehicle with the front and rear wheels each replaced by one single wheel at the line parallel with the vehicle's heading and intersecting the center of gravity. This way of modelling vehicle dynamics is called bicycle models.

The last part important for the thesis is the Vehicle model, which outputs the physical quantities describing how the vehicle will behave depending on the previous signals in the simulation. This is the other component that the network will try to predict the outputs of. It consists of two parts, one longitudinal motion model and one lateral. The longitudinal uses Newton's second law to calculate the dynamics. The lateral is on the other hand based on a linear bicycle model and therefore inputs the longitudinal torque request and front wheel angle request sent out by the Controllers. It outputs the vehicle motion state data. In the software it also has the current vehicle motion state data as inputs, but not all of them will be used for the network. However, since it is impossible to calculate velocity from a torque (which is equivalent to an acceleration), the current longitudinal and lateral velocities will also be used as inputs to the network. The reason why not all five quantities ($v_x, v_y, a_x, a_y, \dot{\psi}$) were included was that tests performed showed that they were not needed to get good predictions and therefore would lead to an unnecessary, essentially, doubling of input data.

3

Method

This section will explain how the theory and background which was reviewed in the previous chapter was actualized in order to acquire the results which will be presented later. The method chapter is split in two main sections, firstly the choices made connected to the structure of the network and hyperparameters will be presented. Afterwards the method associated with the data generation and dataset used will be disclosed.

3.1 Constructing the network

In this part of the report the method for developing the network will be described.

3.1.1 Initial setup

After reviewing the landscape of possible options of programming languages and ways to implement machine learning, the choice landed on Python since it is a high-level language with many powerful modules, which makes it fast to develop in. For developing the network, the machine learning ecosystem Tensorflow was used. In Python it can be downloaded and accessed as a module. During the development is was realized that the training times never became unreasonable due to the rather small datasets used. It was therefore decided to keep the training local and the laptop's built in NVIDIA RTX A4000 8 GB was used. This also has the benefit of making it easier for the Zenseact employees to the retrain the network in the future without having to use a GPU-cluster.

3.1.2 Choosing network model and hyperparameters

Because of LSTMs known abilities to use temporal data to gain information of the time evolution of the system, the main idea of the structure of the network was to first feed the normalized inputs to one, or a series of, LSTM-layers. After those, the output would be fed into one, or a couple of, FF-layers before the output. The normalization was done on all features individually by subtracting the mean and dividing by the standard deviation of the training set. One idea was also to append parallel FF-layers as branches after the shared FF-layers. These would then only be trained to predict one quantity each, with the predicted benefit of more accuracy with the downside of longer training times. However, after some tests it was concluded that the parallel layers gave worse prediction performance than just larger shared FF-layers and were therefore not used.

The first idea was to use the Adam optimizer proposed by Kingma and Ba [54]. It is based on stochastic gradient descent and uses adaptive learning rates based on moments combined with RMSProp [55]. However, inspired by [20], a test was made with a more classic stochastic gradient descent optimizer with *Nesterov accelerated gradient method*, which is supposed to converge faster than regular gradient descent [20], and about as fast as Adam [54]. This was observed, but SGD with Nesterov and 0.9 momentum gave slightly lower loss than Adam and was therefore used. It is already implemented in Tensorflow with the SGD Keras-layer and nesterov-argument [56].

As described in section 6.5 in [20], there could be benefits of not having a static learning rate. For example, smaller learning rates have a chance of getting stuck in a local minima, while large learning rates might not converge. Because of this, the plan was to use a custom dynamics learning rate. It was made based on exponential decay and to further help breaking loose from local minimas, a stochastic parameter that each epoch might cause the learning rate to reset to the initial one was implemented.

However as will be reported in section 3.1.3, it was possible to use a relatively high learning rate, leading to the stochastically peaking learning rates not being needed. The exponential decay, on the other hand, was kept.

To prevent both the over- and underfitting problem explained in section 2.1.3, a form of early stopping with patience was used. Instead of the more basic approach explained in section 2.1.5, that immediately aborts the training when the validation loss increases by an $\varepsilon : 1 \gg \varepsilon > 0$, it started an epoch based patience timer. When the network was trained, it calculates the linear trend in validation loss over the last n epochs, where n is a number set when initializing the training. If the trend turned positive, meaning that the validation loss was increasing, the training was stopped and the weights corresponding to the best validation loss of the entire training session was returned. The reason this more advanced algorithm was used is that sometimes during the training, the validation loss could have points of increase, or almost constant values, before starting to decrease again. Aborting at this point would have led to a slightly underfit model. An example of this is shown later, in figure 4.14. During the training the validation loss increased many times but the trend was negative. There was also a problem with increasing validation loss the first couple of epochs before then starting to decrease in some training sessions. This was solved by implementing a variable that decides the first epoch the validation loss patience should start measuring from. Note that the validation losses were saved during these first epochs as well, so the weights corresponding to the best validation loss still were returned. Some test were made with drop out layers to prevent overfitting, but none of those performed better than those without. This also correlates well with the results in [54], showing that SGD with Nesterov performs worse with drop out than without.

Depending on the task the network should accomplish, a basic hyperparameter search was carried out by starting from small values and exponentially increasing them until reaching degrading performance, or in the case of number of neurons and layers, until training times became too long to be trained locally in a reasonable time. With for example a GPU-cluster larger networks could be trained but using such methods would oppose the purpose stated in section 1.2. For example, firstly trying one LSTM layer with 8, 32, 128 and 512 neurons, each with a FFNN layer with 8, 32, 128 and 512 neurons with an initial learning rate of 1×10^{-2}, 1×10^{-3}, 1×10^{-4}, 1×10^{-5} and 1×10^{-6}. Then doing the same for two RNN layers and two, three and four FF-layers, until a solid foundation of the different structures performance and a choice of parameters could be made.

3.1.3 Final network model and hyperparameters

For all three tasks learned (Controllers, Vehicle model and Closed loop), the same network structure was used. After testing different hyperparameters, it was concluded that the best was a larger LSTM layer followed by double same sized dense layers. A good choice based on testing was 128 LSTM neurons and then two fully connected 32 neuron dense layers with Leaky ReLU activation functions before a single output neuron per target. It was noted that even a relatively high initial

learning rate of $lr_0 = 1 \times 10^{-2}$ allowed converging. With this, an exponential decay of $lr = lr_0 e^{-i/60}$ was used, where i is the epoch. The high learning rate consequently made its stochastic reset unnecessary and this was therefore disabled. Each training session lasted 100 epochs and the model with lowest validation loss was used for evaluation of prediction performance. For each dataset and prediction task, the training and evaluation process was done ten times to ensure a transparent presentation of the average performance.

3.1.4 Evaluation metrics

In the training process the normalized mean squared error, NMSE, was used as loss function and normalized mean absolute error, NMAE, as extra metric. These can both be calculated from their non-normalized forms, i.e.

$$E_{\text{mse}} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2,$$
$$E_{\text{mae}} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|,$$

where y_i was the ith target and $\hat{y}_i$ was the ith prediction of that target. From this the normalized metrics were acquired from dividing by the standard deviation of the targets,

$$E_{\text{nmse}} = \frac{E_{\text{mse}}}{\text{std}(y)},$$
$$E_{\text{nmae}} = \frac{E_{\text{mae}}}{\text{std}(y)},$$

where the standard deviation was calculated as

$$\text{std}(y) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2},$$

where $\bar{y}$ was the mean of y, calculated per quantity. The choice of NMSE as loss and NMAE for secondary metric was used to harder penalize large errors in training, while also being able to monitor the mean differences. Both of these were evaluated on the validation set during training and on the test set after predictions. From every completed training, the gathered statistics for each quantity were total loss (NMSE), individual losses (NMSE) and NMAE.

$$E_{\text{nmse}} = \frac{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\text{std}(y)}$$

3.2 Data collection

Since one of the main topics investigated was which type of training data was best suitable to catch the most amount of vehicle dynamics, the data collection became

an important part of the thesis. This section provides information about the types of data generated and used, how it was structured for use as inputs to the network, and the reasons behind the choices made.

3.2.1 Generation of training and validation data

Since the purpose of the network was to predict how the simulation software would calculate the vehicle dynamics, the datasets used for training and validation were generated with the simulation software. The parameters that were tuned in each simulation were first and foremost the initial longitudinal velocity, scheme for how the longitudinal acceleration should be changed in time and the path the vehicle should follow. Parameters that on the other hand always kept constant initial values were heading, position, lateral velocity and lateral acceleration, $\psi_0 = x_0 = y_0 = v_y^0 = a_y^0 = 0$. Three datasets were used. Firstly, since one of the assumptions of the bicycle model is zero longitudinal acceleration, this was the first way the inputs were set up. As a first test, v_x^0 was set to 60 km/h and the vehicle was simulated for 65 s. To excite the lateral dynamics in the simulation, the road was made a sine wave. This was chosen to let the vehicle turn at chosen frequency and angles at different velocities, delivering more data in each simulation. In this case, roads with the lateral amplitudes and frequencies given in table 3.1 were used. The reason for these values is that using higher frequency for the higher amplitudes gave too high lateral accelerations ($> 4.5\,\mathrm{m/s^2}$) [57]. In the rest of the report, this will be referred to as the "Constant" dataset. There was also an idea to generate data at different velocities in order to give the network more information of the vehicle dynamics. In this case the data was simulated at constant velocity from 10 km/h to 130 km/h with 20 km/h steps and the same sinusoidal roads. In the end, this dataset was not used since it gave worse performance than the dataset only consisting of $v_x = 60$ km/h.

Secondly, another dataset was made using a sinusoidally changing longitudinal acceleration of $a_x = 2\sin(0.02t)\mathrm{m/s^2}$. Because of this, it got the name the "Sine" dataset. In this case two initial longitudinal velocities were used, 60 km/h and 120 km/h. The used parameters for the sinusoidal road can be seen in table 3.2, again with some excluded due to high lateral acceleration. The sine acceleration was chosen because the vehicle would with it turn at a chosen frequency and angles at different velocities in each simulation, delivering more information about its dynamics than at constant velocity.

One downside with the sine road shape was that they only had one set frequency and amplitude per simulation. Could more nuanced road shapes provide even more information of vehicle dynamics? To answer this, the third and final dataset therefore kept the sinusoidal longitudinal acceleration, but also introduced roads in the form of chirp-waves. Shortly, a chirp wave is as a sine wave, but with the frequency changing as a sine wave as well. Because of this road shape, the dataset will be identified as the "Chirp" dataset. This made the vehicle both turn with different frequencies and velocities in each simulation, with the hypothesis that this data would be more dense with information about the vehicle dynamics. The downside with this approach was that the simulation time initially used for the previous tests (65 s) was

not enough for both chirp wave frequency to cover the desired span of frequencies and for the vehicle to do at least two turns before the frequency changed to much. The solution became to triple the simulation time to 200 s. Between each simulation the initial velocity was changed from 20 km/h to 120 km/h with 20 km/h steps, a longitudinal acceleration of $a_x = \sin(0.05t)\text{m/s}^2$ and chirp frequency evolving from 0.01 Hz to 1 Hz.

Table 3.1: The chosen combinations of amplitudes and frequencies used in the Constant dataset, i.e. data with the vehicle following a sinusoidal road at a constant longitudinal velocity of 60 km/h.

Amplitude [m]	Frequency [Hz]
16	0.0625
16	0.031,25
4	0.0625
4	0.015,625
1	0.25
1	0.0625
1	0.015,625
0.25	0.5
0.25	0.125
0.25	0.031,25
0.0625	1
0.0625	0.25
0.0625	0.0625

3.2.2 Datasets used for training, validation and testing

In order to evaluate the performance of the network, both during training and for overall performance, a layout with training, validation and test set was used. The training and validation sets were derived from the data acquired from the generation described in section 3.2.1. The test set was a collection of 21 log files from real test drives, in order to evaluate how the network was performing on non-synthetic data, since that is what the network is supposed to be used for. Each training either the Constant, Sine or Chirp dataset was loaded, concatenated and batched. The training and validation set were cut into 30 time steps long batches with 80 % randomly selected for training and the other 20 % for validation. The test set was also batched into 30 time step slices, but each batch was just moved over one time step, leading to the network predicting all values (after the 30th time step) instead of only the 30th value. This gave in total 48,617 batches of 30 time steps.

When using the Constant dataset, only the lateral quantities were used as targets.

Table 3.2: The chosen combinations of amplitudes and frequencies used in the Sine dataset, i.e. data with the vehicle following a sinusoidal road, starting with the initial longitudinal velocities in the table and a sinusoidally changing longitudinal acceleration of $2\sin(0.02t)\mathrm{m/s^2}$.

$v_x^0 = 60\,\mathrm{km/h}$		$v_x^0 = 120\,\mathrm{km/h}$	
Amplitude [m]	Frequency [Hz]	Amplitude [m]	Frequency [Hz]
16	0.0625	8	0.125
4	0.125	8	0.031,25
4	0.031,25	2	0.25
1	0.25	2	0.0625
1	0.0625	2	0.015,625
1	0.015,625	0.5	0.5
0.25	0.5	0.5	0.125
0.25	0.125	0.5	0.031,25
0.25	0.031,25	0.125	0.25
0.0625	1	0.125	0.0625
0.0625	0.25	0.125	0.015,625
0.0625	0.0625		

This is because this dataset was generated with zero longitudinal acceleration and constant longitudinal velocity and the network therefore had no data to learn the longitudinal dynamics from. In practice this meant that current longitudinal velocity was not used as input and the longitudinal velocity or acceleration as output for predictions of the Closed loop and Vehicle model. For the Controllers, this implied not having the total wheels torque request as output and instead only using the front wheel angle request.

4

Results and Discussion

In this chapter the results of the thesis will be given, starting of with how well the network was able to predict the dynamics of the Closed loop, Controllers and lastly Vehicle model. For all three tasks the network parameters used were one LSTM layer with 128 neurons, followed by two dense layers with 32 neurons each and lastly an output layer with one neuron for each target. The network was trained with the Constant, Sine and Chirp datasets individually for 100 epochs each with 1×10^{-2} initial learning rate. For definitions of these datasets see section 3.2.1. The results in each section will first show three figures of the predictions of each network on the test set for that prediction task. For each permutation of dataset and prediction task the network was trained and evaluated ten times. From these values 95 %-confidence intervals of the NMSE for each quantity were calculated and their bounds will be presented in tables 4.1, 4.3 and 4.4 below. There will also be a figure showing one of the training and validation NMSE and NMAE together with a prediction on the validation set. In the first section, predicting the Closed loop, the results from a bonus test will also be shown. In this case the Chirp dataset was used for training, but all five quantities representing the current vehicle motion state as inputs, instead of just the velocity.

In the figures showing predictions on the test set, for example figure 4.1, large spikes in the predictions can be seen. In some occasions these spikes can be found at the same time in multiple quantities. The reason for this originate from the way all the log files in the test set first is concatenated and then split into batches. This will lead to the start of the batch being the end of one log file and the end and target being the start of the next file. Also note that these figures shows the real values of all quantities, but all measurements on the performance of the network are on normalized data.

4.1 Learning and predicting the Closed loop

To have the neural network learn both the Controllers and Vehicle model together was one of the main purposes of the thesis. In practice this implies predicting the vehicle dynamics $(v_x, v_y, a_x, a_y, \dot{\psi})$ from the longitudinal acceleration request, curvature, heading offset, lateral offset and current velocity. The first result of this can be seen in figure 4.1, where the network is trained on the Constant dataset. As previously mentioned, the predictions made with this dataset are only on lateral quantities, i.e. lateral velocity, acceleration and yaw rate in this case. As can be seen in table 4.1, there is not a best choice of dataset for this task. The network trained on the Constant dataset gave the best predictions lateral velocity, the Sine dataset was best for longitudinal velocity and the Chirp for longitudinal acceleration, considering the 95 %-confidence interval. In figure 4.2 an indication of the reason the metrics are not lower will be made. This shows that the network has a good ability to predict the pattern of the velocity, but less so with the amplitude. The acceleration and yaw rate also has a somewhat correct shape, but is delayed compared to the targets. In figure 4.3 and 4.4, the predictions of the network trained on the Sine and Chirp dataset are shown, respectively. Since the vehicle has a longitudinal acceleration in these datasets the longitudinal velocity and acceleration are also used as targets.

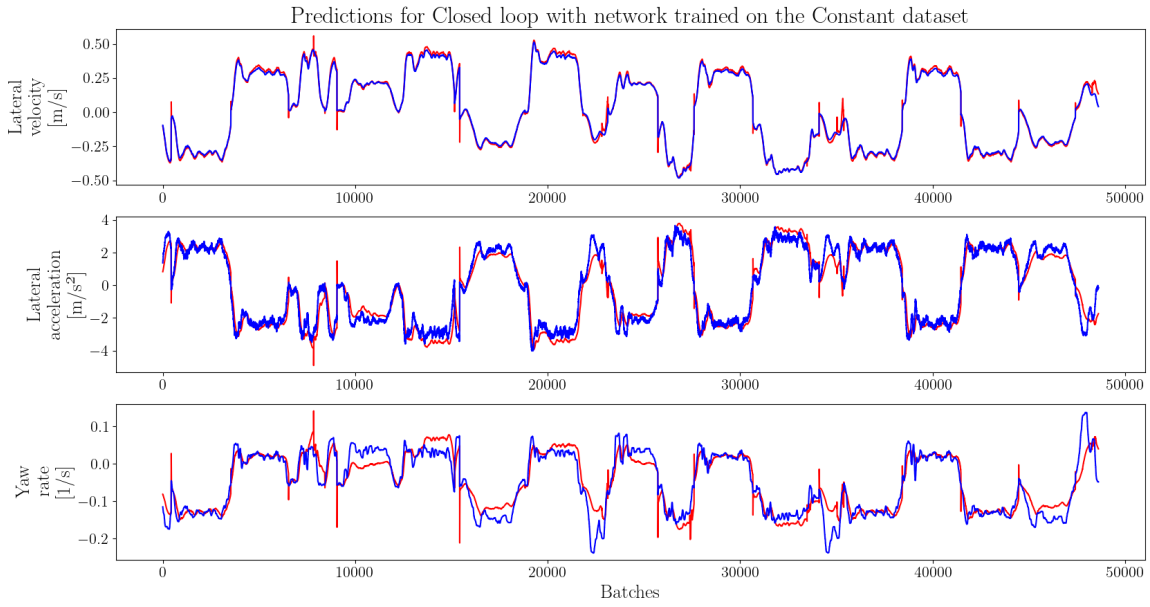

Figure 4.1: Predictions (red) made on the test set (blue) on the Closed loop. The network was trained on the Constant dataset. Since the data does not contain information about longitudinal vehicle dynamics, the longitudinal velocity and acceleration are not used as targets. The NMSE loss of each quantity are written in table 4.1.

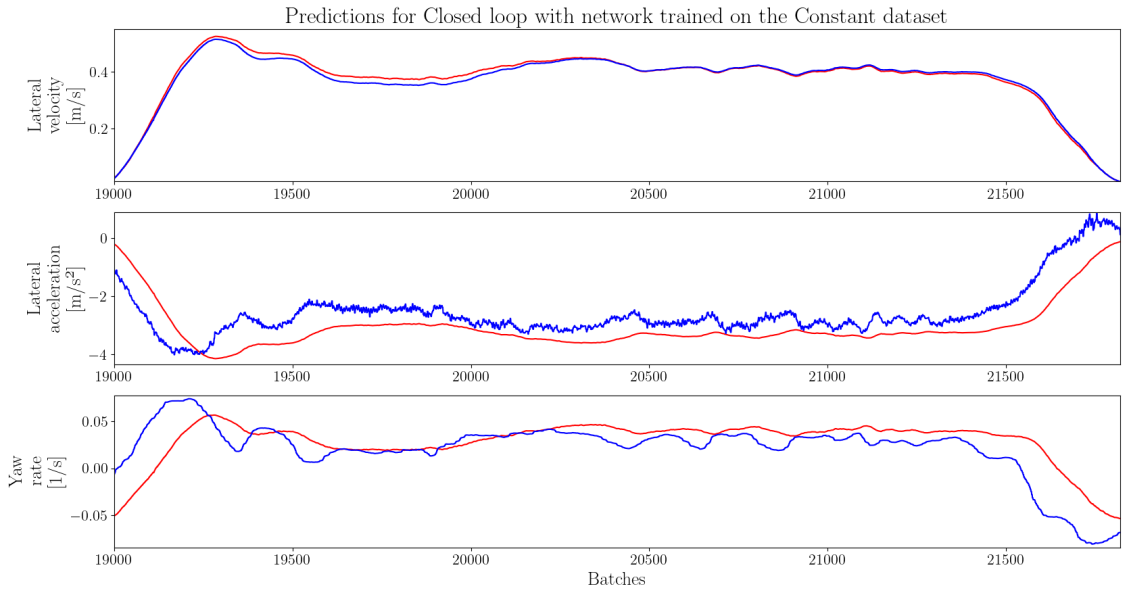

Figure 4.2: Zoomed in version of 4.1. Here two patterns that separates predicting from targets can be noted. The velocity, whose current state is an input, has a shape similar to the target but with a slightly different amplitude. The acceleration and yaw rate on the other hand has a shape roughly comparable to the target, but translated forward in time.

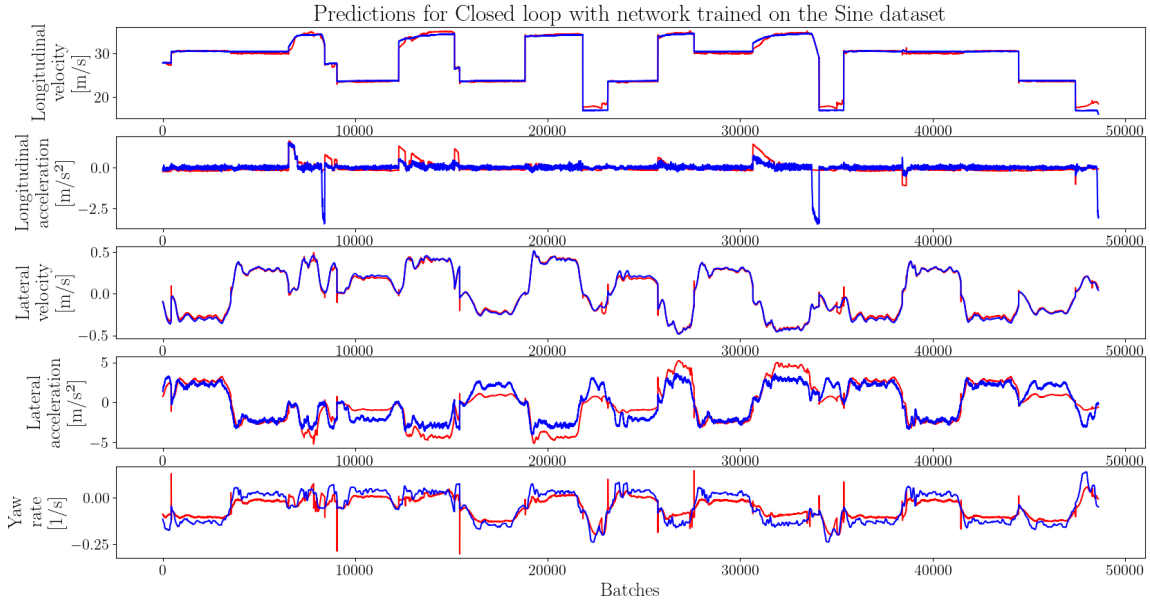

Figure 4.3: Predictions (red) made on the test set (blue) on the Closed loop. The network was trained on the Sine dataset. The NMSE loss of each quantity are written in table 4.1.

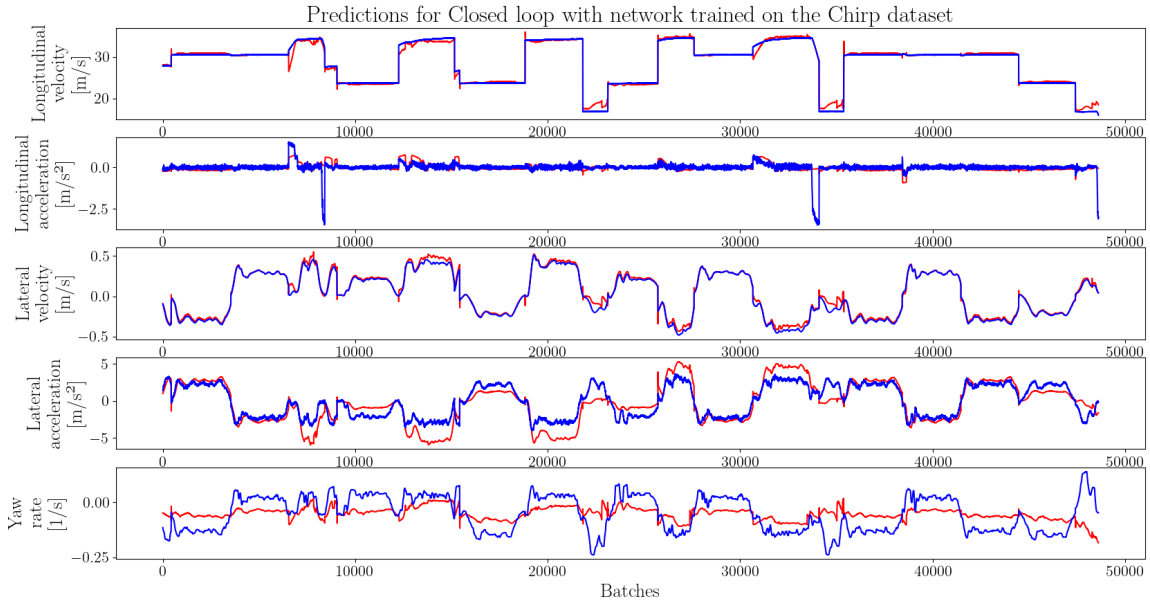

Figure 4.4: Predictions (red) made on the test set (blue) on the Closed loop. The network was trained on the Chirp dataset. The NMSE loss of each quantity are written in table 4.1.

Table 4.1: Comparison with 95 %-confidence intervals of the losses for all individual normalized quantities predicted by the network on the test set when trained on the Constant, Sine and Chirp dataset respectively. Note that the Constant dataset can not be used for predicting the longitudinal quantities.

Quantity	Constant NMSE [1]	Sine NMSE [1]	Chirp NMSE [1]
Longitudinal velocity	N/A	[0.007, 0.012]	[0.013, 0.017]
Longitudinal acceleration	N/A	[1.070, 1.120]	[0.988, 1.049]
Lateral velocity	[0.008, 0.016]	[0.066, 0.079]	[0.176, 0.234]
Lateral acceleration	[0.064, 0.072]	[0.065, 0.068]	[0.074, 0.099]
Yaw rate	[0.150, 0.183]	[0.153, 0.198]	[0.674, 0.850]

Next is figure 4.5 showing the training and validation metrics during training together with a prediction on the validation set. For this the Chirp dataset was used. What can be noticed is the phenomenon of increasing validation loss explained in section 3.1.2. This is not a problem since the network will continue the entire 100 epochs, but at the end only return the weights resulting in the lowest validation loss.

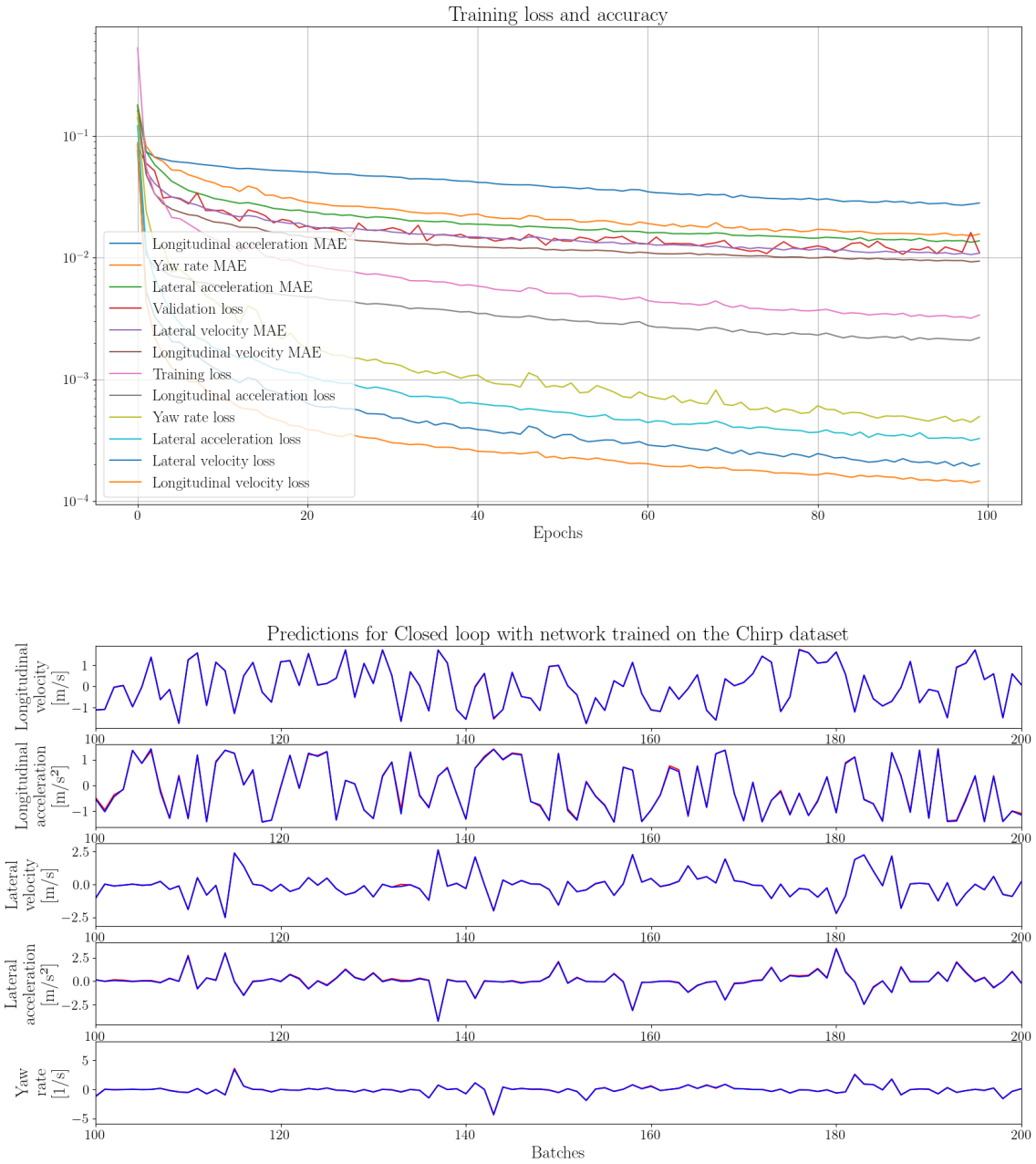

Figure 4.5: Training session with the Chirp dataset to predict the Closed loop. Above is the training and validation losses (NMSE) and NMAE of all quantities during the training. Below is an example of predictions on shuffled batches of the validation data after the training. The blue graphs are the targets and the red ones are the predictions.

Figure 4.5 was the last result that will be included in all result sections connected to prediction of a component. Here however a bonus result is also included, where the Chirp-trained network had the knowledge of current vehicle motion extended to also include the acceleration and yaw rate. These are inputs to the Controllers in the simulation software, but for the results in the rest of the report they were not included. How the predictions compare to the targets can be seen in figure 4.6.

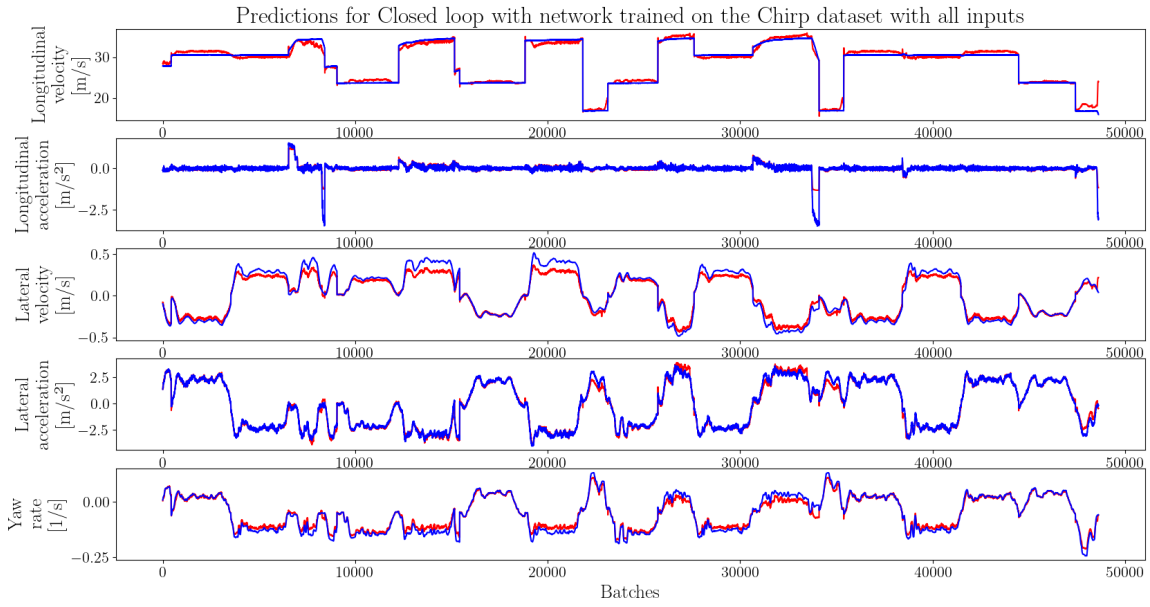

Figure 4.6: Predictions (red) made on the test set (blue) for the Closed loop, trained on Chirp data but with extra inputs. For this the current accelerations and yaw rate were also used as inputs. The NMSE can be seen in table 4.2.

When doing predictions on the test set, the quantities corresponding to the lowest losses in table 4.1 became the same or slightly worse. This would be both velocities and the lateral acceleration. Predictions on the longitudinal acceleration and yaw rate on the other had became much better, with both quantities upper bound of the 95 %-confidence intervals being about a third of the lower bounds of those in table 4.1. The list of confidence intervals for the losses acquired can be see in table 4.2.

Despite not being a part of the outlined method, this test contributed to the understanding of the capacity of the model. The previous predictions which only used the longitudinal and lateral velocity as inputs in general also had good accuracy for those two quantities. In this case however, it was expected that the predictions for the accelerations and yaw rate would improve. Examining some of the prediction plots, for example the lateral acceleration in figure 4.4, it could be argued that the network captured the relations in between its predictions, i.e. it has the same shape as the targets, but with incorrect amplitude. Using the accelerations as inputs therefore was believed to mainly help the network to get closer to the amplitudes of the targets, rather than the actual patterns. This seems to be a correct statement, and examining the plots more closely it was observed that the amplitude was mostly

Table 4.2: 95 %-confidence intervals of the losses for all individual normalized losses from predicting the Closed loop using the Chirp dataset and complete current vehicle motion state data as input.

Quantity	Chirp NMSE with $a_x, a_y, \dot{\psi}$ [1]
Longitudinal velocity	[0.021, 0.033]
Longitudinal acceleration	[0.261, 0.305]
Lateral velocity	[0.097, 0.141]
Lateral acceleration	[0.045, 0.073]
Yaw rate	[0.031, 0.052]

correct for these quantities, but still with a deficiency in the minor prediction details. This is interpreted as a saturation of the network capacity, more data does not give an actual "better prediction" in the sense that the dynamics are captured. This correlates well with section 3.2.1, explaining that the extended Constant dataset did not generate lower losses either. Both of these conclusions strengthen the theory above, that to get higher accuracy, multiple parts of the system will have to be extended to improve the performance.

4.2 Learning and predicting the Controllers

Before the network had been trained to do any prediction, the hypothesis was that predicting the output of the complete simulation could be challenging. This claim originates from the knowledge of the magnitude of non-linearity and complexity when comparing the Controllers against the Vehicle model. If that would have been the case, predicting the Controllers were also interesting, both in a research perspective and in the context of what it would bring to Zenseact.

As for predicting the Closed loop, the Constant dataset does not contain longitudinal data useful for these kinds of predictions, so when training the network in this dataset only the total wheels torque will be used as input. The result can be seen in figure 4.7. With the network trained on the Sine (figure 4.8) and Chirp (figure 4.9) dataset, both outputs of the Controllers were used as outputs on the network. The compiled list of losses for the three prediction scenarios are disclosed in table 4.3. The table shows that the Sine dataset gave slightly better prediction than the Chirp on the total wheels torque request, but because the intervals overlap this cannot be concluded with 95 %-confidence. For the angle request, the Chirp dataset gave the best prediction.

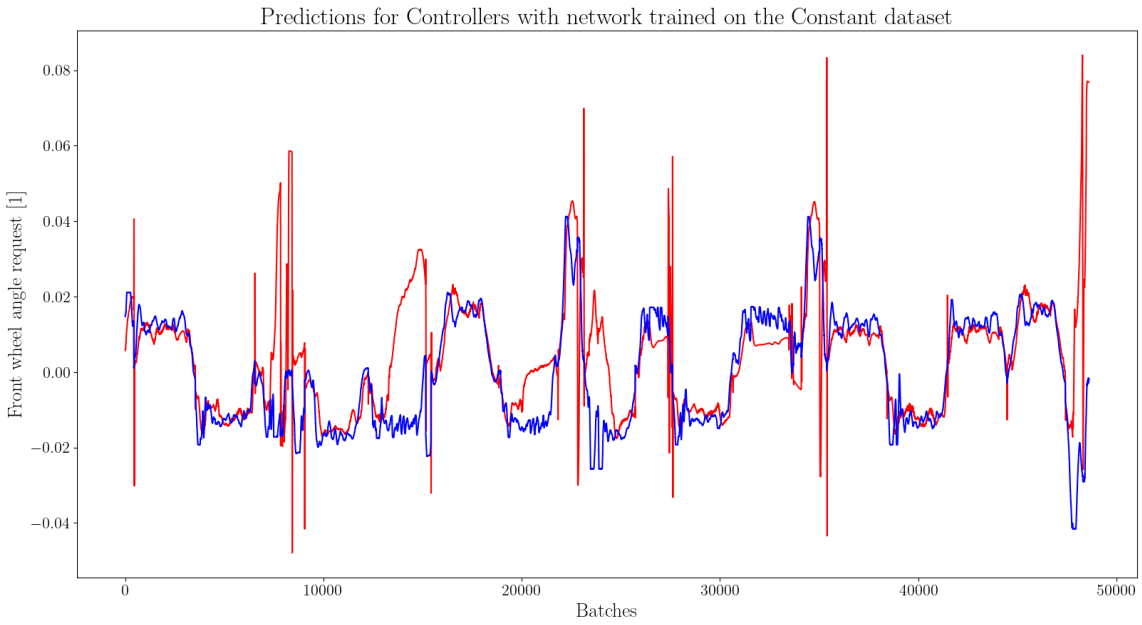

Figure 4.7: Predictions (red) made on the test set (blue) on the Controllers. The network was trained on the Constant dataset. Since the data does not contain information about longitudinal vehicle dynamics, the total wheels torque is not used as target. The NMSE loss for the front wheel angle request is written in table 4.3.

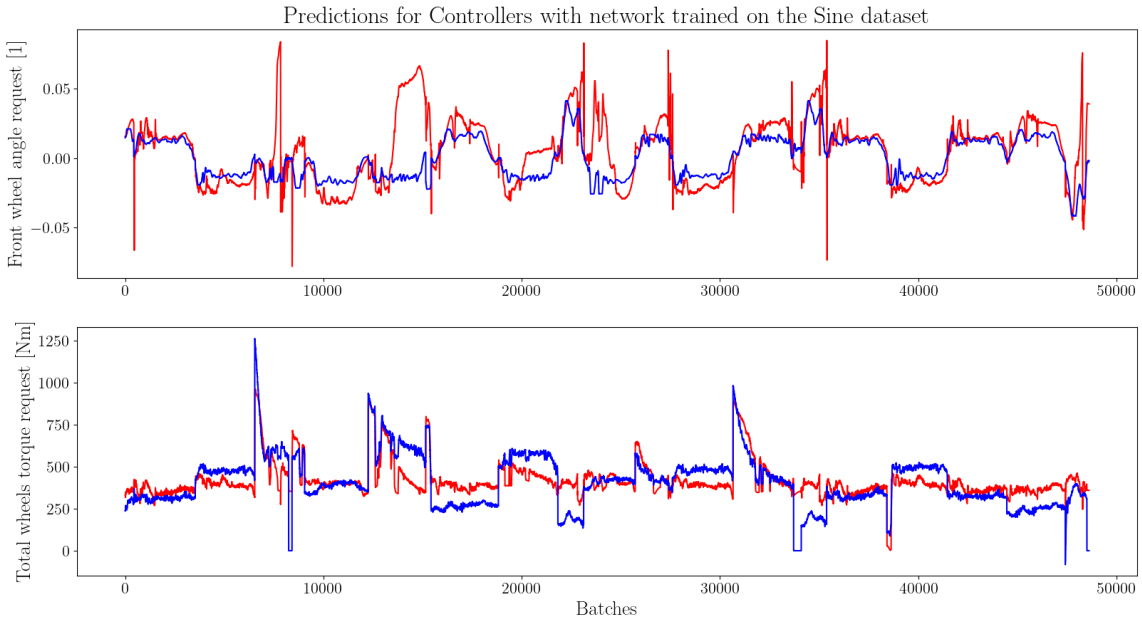

Figure 4.8: Predictions (red) made on the test set (blue) on the Controllers. The network was trained on the Sine dataset. The NMSE loss of the torque and angle request are written in table 4.1.

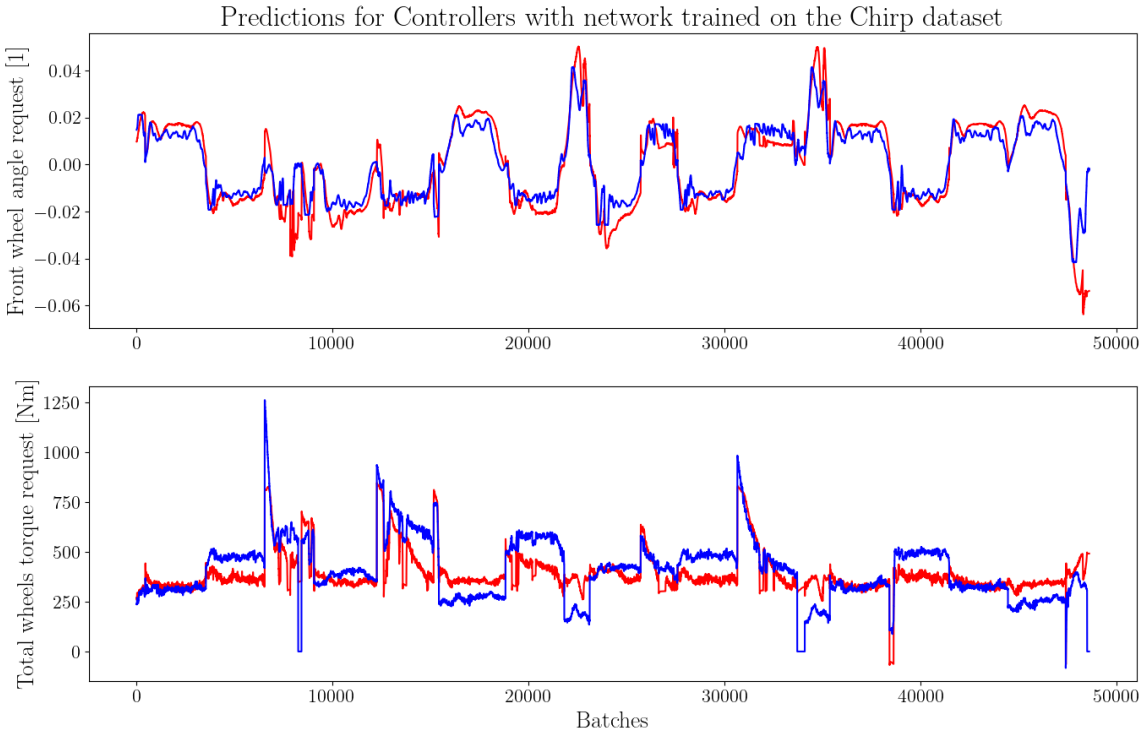

Figure 4.9: Predictions (red) made on the test set (blue) on the Controllers. The network was trained on the Chirp dataset. The NMSE loss of the torque and angle request are written in table 4.1.

Table 4.3: Comparison with 95 %-confidence intervals of the losses for all individual normalized quantities predicted by the network on the test set when trained on the Constant, Sine and Chirp dataset respectively. Note that the Constant dataset can not be used for predicting the total wheels torque.

Quantity	Constant NMSE [1]	Sine NMSE [1]	Chirp NMSE [1]
Total wheels torque request	N/A	[0.523, 0.581]	[0.561, 0.628]
Front wheel angle request	[1.480, 1.641]	[0.731, 1.003]	[0.226, 0.303]

Lastly, a training session on the Chirp dataset can be examined in figure 4.10. This shows the NMSE and NMAE loss for both quantities, total training and validation loss for each epoch, together with a prediction on a part of the validation set.

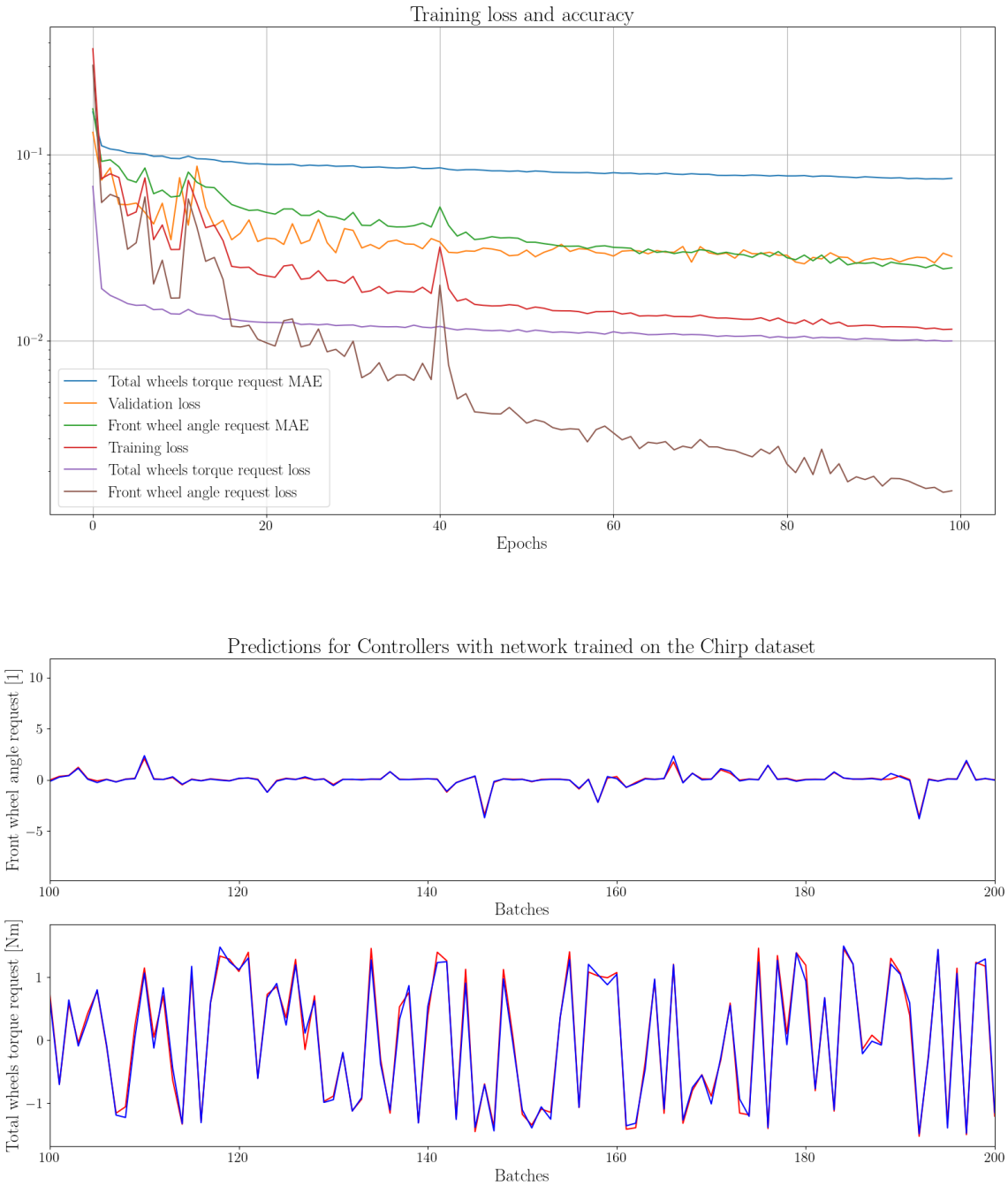

Figure 4.10: Training session with the Chirp dataset to predict the Controllers. Above is the validation and training losses (NMSE) and NMAE during the training and below is an example of predictions on shuffled batches of validation data after the training. The blue graphs are the targets and the red ones are the predictions.

4.3 Learning and predicting the Vehicle model

The third prediction task was learning the output of the Vehicle model, i.e. predicting the vehicle dynamics v_x, v_y, a_x, a_y and $\dot{\psi}$ from the total wheels torque request and front wheel angle request. As explained in section 4.2, having a network doing the same as the Vehicle model was the least important of the three tasks to learn in the perspective of value for Zenseact, but still was considered an interesting task to investigate.

As for predicting the Closed loop, the Constant dataset does not contain longitudinal data and will not be used for predicting these either. The result of predicting the other quantities can be seen in figure 4.11. With the network trained on the Sine (figure 4.12) and Chirp (figure 4.13) dataset, both outputs of the Controllers were used as output. How the training and validation metrics changed during a training session with the Chirp dataset and a prediction on the validation set can be seen in figure 4.14. The compiled list of losses for the three prediction scenarios are disclosed in table 4.4. The table shows that there is not a best choice of dataset for predicting the Vehicle model. The Sine dataset gave the network the best performance for longitudinal velocity and yaw rate. The Constant gave the best predictions for lateral velocity.

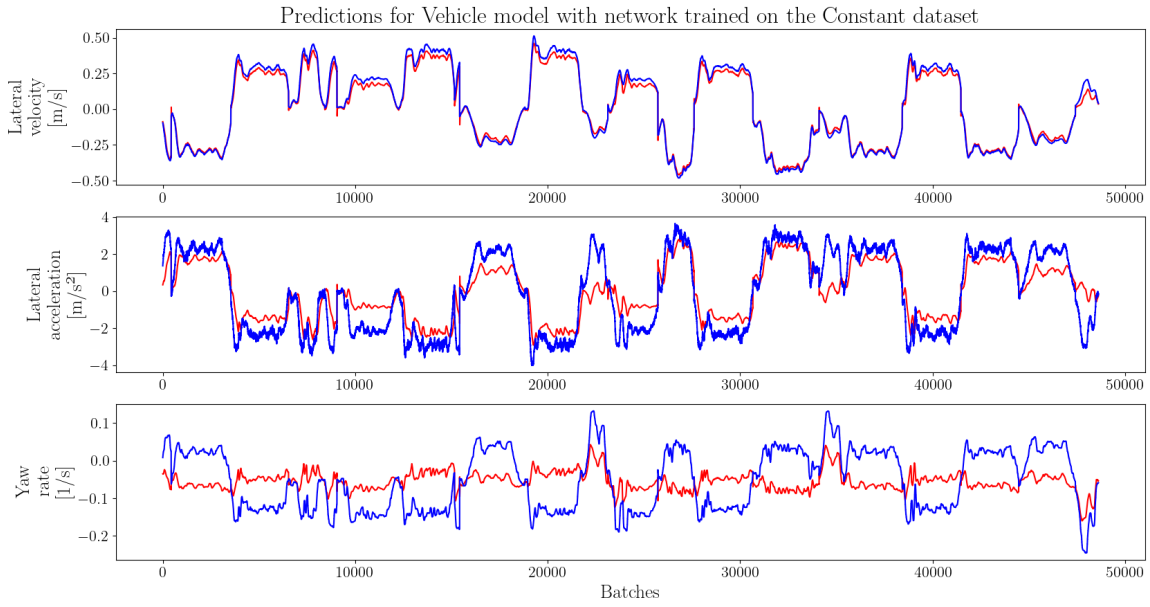

Figure 4.11: Predictions (red) made on the test set (blue) on the Vehicle model. The network was trained on the Constant dataset. Since the data does not contain information about the longitudinal vehicle dynamics, the longitudinal velocity and acceleration are not used as targets. The NMSE loss of each quantity are written in table 4.4.

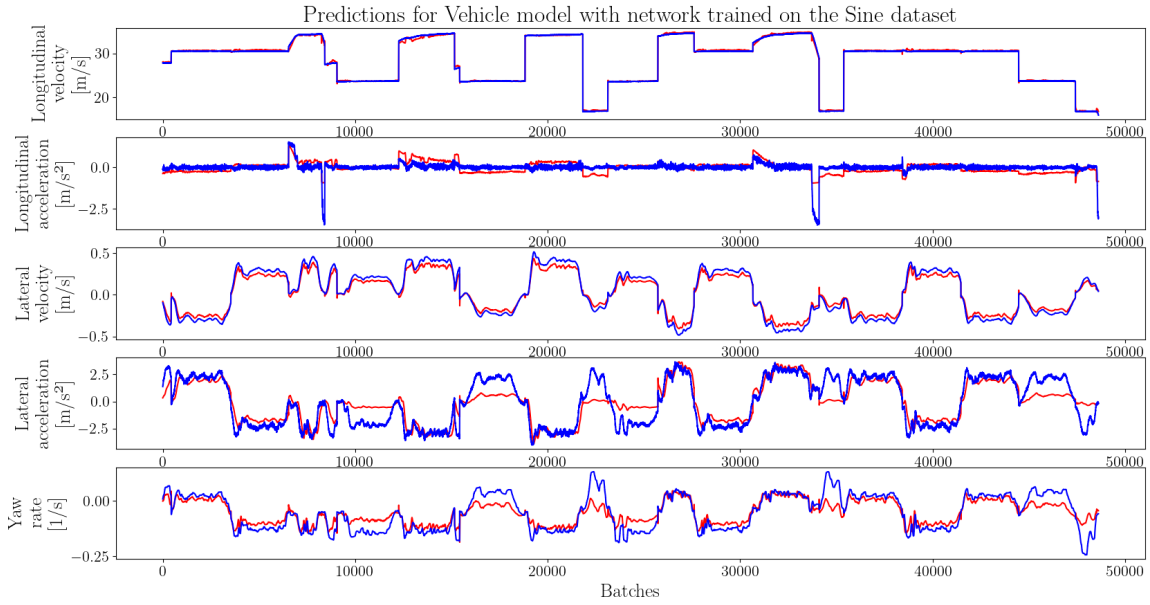

Figure 4.12: Predictions (red) made on the test set (blue) on the Vehicle model. The network was trained on the Sine dataset. The NMSE loss of each quantity are written in table 4.4.

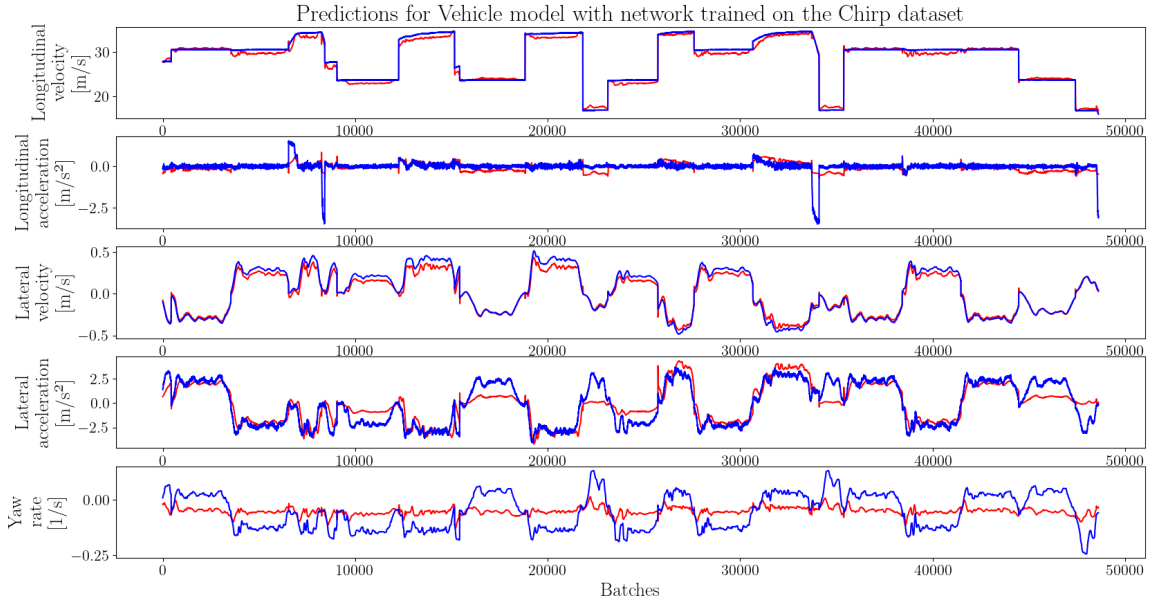

Figure 4.13: Predictions (red) made on the test set (blue) on the Vehicle model. The network was trained on the Chirp dataset. The NMSE loss of each quantity are written in table 4.4.

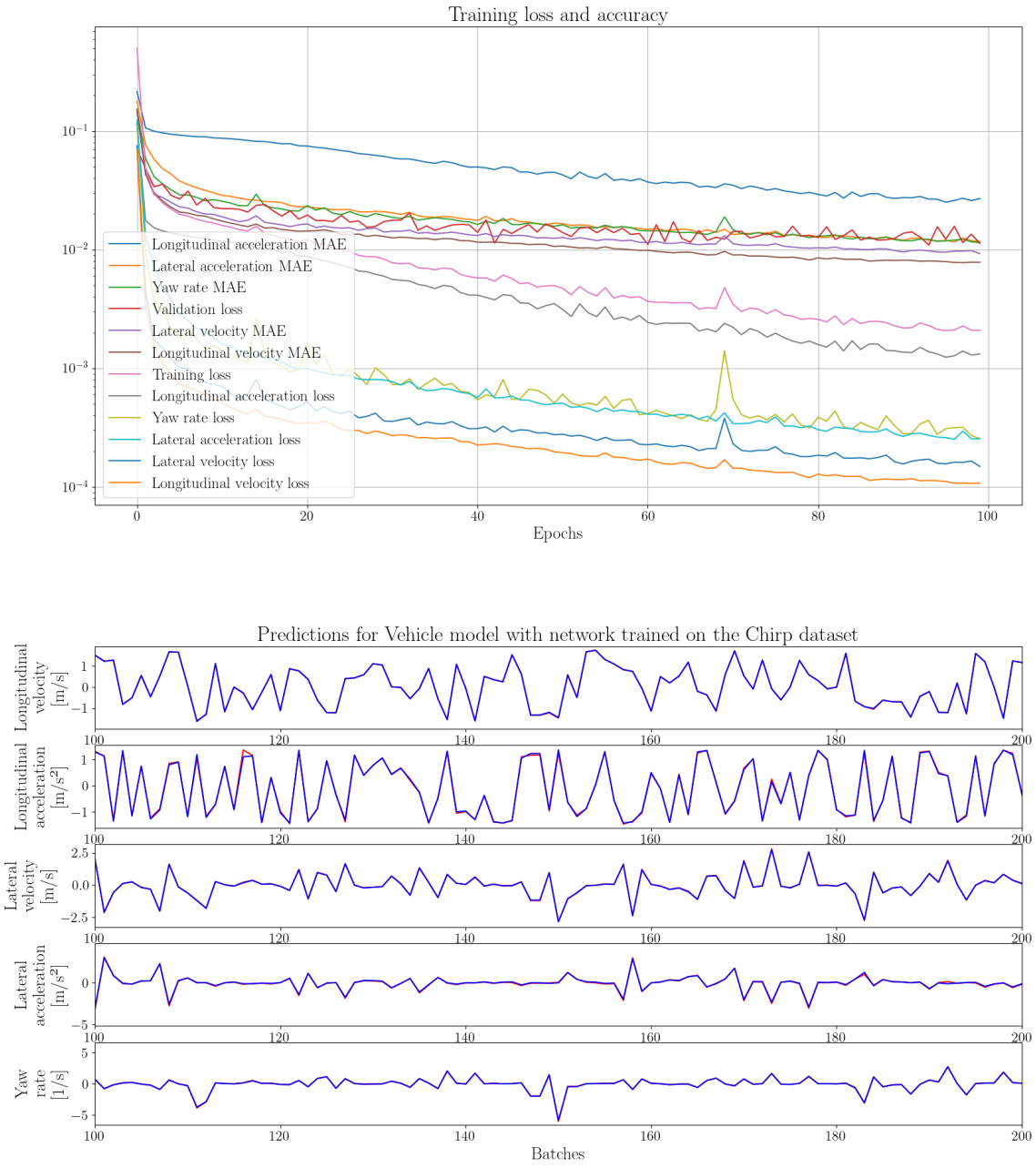

Figure 4.14: Training session with the Chirp dataset to predict the Vehicle model. Above is the training and validation losses (NMSE) and NMAE of all quantities during the training. Below is an example of prediction on shuffled batches of validation data after the training. The blue graphs are the targets and the red ones are the predictions.

Table 4.4: Comparison with 95 %-confidence intervals of the losses for all individual normalized quantities predicted by the network on the test set when trained on the Constant, Sine and Chirp dataset respectively. Note that the Constant dataset can not be used for predicting the longitudinal quantities.

Quantity	Constant NMSE [1]	Sine NMSE [1]	Chirp NMSE [1]
Longitudinal velocity	N/A	[0.0046, 0.0065]	[0.0104, 0.0218]
Longitudinal acceleration	N/A	[0.9339, 0.9728]	[0.9482, 1.1291]
Lateral velocity	[0.0039, 0.0102]	[0.0189, 0.0335]	[0.0384, 0.0776]
Lateral acceleration	[0.2032, 0.2472]	[0.2175, 0.2386]	[0.2436, 0.3218]
Yaw rate	[0.9190, 1.0894]	[0.2365, 0.3365]	[0.4945, 0.6597]

4.4 Comparison to simulation software

In this section a comparison between the simulation software and the network will be made. Starting of with the Constant dataset. It consists of twelve files with a 65 s simulation on each. Doing the simulations took each time between 8 minutes and 30 seconds to 9 minutes. This dataset consists of 84,528 time steps. When split 80−20 % for training and validation, this resulted in 2253 batches of 30 time steps for training and 564 batches for validation. With the 128 LSTM neurons, double 32 neuron dense layers and training to predict the Closed loop, corresponded in total to 73,891 trainable parameters or 288.64 kB. The five times tested for generating these results, the training took from 61−62 seconds. Since this is for the Constant dataset, the longitudinal quantities were not included. The total input list was therefore the curvature, heading offset, lateral offset and current lateral velocity. The outputs were the lateral velocity, acceleration and yaw rate. Using this dataset as a test set for prediction (which only was done for evaluating the time aspect), it was split to 84,495 batches of 30 time steps, each with four inputs and three outputs. The total prediction times were all between 6−7 seconds, corresponding to 75.9 µs on average per batch.

5

Conclusion

In section 4.1, 4.2 and 4.3 a neural network was trained on three different datasets in order to predict the output from the Closed loop, the Controllers and the Vehicle model, respectively. Three sets of data were used for training, the Constant (with only lateral data), Sine and Chirp dataset. Comparing the three types of predictions done, the two where the network had the task of predicting vehicle motion state data, i.e. the Closed loop and the Vehicle model, had the best results, apart from the longitudinal acceleration. Overall the datasets gave similar results in many cases and there is not a best choice for all applications. There is however the possibility that a dataset containing log files from different types of data could give even better results. For example a dataset with both the Constant and Sine data might give the network both useful longitudinal and lateral information, which then could transfer to more correct predictions. Depending on how other datasets are formed, the network structure and hyperparameters might benefit from a slight adjustment in order to draw full benefit from them.

However, the predictions still have errors and some predictions leave more to be desired. After trying many different methods, training sets and hyperparameters it became clear that these kinds of predictions, as was stated in section 1.3 regarding simulating vehicle dynamics, are hard problems. Even though there are several possibilities of which parts that should be changed for higher accuracy, some suggestions can be made. Since the tools used to prevent overfitting, as drop out layers, gave worse performance, this could tell that the number of parameters in the network is not too large for the problem at hand. At the same time, adding parallel branches of dense layers for each output did not help either, which would tell that the network does not need more parameters either. So is the network structure perfect? Most probably not, but since extensive searches of hyperparameters were made, this might be a good layout in the context it was tested in. I instead believe that higher accuracy could be achieved by generating diverse enough training data to give the network the possibility to learn a more complete view of the vehicle dynamics. The used datasets were generated in a way that would be easy and fast for Zenseact's engineers to recreate when the simulation software changes, but at the same time information dense. In that regard, the datasets used are considered adequate. Only containing a couple thousand data points, allowing the network to be trained in less than 2 minutes on a laptop graphics card, but still delivering good performance on most tested quantities. In testing it was noticed that the network got low validation loss early in the training sessions, for example as the one in figure 4.14. In the first

couple of epochs there was a large decrease in the validation loss, likely correlated with the high learning rate. These parameters worked well together and as a starting route to follow would be a larger and more diverse dataset, potentially with a larger network with lower learning rate. This would however most probably lead to longer training times, which then would be a tradeoff compared to the current system.

Instead of the current layout with LSTM and FF-layers, other ways to continue the development of the network were found. Depending on how the thesis was formed, the plan from the beginning was to start with the black box version presented in this report, and then compare it with another type of network. One idea was transfer learning [58], which could have been structured either as a network trained on simulation data and then on logs of real driving scenarios, or in the opposite, firstly trained on low noise data from real test drives and secondly fine tuned with simulation data, depending on the task it would have been used for. This could have proved very useful if the training times would have been long, but for the system developed which retrains in around less than 2 minutes, this is less of an argument. The other version that was discussed for this thesis was to compare the black box with a grey box. Originating in the hypothesis that it would have been hard to train a black box to do the Closed loop predictions and easier to predict the Controllers, the idea with the grey box would have been a network doing predictions for the Controllers and then feeding that input to the existing Vehicle model, or for example a simpler bicycle model. However, due to the good results for predicting the Closed loop and worse results for predicting the Controllers, this is less promising for further development. Something on the other hand that could give good results is a method proposed in [59], called *Hybrid Backpropagation Parallel Reservoir Networks*. It shows an interesting structure based on another type of RNN called *Reservoir computing*, which is supposed to outperform LSTM networks, both in prediction accuracy and needed resources.

To conclude, this thesis demonstrates a method for investigating how a deep learning network can be utilized for signal prediction in automated driving systems development and consistency analysis. With a goal of being useful in development, three datasets are generated to enable fast training and to be easy to replicate. The network is constructed by a combination of LSTM- and FF-layers in order to capture spatiotemporal vehicle dynamics and internal signals. Some errors do still exist in the predictions and other network structures [59] or types of vehicle driving schemes in the training data could be worth investigating in future work. Nevertheless, predictions of full Closed loop simulation are promising, both in prediction time and accuracy. Also managing predictions near real time (75.9 µs per batch), it has potential of being a core part in several tools where rapid predictions of vehicle dynamics can be crucial. One proposition is predicting signals for data in log files of real test drives to calculate KPIs and thereby identify discrepancies. Utilizing the same methodology could allow for a similar tool for real time discrepancy notification during test drives, or potentially to get direct response of how signals behave with a new software, instead of doing a full simulation. All together, this thesis is hoped to leverage the work for the engineers at Zenseact to develop software for a safer society without traffic accidents.

Bibliography

[1] Xiaolong Luo et al. "High-risk powered two-wheelers scenarios generation for autonomous vehicle testing using WGAN". In: *Traffic Injury Prevention* 26.2 (2025), pp. 243–251. DOI: 10.1080/15389588.2024.2399305.

[2] A L Dulmage and N S Mendelsohn. "Coverings of bipartite graphs". en. In: *Canad. J. Math.* 10.0 (1958), pp. 517–534. DOI: 10.4153/CJM-1958-052-0.

[3] Erik Frisk, Mattias Krysander, and Daniel Jung. "A Toolbox for Analysis and Design of Model Based Diagnosis Systems for Large Scale Models". In: vol. 50. 1. 2017, pp. 3287–3293. DOI: 10.1016/j.ifacol.2017.08.504.

[4] Daniel Jung. "A generalized fault isolability matrix for improved fault diagnosability analysis". In: vol. 2016-November. 2016, pp. 519–524. DOI: 10.1109/SYSTOL.2016.7739801.

[5] Arman Mohammadi, Mattias Krysander, and Daniel Jung. "Consistency-based diagnosis using data-driven residuals and limited training data". In: *Control Engineering Practice* 159 (2025). DOI: 10.1016/j.conengprac.2025.106283.

[6] Mattias Krysander, Jan Åslund, and Mattias Nyberg. "An Efficient Algorithm for Finding Minimal Overconstrained Subsystems for Model-Based Diagnosis". In: *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans* 38.1 (2008), pp. 197–206. DOI: 10.1109/TSMCA.2007.909555.

[7] Silke Merkelbach et al. "Using Multi-Modal LLMs to Create Models for Fault Diagnosis". In: *35th International Conference on Principles of Diagnosis and Resilient Systems (DX 2024)*. Ed. by Ingo Pill, Avraham Natan, and Franz Wotawa. Vol. 125. Open Access Series in Informatics (OASIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024, 31:1–31:15. ISBN: 978-3-95977-356-0. DOI: 10.4230/OASIcs.DX.2024.31.

[8] Renato Murata et al. "Fault Detection and Isolation Based on Structural Analysis: Application to a Multi-Engine Propulsion Cluster". In: *Sensors* 25.4 (2025). DOI: 10.3390/s25041054.

[9] Anna Sztyber-Betley et al. "Diagnosis test selection for distributed systems under communication and privacy constraints". In: *Applied Intelligence* 55.7 (2025). DOI: 10.1007/s10489-025-06543-w.

[10] C.G. Pérez et al. "Decentralized diagnosis in a spacecraft attitude determination and control system". In: vol. 659. 1. 2015. DOI: 10.1088/1742-6596/659/1/012054.

[11] Debbie Aisiana Indah et al. "Enhancing data efficiency for autonomous vehicles: Using data sketches for detecting driving anomalies". In: *Machine Learning with Applications* 15 (2024), p. 100530. ISSN: 2666-8270. DOI: 10.1016/

j.mlwa.2024.100530. URL: https://www.sciencedirect.com/science/article/pii/S2666827024000069.

[12] Andreas Lundgren and Daniel Jung. "Data-driven fault diagnosis analysis and open-set classification of time-series data". In: *Control Engineering Practice* 121 (Apr. 2022), p. 105006. ISSN: 09670661. DOI: 10.1016/j.conengprac.2021.105006.

[13] Kristofer D. Kusano et al. *Comparison of Waymo Rider-Only Crash Rates by Crash Type to Human Benchmarks at 56.7 Million Miles*. 2025. DOI: 10.48550/arXiv.2505.01515. arXiv: 2505.01515. URL: https://arxiv.org/abs/2505.01515.

[14] Matteo Rizzi et al. "How close to zero fatalities can Volvo cars get by 2020? An analysis of fatal crashes with modern Volvo passenger cars in Sweden". In: *Proceedings of the ESV Conference 2019*. 2019.

[15] Anders Lie, Claes Tingvall, and Maria Håkansson. "Automated vehicles: How do they relate to vision zero". In: *The Vision Zero Handbook: Theory, Technology and Management for a Zero Casualty Policy*. 2022, pp. 1057–1071.

[16] Kim Strandberg, Tomas Olovsson, and Erland Jonsson. "Securing the Connected Car: A Security-Enhancement Methodology". In: *IEEE Vehicular Technology Magazine* 13.1 (Mar. 2018), pp. 56–65. ISSN: 1556-6072. DOI: 10.1109/MVT.2017.2758179.

[17] Kendrick Hardaway, Oscar Teran, and Hua Cai. "Silent emissions: The cyber-infrastructure environmental impacts of autonomous vehicles". In: *Applied Energy* 392 (2025). DOI: 10.1016/j.apenergy.2025.125949.

[18] Md. Ebrahim Shaik, Md. Milon Islam, and Quazi Sazzad Hossain. "A review on neural network techniques for the prediction of road traffic accident severity". In: *Asian Transport Studies* 7 (2021), p. 100040. ISSN: 2185-5560. DOI: 10.1016/j.eastsj.2021.100040. URL: https://www.sciencedirect.com/science/article/pii/S2185556021000080.

[19] Saurabh Vyas et al. "Computation Through Neural Population Dynamics". In: *Annual Review of Neuroscience* 43.Volume 43, 2020 (2020), pp. 249–275. ISSN: 1545-4126. DOI: 10.1146/annurev-neuro-092619-094115. URL: https://www.annualreviews.org/content/journals/10.1146/annurev-neuro-092619-094115.

[20] Bernhard Mehlig. *Machine Learning with Neural Networks: An Introduction for Scientists and Engineers*. Cambridge University Press, 2021.

[21] xkcd. *xkcd: Machine Learning*. URL: https://xkcd.com/1838 (visited on 05/08/2025).

[22] Warren S McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". en. In: *Bulletin of Mathematical Biophysics* 5.4 (Dec. 1943), pp. 115–133. DOI: 10.1007/BF02478259.

[23] Thinagaran Perumal et al. "A comprehensive overview and comparative analysis on deep learning models". en. In: *Journal on Artificial Intelligence* 6.1 (2024), pp. 301–360.

[24] Iqbal H Sarker. "Deep learning: A comprehensive overview on techniques, taxonomy, applications and research directions". en. In: *SN Comput. Sci.* 2.6 (Aug. 2021), p. 420.

[25] Yann LeCun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2323. DOI: 10.1109/5.726791.

[26] Ernst Kussul and Tatiana Baidyk. “Improved method of handwritten digit recognition tested on MNIST database”. In: *Image and Vision Computing* 22.12 (2004). Proceedings from the 15th International Conference on Vision Interface, pp. 971–981. ISSN: 0262-8856. DOI: 10.1016/j.imavis.2004.03.008.

[27] Johannes Lederer. “Activation Functions in Artificial Neural Networks: A Systematic Overview”. In: *CoRR* abs/2101.09957 (2021). DOI: 10.48550/arXiv.2101.09957. arXiv: 2101.09957. URL: https://arxiv.org/abs/2101.09957.

[28] Koushik Biswas et al. “SAU: Smooth activation function using convolution with approximate identities”. In: *European Conference on Computer Vision*. Springer. 2022, pp. 313–329. DOI: 10.1007/978-3-031-19803-8_19.

[29] Saedeh Abbaspour et al. “A Comparative Analysis of Hybrid Deep Learning Models for Human Activity Recognition”. In: *Sensors* 20.19 (2020). ISSN: 1424-8220. DOI: 10.3390/s20195707. URL: https://www.mdpi.com/1424-8220/20/19/5707.

[30] Deepika Singh et al. “Human activity recognition using recurrent neural networks”. In: *Lecture Notes in Computer Science*. Lecture notes in computer science. Cham: Springer International Publishing, 2017, pp. 267–274.

[31] Tripti Kumari et al. “Weather Forecasting with Time Series Dataset Using Deep Learning Algorithms”. In: *Advances in Science, Technology and Innovation* (2025), pp. 99–104. DOI: 10.1007/978-3-031-72747-4_15. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85218690566&doi=10.1007%2f978-3-031-72747-4_15&partnerID=40&md5=204fe4549280bc7bb1076eb73ad2d7ce.

[32] Sepp Hochreiter and Jürgen Schmidhuber. “Long Short-Term Memory”. In: *Neural Computation* 9 (Nov. 1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735.

[33] Guillaume Chevalier. *File:LSTM Cell.svg*. 2018. URL: https://commons.wikimedia.org/w/index.php?curid=109362147 (visited on 05/17/2025).

[34] Xavier Guyon and Jian-Feng Yao. “On the underfitting and overfitting sets of models chosen by order selection criteria”. en. In: *J. Multivar. Anal.* 70.2 (Aug. 1999), pp. 221–249. DOI: 10.1006/jmva.1999.1828.

[35] Syed Arslan Ali et al. “An optimally configured and improved deep belief network (OCI-DBN) approach for heart disease prediction based on ruzzo–tompa and stacked genetic algorithm”. In: *IEEE Access* 8 (2020), pp. 65947–65958. DOI: 10.1109/ACCESS.2020.2985646.

[36] Richard Simon et al. “Pitfalls in the use of DNA microarray data for diagnostic and prognostic classification”. en. In: *J. Natl. Cancer Inst.* 95.1 (Jan. 2003), pp. 14–18. DOI: 10.1093/jnci/95.1.14.

[37] Lutz Prechelt. “Automatic early stopping using cross validation: quantifying the criteria”. en. In: *Neural Netw.* 11.4 (June 1998), pp. 761–767. DOI: 10.1016/S0893-6080(98)00010-0.

[38] Python Software Foundation. *About Python*. 2025. URL: https://www.python.org/about (visited on 05/02/2025).

[39] D Criado-Ramón, L G B Ruiz, and M C Pegalajar. "A parallel approach to accelerate neural network hyperparameter selection for energy forecasting". en. In: *Expert Syst. Appl.* 279.127386 (June 2025), p. 127386. DOI: 10.1016/j.eswa.2025.127386.

[40] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org.

[41] Keras. *About Keras 3*. URL: https://keras.io/getting_started/about (visited on 05/07/2025).

[42] TensorFlow. *Why TensorFlow*. URL: https://www.tensorflow.org/about (visited on 05/02/2025).

[43] TensorFlow. *Module: tf.keras.layers*. 2024. URL: https://www.tensorflow.org/api_docs/python/tf/keras/layers (visited on 04/29/2025).

[44] TensorFlow. *Save, serialize, and export models*. 2023. URL: https://www.tensorflow.org/guide/keras/serialization_and_saving (visited on 04/25/2025).

[45] HDF Group. *The HDF5 Library & File Format*. URL: https://www.hdfgroup.org/solutions/hdf5 (visited on 05/07/2025).

[46] HDF5 for Python. *HDF5 for Python*. URL: https://www.h5py.org (visited on 05/10/2025).

[47] Yanzhi Lv et al. "Autonomous overtaking decision and motion planning of intelligent vehicles based on graph convolutional network and conditional imitation learning". In: *Proceedings of the Institution of Mechanical Engineers, Part D* 239.2-3 (2025), pp. 930–945. DOI: 10.1177/09544070231206447.

[48] S Fuchshumer, K Schlacher, and T Rittenschober. "Nonlinear vehicle dynamics control - A flatness based approach". In: *Proceedings of the 44th IEEE Conference on Decision and Control*. Seville, Spain: IEEE, 2006. DOI: 10.1109/CDC.2005.1583203.

[49] P Riekert and T E Schunck. "Zur Fahrmechanik des gummibereiften Kraftfahrzeugs". de. In: *Ing. Arch* 11.3 (June 1940), pp. 210–224. DOI: 10.1007/BF02086921.

[50] Selim Solmaz et al. "Real-time multiple-model estimation of centre of gravity position in automotive vehicles". en. In: *Veh. Syst. Dyn.* 46.9 (Sept. 2008), pp. 763–788. DOI: 10.1080/00423110701602670.

[51] A T van Zanten et al. "Vehicle stabilization by the vehicle dynamics control system ESP". In: *IFAC Proc. Vol.* 33.26 (Sept. 2000), pp. 95–102. DOI: 10.1016/S1474-6670(17)39127-9.

[52] Ola Benderius. "Robot kinematics and dynamics". Chalmers University of Technology, TME290 Autonomous Robots, Lecture 7. May 2024.

[53] Rahul Prakash and Dharmendra Kumar Dheer. "Vehicle state estimation using a maximum likelihood based robust adaptive extended kalman filter considering unknown white Gaussian process and measurement noise signal". In: *Engineering Research Express* 5.2 (June 2023), p. 025066. DOI: 10.1088/2631-8695/acd73e.

[54] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG]. URL: https://arxiv.org/abs/1412.6980.

[55] Geoffrey Hinton. *Overview of mini-batch gradient descent*. 2012.

[56] TensorFlow. *tf.keras.optimizers.SGD*. 2024. URL: https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/SGD (visited on 05/10/2025).

[57] Kartik Seshadri Chari. "Dynamic Modelling and Optimal Control of Autonomous Heavy-duty Vehicles". MA thesis. KTH, School of Electrical Engineering and Computer Science (EECS), 2020, p. 77.

[58] Qiang Yang et al. *Transfer learning*. en. Cambridge, England: Cambridge University Press, Feb. 2020. DOI: 10.1017/9781139061773.

[59] Matthew Evanusa et al. "Hybrid Backpropagation Parallel Reservoir Networks". In: *CoRR* abs/2010.14611 (2020). DOI: 10.48550/arXiv.2010.14611. arXiv: 2010.14611. URL: https://arxiv.org/abs/2010.14611.

DEPARTMENT OF PHYSICS
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY