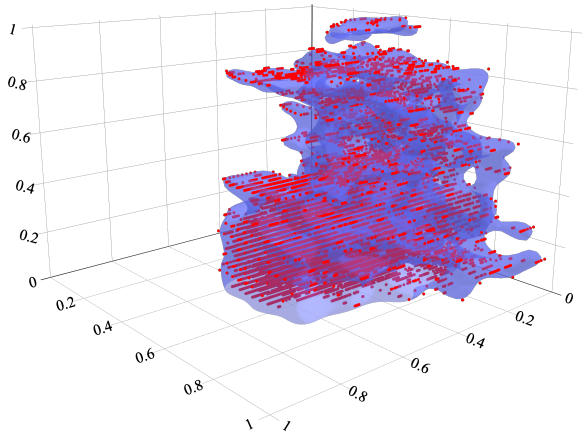
Robuste Machine-Learning-Modelle und deren Plausibilisierung für die Anwendung im PKW-Thermomanagement

Autor: Felix Alexander Korthals
Erstreferent: Prof. Dr.-Ing. S. Rinderknecht

TECHNISCHE UNIVERSITÄT DARMSTADT



MASCHINENBAU
We engineer future
IMS

Robuste Machine-Learning-Modelle und deren Plausibilisierung für die Anwendung im PKW-Thermomanagement

Am Fachbereich Maschinenbau
an der Technischen Universität Darmstadt
zur
Erlangung des Grades eines Doktor-Ingenieurs (Dr.-Ing.)
genehmigte

Dissertation

vorgelegt von
Felix Alexander Korthals, M.Sc.
aus Stuttgart

Berichterstatter:	Prof. Dr.-Ing. Stephan Rinderknecht
Mitberichterstatter:	Prof. Dr.-Ing. Steven Peters
Tag der Einreichung:	18.04.2023
Tag der mündlichen Prüfung:	27.06.2023

Darmstadt 2023
D17

MASCHINENBAU
We engineer future IMS

Felix Alexander Korthals: Robuste Machine-Learning-Modelle und deren Plausibilisierung für die Anwendung im PKW-Thermomanagement
Darmstadt, Technische Universität Darmstadt
Veröffentlichungsjahr der Dissertation auf TUprints: 2023
URN: urn:nbn:de:tuda-tuprints-264500
Tag der mündlichen Prüfung: 27.06.2023

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit, abgesehen von den in ihr ausdrücklich genannten Hilfen, selbständig verfasst habe.

Stuttgart, den 18. April 2023

Felix Korthals

Kurzfassung

Steigende Energiepreise und neue Gesetzesrichtlinien zum Klimaschutz haben die Automobilindustrie dazu gezwungen, ihren Fokus in der Antriebsstrangentwicklung deutlich zu verändern. Das bisherige Streben nach Leistungssteigerung beim Verbrennungsmotor wurde durch den Fokus auf eine Flotte mit geringem CO_2-Ausstoß und einem hohen Anteil an batterieelektrischen Fahrzeugen abgelöst. Die optimale Nutzung der Antriebsenergie steht dabei an erster Stelle und wird in allen Bereichen der Fahrzeugentwicklung angestrebt. Dabei kommt dem Thermomanagement eine wichtige Rolle zu. Durch eine prädiktive Temperaturregelung kann die verfügbare Energie sowohl bei Fahrzeugen mit Verbrennungsmotor als auch bei Fahrzeugen mit teil- oder vollelektrifiziertem Antriebsstrang effizient genutzt werden. Um energieeffiziente Lösungen zu entwickeln, sind die Software-Qualität und ein hoher Automatisierungsgrad essentiell. In Kombination mit datenbasierten Methoden kann in kurzer Zeit eine große Anzahl an Funktionen für unterschiedliche Varianten erstellt werden.
In dieser Arbeit wird ein automatisierter und datenbasierter Modellierungsprozess unter Verwendung von Fahrzeugmessdaten entwickelt. Dazu werden die vorhandenen Kommunikationsschnittstellen zwischen Fahrzeug und Cloud-Backend genutzt, um eine *Machine Learning* (ML)-Pipeline auf skalierbaren Cloud-Ressourcen zu entwickeln und zu implementieren. So können ML-Modelle mit einer besseren Modellgüte in einem automatisierten Ablauf erstellt und an das Fahrzeug gesendet werden. Dadurch wird der manuelle Kalibrierungsaufwand für die Regelungsmodelle reduziert. Bisher verwendete physikalisch motivierte Ersatzmodelle werden durch ML-Modelle substituiert. Durch die Automatisierung mittels der ML-Pipeline kann der gesamte Entwicklungsprozess von Regelungsmodellen beschleunigt werden.
Die benötigten Trainingsdaten werden vom Fahrzeug an ein Cloud-Backend übertragen und dort vorverarbeitet. In zwei Fallstudien werden geeignete künstliche neuronale Netzwerkarchitekturen untersucht und bewertet. Zusätzlich wird eine Methodik zur Plausibilisierung der Modellberechnungen eingeführt. Sie basiert auf den verwendeten Trainingsdaten und verknüpft zulässige Ausgangsdatenräume mit sogenannten *Backwards Reachable Set*s. Für Lipschitz-stetige *Feedforward Neural Network*s mit *Rectified Linear Unit*s kann so eine rückverknüpfte Plausibilitätsanalyse durchgeführt werden. Damit können Modellberechnungen als plausibel oder implausibel klassifiziert werden.
In der Anwendung liefern die ML-Modelle eine bessere Modellgüte als die bisher eingesetzten physikalisch motivierten Ersatzmodelle. Für den Zielanwendungsfall auf Vorausschaudaten lässt sich so eine genauere Berechnung des zu erwartenden Energieeintrags in den Kühlkreislauf erstellen. Die Verwendung der serienmäßig verfügbaren Fahrzeugkommunikationsschnittstelle und eines cloudbasierten Trainings ermöglicht eine automatisierte Anpassung der auf dem Fahrzeug ausgeführten Regelungsmodelle. Damit kann der Entwicklungsprozess in die Cloud verlagert werden. Durch skalare Gütekriterien und ein standardisiertes Vorgehen können Modelle mit höherer Güte in kürzerer Zeit bereitgestellt werden.

Abstract

Rising energy prices and new legislation on climate protection have forced the automotive industry to significantly change its focus concerning powertrain development. The previous pursuit of increasing the performance of combustion engines has been replaced by focusing on a fleet with minimal CO_2 emissions and a high proportion of battery electric vehicles. The optimal usage of the operating power is the most important task for all domains of vehicle development. The thermal management plays an important role in this system. A predictive temperature control can support to make use of the available energy in an efficient way. This is true for vehicles with internal combustion engine as well as for fully electrified powertrains. To develop energy efficient solutions, software quality and a high degree of automation is essential. Combined with data-based methods a large number of functions for different variants can be created in a short amount of time.
In this work an automated and data-based modeling process is being developed using vehicle measurement data. Therefore, the existing communication interfaces between vehicle and cloud backend are used to develop and implement an ML pipeline on scalable cloud resources. This way, ML models of better model quality can be created in an automated workflow and can be transferred to the vehicle.
This is reducing the manual effort of calibrating control models. Currently used physically motivated substitute models are replaced by ML models. By using the automation of the ML pipeline, the whole development process of control models can be accelerated. Necessary training data is transferred from vehicle to a cloud backend, where it gets preprocessed. In two case studies suitable architectures of artificial neural networks are analyzed and evaluated. Additionally, a method to assess the plausibility of ML model calculations is developed. This method is based on utilized training data and is linking valid output data domains with *Backwards Reachable Sets*. For Lipschitz-continuous *Feedforward Neural Networks* with *Rectified Linear Units* a backwards reachable plausibility assessment can be performed. In this way model calculations can be classified as plausible or implausible.
In use, the ML models provide a better model quality as the up to now used physically motivated substitute models. This leads to a more precise calculation of the expected energy input into the cooling circuit for the target use case on prediction data. The utilization of the standard vehicle communication unit and of a cloud-based training is enabling an automated adaption of the control models run on the vehicle. As a result, the development process can be shifted to the cloud. By using scalar model quality criteria and a standardized approach, ML models can be provided with higher model quality in shorter time.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Symbolverzeichnis

Lateinische Formelzeichen

Symbol	Beschreibung	Einheit
a_{St}	Streckensteigung	%
a_{HRC}	Prädizierte Streckensteigung aus HRC	%
b_{St}	Prädizierte Bebauung an der Fahrstrecke	
$\boldsymbol{b}$	Biasvektor	
C_{m}^{c}	Geometrischer Schwerpunkt von $\mathcal{D}_{\mathrm{bw,sub}}^{c}$	
$\mathcal{D}_{\mathrm{aus}}$	Ausgangsdatenraum	
$\mathcal{D}_{\mathrm{aus,sub}}$	Unterdatenraum in $\mathcal{D}_{\mathrm{aus}}$	
$\mathcal{D}_{\mathrm{bw,sub}}$	*Backwards reachable set* in $\mathcal{D}_{\mathrm{ein}}$	
$\mathcal{D}_{\mathrm{ein}}$	Eingangsdatenraum	
G_{Getr}	Getriebegang	
$\mathbf{h}^{(r)}$	Ausgangsvektor der verdeckten Schicht r	
i	Zählvariable (Modelleingänge)	
j	Anzahl der Eingangsgrößen	
k	Anzahl der Ausgangs- bzw. Zielgrößen	
k_{St}	Straßenklasse	
l	Zählvariable (Modellausgänge)	
m_{Fzg}	Fahrzeugmasse	kg
M_{mot}	Drehmoment des Verbrennungsmotors	Nm
n_{e}	Anzahl Epochen	
n_{ers}	Größe des *sliding window* zur Ersatzwertbildung	
n_{g}	Anzahl der Trainingsläufe pro Hyperparameterset	
n_{h}	Anzahl verdeckter Schichten	
n_{mb}	Datenpunkte pro *mini-batch*	
n_{n}	Anzahl künstlicher Neuronen pro verdeckter Schicht	
N_{mot}	Drehzahl des Verbrennungsmotors	1/min
$\overline{o}_{\mathrm{EG}}$	durchschnittliche Mehrfachabdeckung eines Datenpunktes	
Q_{in}	Gesamter Energieeintrag in den KKL über die Zeit	J
$\dot{Q}_{\mathrm{in}}$	Gesamter Wärmestrom in den KKL	W
$\dot{Q}_{\mathrm{mot}}$	Wärmestrom in Motorblock als Motorabwärme	W
$\dot{Q}_{\mathrm{Oel}}$	Wärmestrom des Getriebeölkühlers in KKL	W
$\dot{Q}_{\mathrm{Reib}}$	Wärmestrom in KKL (Reibleistung Kolben im Motorblock)	W
$\dot{Q}_{\mathrm{Zyl}}$	Wärmestrom aus Zylinderkopf in den KKL	W

r_{St}	Kurvenradius	
R^2	Bestimmtheitsmaß	
$\mathbb{S}_{\text{ges}}$	Gesamtdatensatz	
$\mathbb{S}_{\text{test}}$	Testdatensatz	
$\mathbb{S}_{\text{train}}$	Trainingsdatensatz	
$\mathbb{S}_{\text{vali}}$	Validierungsdatensatz	
$T_{\text{mot,aus}}$	Temperatur Kühlflüssigkeit (Motorausgang vor Thermostat)	°C
v_{Fzg}	Fahrzeuggeschwindigkeit	km/h
v_{lim}	Geschwindigkeitslimit aus HRC	km/h
$w_{n_n,j}$	Einträge der Gewichtsmatrix (Netzwerkgewichte)	
$\mathbf{W}$	Gewichtsmatrix	
$\boldsymbol{x}$	Eingangsvektor (Modelltraining)	
$\boldsymbol{x}_{\text{HRC}}$	Vorausschauvektor aus HRC	
$\boldsymbol{x}_{\text{ign}}$	Durch Entscheidungsgrenze ignorierte Eingangsvektoren	
$\mathcal{X}$	Originaldatenraum	
$\boldsymbol{y}$	Zielvektor (Modelltraining)	
$\hat{\boldsymbol{y}}$	Modellberechnung	
$\mathcal{Z}$	Suchraum	

Griechische Formelzeichen

Symbol	Bedeutung
γ	Parameter steuert Straffheit der Hyperebene der OCSVM
ϵ_0	Initiale Lernrate
η_{Antr}	Wirkungsgrad Antriebsstrang
λ	Hyperparameter für Regularisierungsterm
μ	Erwartungswert
ν	Parameter definiert Anteil der Stützvektoren für OCSVM
ξ	Schlupfvariable
ρ	Offset
σ	Standardabweichung
ϕ	Abbildungsfunktion
Φ	Aktivierungsfunktion eines künstlichen Neurons
ω	Vektor zur Beschreibung der Hyperebene einer OCSVM

Abkürzungen

ADAS	*Advanced Driver Assistance System*
ADASIS	*Advanced Driver Assistance System Interface Specification*
API	*Application Programming Interface*
BEV	*Battery Electric Vehicle*
BPTT	*Backpropagation Through Time*
BRS	*Backwards Reachable Set*
CAN	*Controller Area Network*
CAN-FD	*Controller Area Network with Flexible Data-Rate*
CI/CD	*Continuous Integration / Continuous Deployment*
CNN	*Convolutional Neural Network*
CPC	*Central Powertrain Controller*
CPU	*Central Processing Unit*
CRISP-DM	*Cross Industry Standard Process for Data Mining*
CRISP-ML(Q)	*Cross Industry Standard Process for Machine Learning with Quality Assurance Methodology*
DoE	*Design of Experiments*
EA	evolutionärer Algorithmus
EG	Entscheidungsgrenze
FNN	*Feedforward Neural Network*
GAN	*Generative Adversarial Network*
GPU	*Graphics Processor Unit*
HPS	Hyperparameterset
HRC	*Horizon Reconstructor*
HT	Hochtemperatur
HV	Hochvolt
KI	Künstliche Intelligenz
KKL	Kühlkreislauf
KMU	kleinere und mittlere Unternehmen
KNN	künstliches neuronales Netzwerk
LSTM	*Long Short-Term Memory*

ML	*Machine Learning*
MLP	*Multilayer Perceptron*
MPP	*Most Probable Path*
MSE	*Mean Squared Error*
NARX	*Nonlinear Autoregressive Network with Exogenous Inputs*
OCSVM	*One-Class Support Vector Machine*
PE	physikalisch motiviertes Ersatzmodell
PHEV	*Plug-in Hybrid Electric Vehicle*
PKW	Personenkraftwagen
PSO	Partikelschwarmoptimierung
ReLU	*Rectified Linear Unit*
RMSE	*Root Mean Squared Error*
RNN	*Recurrent Neural Network*
RPA	rückverknüpfte Plausibilitätsanalyse
SGD	*Stochastic Gradient Descent*
SNR	*Signal-to-Noise Ratio*
SSE	*Sum of Squared Errors*
SST	*Total Sum of Squares*
SVM	*Support Vector Machine*
TCU	*Telemetry Control Unit*
TM	Thermomanagement

1 Einleitung

Die Durchsetzung klimaschonender Gesetzesrichtlinien und strengerer Abgasnormen hat die Automobilbranche zu Beginn des letzten Jahrzehnts dazu gezwungen, den Fokus in der Antriebsstrangentwicklung zu ändern. Leistungssteigerungen durch die kontinuierliche Weiterentwicklung des Verbrennungsmotors gehören der Vergangenheit an. Steigende Energiepreise, gesetzlich festgelegte Abgasnormen und innerstädtische Fahrverbote erzwingen ein Umdenken. Die Einhaltung des Flottengrenzwertes für den CO_2-Ausstoß ist ohne batterieelektrische Fahrzeuge für die Hersteller nicht möglich [1]. Ein konventionelles Fahrzeug mit Verbrennungsmotor gibt ca. 30 % der im Kraftstoff enthaltenen inneren Energie in Form von Wärme über das Abgas an die Umwelt ab [2]. Die Nutzung dieser Abwärme kann durch ein effizientes Thermomanagement (TM) optimiert und für die Innenraumklimatisierung verwendet werden. Im Vergleich dazu entsteht bei batterieelektrischen Fahrzeugen (engl. *Battery Electric Vehicle*s (BEVs)) kaum überschüssige Abwärme. Die daher zusätzlich benötigte Heizleistung entzieht der Traktionsbatterie wertvolle Energie. Ein optimal an das Fahrzeug angepasstes, lernendes TM kann dazu beitragen die Reichweite von Plug-In-Hybriden (engl. *Plug-in Hybrid Electric Vehicles* (PHEVs)) und BEVs zu maximieren sowie deren Energieverbrauch und damit verbundenen Schadstoffausstoß zu reduzieren.

Um energieeffiziente und innovative Lösungen zu entwickeln, spielt neben der Antriebstechnologie ein ganzheitlicher Ansatz eine entscheidende Rolle. Dazu fokussieren die Hersteller auf Softwarequalität und Skalierbarkeit. Das Lernen aus Fahrzeugdaten ist ein wichtiger Pfeiler, um eine automatisierte Entwicklung in schnellen Zyklen durchführen zu können. Dazu ist Expertise und Fachwissen zum Thema Künstliche Intelligenz (KI) und speziell im Themenbereich des ML ausschlaggebend.
Laut einer vom Bundesministerium für Wirtschaft und Klimaschutz (BMWK) (ehem. (BMWi)) beauftragten Studie aus dem Dezember 2020 können Unternehmen durch den Einsatz von KI bis zu 25 % mehr Gewinne erwirtschaften. Der Großteil der Unternehmen fällt unter die Kategorie kleinere und mittlere Unternehmen (KMU) [3]. Weitere Forschungen aus dem Jahr 2022 unterstreichen den Umstand, dass der Einsatz von KI die Innovationskraft und Produktivität von Unternehmen steigert [4]. Es zeigt sich, dass der langfristige Einsatz dieser Methoden einen stetig wachsenden Einfluss auf die Produktivität der Unternehmen erzeugt.
Für die Automobilhersteller bieten KI-Methoden einen Baustein, um ihre Produkte in den engen Schranken der Umweltstandards innovativ und attraktiv zu gestalten. Die Umstellung von Verbrennungsmotoren auf elektrifizierte Hybride und vollelektrische

Antriebe erzeugt tiefgreifende Veränderungen in der Branche. In den letzten Jahrzehnten wurde systematisch an einer Verringerung des Kraftstoffverbrauchs sowie des Schadstoffausstoßes der bewährten Diesel- und Ottotechnologie geforscht. Aufgrund strengerer gesetzlicher Vorgaben und Strategiewechsel richtet sich die Branche derzeit fast ausschließlich auf vollelektrische Fahrzeuge aus. Seit 2020 gilt ein verschärfter Flottengrenzwert für den CO_2-Ausstoß von durchschnittlich $95\frac{g}{km}$ für neu zugelassene Personenkraftwagen (PKW). Bis zum Jahre 2025 bzw. 2030 soll dieser Grenzwert weiter reduziert werden. Ziel ist eine CO_2-Minderung von 15% bzw. 37,5% gegenüber dem Grenzwert von 2020. Dies sieht die EU-Verordnung zur Verminderung der CO_2-Emissionen von Personenkraftwagen [1] vor. Die *Mercedes-Benz Group AG* erwartet für das Jahr 2030 einen zulässigen CO_2-Ausstoß von 65 g/km pro Fahrzeug. Dies käme einem Verbrauch von 2,4 L Diesel bzw. von 2,7 L Benzin pro 100 km Fahrtstrecke gleich [5]. Unter anderem aus diesem Grund änderte die *Mercedes-Benz Group AG* ihre Strategie in den letzten Jahren von *electric first* zu *electric only* [6], [7]. Um die Emissionsvorgaben zu erreichen, wird auf eine Elektrifizierung des Antriebsstrangs hingearbeitet. Die Strategie sieht vor, bis 2030 möglichst in allen Märkten ausschließlich BEVs anzubieten [7]. Automobilhersteller weltweit verfolgen ähnliche Pläne.
Das Ziel der Hersteller ist eine sukzessive Elektrifizierung ihrer Fahrzeugflotte. Dabei stehen im Moment sowohl PHEVs als auch BEVs zum Verkauf. Die *Mercedes-Benz Group AG* hat im Jahr 2021 knapp über 227.000 PHEVs und BEVs verkauft. Das entspricht 9,5 % aller in 2021 von ihr verkauften Fahrzeuge. Im Moment überwiegt der Anteil an PHEVs [6], [8]. Insgesamt wurden knapp 50.000 BEVs verkauft. Im Vergleich zum Vorjahr entspricht das einer Steigerung um fast 155 %. Bei mehr als 2.093.000 verkauften PKW weltweit entspricht das nur ca. 2 % aller Fahrzeuge. Die Tendenz ist jedoch steigend und bietet ein großes Potenzial. Dazu passt die Ankündigung, dass ab 2025 nur noch E-Fahrzeug-Plattformen neu in den Markt eingeführt werden sollen.

Steigende Rechenleistung an Bord der Fahrzeuge sowie eine Anbindung an ein leistungsstarkes Cloud-Backend ermöglichen die Datenverarbeitung, Auswertung und Modellbildung auf skalierbaren Cloud-Ressourcen. Dies ist essentiell für eine datengetriebene Entwicklung und ermöglicht ein standardisiertes und automatisiertes Vorgehen. Die Optimierung von Prozessen und Arbeitsschritten bietet einen Mehrwert in der Entwicklung der Fahrzeuge. Die spezifische Anpassung von Fahrzeugen an Flotten oder Kundenverhalten zielt auf eine weitere individualisierte Optimierung des Energieverbrauchs ab.
Dieser Umbruch ist enorm und steht erst am Anfang. Zu Beginn wird das Vorhaben durch teilweise elektrifizierte Antriebsstränge, der sogenannten Hybridisierung, unterstützt [9]. Im Zuge dessen müssen nicht nur Betriebsstrategien und Regelkonzepte angepasst werden, sondern auch die Art und Weise, wie diese im Entwicklungszyklus des Fahrzeugs erstellt und kalibriert werden. Diese Transformation erzeugt Innovationsbedarf an unterschiedlichen Stellen. Die klassische Antriebsstrangentwicklung muss drängende Fragen nach Technologie und Fertigung der Batteriespeicher und Elek-

tromotoren beantworten. Gleichzeitig gewinnt die Fahrzeugsoftware und der damit verbundene Kundennutzen an Einfluss. Software- und Hardwareentwicklungszyklen sollen entkoppelt werden, um Innovationen häufiger und flexibler auf den Markt bringen zu können.
Dies bedeutet für das Entwicklungsumfeld eine schnellere Taktzeit bei gleichzeitig steigender Variantenvielfalt. In Bezug auf die Entwicklung von voll und teilweise elektrifizierten Fahrzeugantriebssträngen müssen Methoden entwickelt werden, die diese Transformation ermöglichen und beschleunigen. Ein essentieller Bestandteil ist die Automatisierung der Entwicklung und Optimierung von Softwarefunktionen mittels Nutzung von Fahrzeugdaten und ML. Ein standardisiertes Vorgehen ermöglicht die Bewertung und Auswahl der Lösungen für verschiedenste Anwendungsfälle nach klaren skalaren Gütekriterien.
Der steigende Anteil an Softwarekomponenten und die steigende Zahl verbauter mechatronischer Komponenten erhöhen die technologische Komplexität. Damit steigt der Bedarf zur Absicherung all dieser Funktionen in Bezug auf die funktionale Sicherheit. Laut der *U.S. National Highway Traffic Safety Administration (NHTSA)* wurden im Jahre 2020 über 31 Millionen Fahrzeuge aufgrund von Sicherheitsmängeln zurückgerufen. Die funktionale Sicherheit der Antriebsstrangsoftware wird nach der *ISO 26262* zertifiziert [10]. Aktuell existieren noch keine Vorgaben für die gesonderte Absicherung von datenbasierten Softwarefunktionen. Im Forschungsprojekt *KI Absicherung* untersucht ein Verbund aus Firmen und Universitäten die Möglichkeiten zur Absicherung von KI im Fahrzeug [11].

1.1 Motivation der Arbeit

Die Motivation dieser Arbeit ist die Entwicklung einer automatisierten und datenbasierten Modellierung unter Verwendung von Fahrzeugmessdaten. Dabei werden die vorhandenen Kommunikationsschnittstellen zwischen Fahrzeug und Cloud-Backend genutzt, um eine ML-Pipeline auf skalierbaren Cloud-Ressourcen auszuführen. Dadurch sollen ML-Modelle mit vergleichbarer oder besserer Modellgüte in einem automatisierten Arbeitsablauf erstellt und an das Fahrzeug gesendet werden. Um die Emissionsziele zu erfüllen, muss jede Komponente in einem PKW auf einen energieeffizienten Betrieb ausgelegt werden. Jede elektrisch oder mechanisch angetriebene Komponente trägt zum Energieverbrauch bei. In Hybrid-Fahrzeugen sind das beispielsweise elektrische Abgasturbolader, elektrische Startergeneratoren, elektrische Kältemittelverdichter und Chiller zur Batteriekonditionierung. Der energieeffiziente Einsatz mittels einer optimalen Regelung kann direkt zur Reduzierung des Energiebedarfs beitragen. Dabei spielt die softwaretechnische Ansteuerung und Regelung dieser Komponenten eine entscheidende Rolle.
Das TM zur energieeffizienten Kühlung der Komponenten des Antriebsstrangs übernimmt dabei die anspruchsvolle Kühlung der Lithium-Ionen-Batterie und weiterer

Leistungselektronik. Dafür werden die bestehenden Niedertemperaturkühlkreisläufe mit dem Kältemittelkreislauf gekoppelt. Die Wechselwirkungen zwischen Innenraumklimatisierung und TM nehmen zu. Für diese neuen komplexen Kühlkreisläufe (KKL) muss eine effiziente Regelung erstellt werden. Die Anzahl der Applikationsparameter steigt.
Zusätzlich generieren die verschiedenen Antriebstechnologien, Baureihen, Leistungsklassen und Ausstattungsoptionen der Fahrzeuge eine große Variantenvielfalt. Die Regelalgorithmen und Softwarefunktionen müssen an jede dieser Varianten optimal angepasst werden können, um die Potenziale für einen energieeffizienten Betrieb voll auszuschöpfen.
Für BEVs verlagert sich die Komplexität. Der Verbrennungsmotor entfällt. Die Abstimmung von Antrieb, Batteriekonditionierung und Innenraumklimatisierung erfordert eine komplexe Auslegung, um den Wirkungsgradvorteil eines BEV in Reichweite und Komfort umsetzen zu können [12]. Die Innenraumkühlung hat direkten Einfluss auf die Reichweite des Fahrzeugs [2], [13]. Während Fahrzeuge mit Verbrennungsmotor die Energie zur Innenraumklimatisierung teilweise aus der Abwärme des Antriebsstrangs gewinnen können und damit eine Steigerung der Effizienz ermöglichen, kommen in BEVs verschiedene Zuheizsysteme zum Einsatz, die den Energiebedarf steigern [14].
Das PKW-TM wird derzeit durch eine Vielzahl von Softwarefunktionen geregelt. Die zugehörige Software wird auf dem Antriebsstrangsteuergerät (engl. *Central Powertrain Controller* (CPC)) ausgeführt. Allein die Software der in dieser Arbeit verwendeten Versuchsfahrzeuge verfügt über mehr als 5.000 Applikationsparameter zu Kalibrierung der verschiedenen Regelalgorithmen im TM. Derzeit basiert die Regelstrategie im PKW-TM der ausgelieferten Serienfahrzeuge auf Kennfeldern und Kennlinien. Diese kennfeldbasierte Strategie regelt die Ansteuerung der Aktuatoren der verschiedenen KKL anhand von hinterlegten Kennlinien und Kennfeldern [15]. Wechselwirkungen einzelner Regelgrößen nehmen durch eine steigende Anzahl an Komponenten in mehreren KKL zu.
In [16] wurde eine modellbasierte Regelung anhand von physikalisch motivierten Ersatzmodellen (PE-Modellen) eingeführt. Der modellbasierte Ansatz ermöglicht es, die Informationen über die wahrscheinlichste Fahrstrecke zu nutzen, um den Energieeintrag in den KKL im Voraus zu berechnen. Dies ermöglicht eine optimierte Ansteuerung der elektrischen Aktuatoren des KKL in Abhängigkeit der prädizierten Fahrtstrecke. Der modellbasierte Ansatz führt zu einer effizienteren Ansteuerung der Aktuatoren und zu einer Verringerung der CO_2-Emissionen [16].
Ähnlich zur kennfeldbasierten Regelung müssen die in [16] eingeführten PE-Modelle anhand einer Vielzahl von Applikationsparametern manuell kalibriert werden. Dies stellt eine Hürde im Hinblick auf die große Variantenvielfalt und die allgemein verkürzten Entwicklungszyklen dar. Gleichzeitig eröffnet eine modellbasierte Regelung viele Möglichkeiten, die verfügbaren Datenschnittstellen, Cloudressourcen und *state-of-the-art* Modellierungsmethoden in die Entwicklung von Regelungsmodellen und Softwarefunktionen zu integrieren.

Um das PKW-TM künftig mit datenbasierten Modellen ausrüsten zu können, werden in dieser Arbeit Forschungsergebnisse präsentiert, die es ermöglichen, eine rückverknüpfte Plausibilitätsanalyse (RPA) der Modellberechnungen durchzuführen. Die entwickelten Methoden sollen einen Mehrwert in der Funktionsentwicklung liefern, indem sie den manuellen und damit verknüpften zeitlichen Aufwand reduzieren und gleichzeitig eine optimale Modellgüte liefern. Weiterer Vorteil ist die flexible Anpassung der datenbasierten Modelle an neue Datengrundlagen resultierend aus Hard- oder Softwareänderungen des Fahrzeugs. Die entwickelte ML-Pipeline liefert für jede Datenaktualisierung als Ergebnis das Modelle mit der besten Modellgüte. Durch die Automatisierung des Vorgehens können Modelle anhand von belastbaren Daten und Bewertungsmetriken erstellt werden. Im Rahmen dieser Arbeit wird der Schwerpunkt auf die automatisierte und abgesicherte Modellerstellung gelegt. Informationen über die vorausliegende Fahrstrecke werden vom Navigationssystem in Form eines Fahrzeughorizonts bereitgestellt. Der sogenannte *Horizon Reconstructor* (HRC) bereitet die Informationen über die vorausliegende Fahrstrecken, den *Most Probable Path* (MPP), auf. Die Daten entsprechen den standardisierten Protokollen der *Advanced Driver Assistance System Interface Specification* (ADASIS). Sie werden in dieser Arbeit zusätzlich aufgezeichnet und zur Überprüfung der entwickelten ML-Modelle genutzt.

1.2 Einordnung und wissenschaftliche Zielstellung

Das TM stellt aufgrund seiner vielen Wechselwirkungen mit andern Systemen (bspw. mit der Innenraumklimatisierung oder dem Batteriemanagement) ein komplexes System dar und wird anhand einer Vielzahl von Applikationsparametern kalibriert. Um für dieses System optimale Parameterkombinationen zu ermitteln, ist eine automatisierte und datenbasierter Auslegung und Optimierung erforderlich. Auf diesem Weg kann anhand objektiver Gütekriterien die höchst mögliche Modellgüte erreicht werden [14]. Da die zunehmende Elektrifizierung des Fahrzeugantriebsstrangs immer weniger Abwärme produziert, benötigen die davon abhängigen Systeme wie die Innenraumklimatisierung oder die Batterietemperaturregelung eine effizientere und genauere Regelung, um keine unnötig hohen Energieverluste zu erzeugen [13]. Dafür ist ein prädiktives TM als wichtiger Baustein des energieeffizienten Betriebs eines PHEV oder BEV einzustufen.

1.2.1 Einordnung

Schon 2014 wurde in [17] dargelegt, dass ein prädiktives TM im PKW sinnvoll ist, um eine Reduzierung des Energiebedarfs zu erzielen.
In [18], [19] wird eine modellprädiktive Regelung vorgeschlagen. Die Autoren setzen auf physikalische Ersatzmodelle zur optimalen Regelung der thermischen Größen. In

[18] werden zwei verschiedene Simulationshorizonte für den modellprädiktiven Regler eines PHEV verwendet, um die unterschiedliche Dynamik von Antrieb und thermischen Lasten abzubilden. Dabei wird der Gesamtkraftstoffverbrauch des Fahrzeugs optimiert. Die Methodik aus [19] zielt auf die Reduktion des Energiebedarfs für die Kühlung der Fahrgastzelle eines BEV ab und bedient sich ebenfalls eines modellprädiktiven Reglers. Auch hier kommt eine modellbasierte Regelung im TM zum Einsatz. Den beiden Arbeiten ist gemein, dass sie auf manuell kalibrierten Modellen basieren, welche nicht ohne Expertenwissen an neue Fahrzeugeigenschaften angepasst werden können.
Ein anderer Ansatz zur prädiktiven Regelung wird in [16] verfolgt. Informationen über die wahrscheinlichste Fahrstrecke (MPP) liefert das Navigationssystem in Form von *Advanced Driver Assistance System* (ADAS)-Horizontdaten. Diese werden mit Hilfe von PE-Modellen des Antriebsstrangs inklusive KKL verarbeitet, um die voraussichtlich ins Kühlsystem eingetragene Wärme $\dot{Q}_{\text{in}}$ zu prädizieren. Grundlegender Unterschied zu den Ansätzen mit einer modellprädiktiven Reglung ist die Verwendung von Kartendaten für die wahrscheinlichste Fahrstrecke. Der MPP ist ohne aktivierte Zielführung im Navigationsgerät verfügbar. Alle diskutierten Arbeiten basieren auf manuell zu kalibrierenden Ersatzmodellen, die jeweils an eine Fahrzeugaufbau- und Fahrzeugantriebsvariante angepasst worden sind. Eine automatisierte Modellierung unterschiedlicher Modellarchitekturen ist nicht möglich. Das in [16] vorgestellte, sogenannte prädiktive TM bietet mit den verwendeten PE-Modellen den Aufsatzpunkt für die vorliegende Arbeit. Die Substitution der PE-Modelle durch ML-Modelle ist ein Ziel dieser Arbeit.
Um den manuellen Aufwand zur Kalibrierung der Modellparameter zu verringern, wird in [20] eine Mehrzieloptimierung durchgeführt. Dafür werden evolutionäre Algorithmen (EA) verwendet, die eine Paretofront mit möglichen optimalen Lösungen aufspannen. Mit Hilfe weiterer Randbedingungen und Expertenwissen konnte die für die Anwendung am besten geeignete Lösung identifiziert werden.
In einer Vorstudie zu dieser Arbeit wurden die Applikationsparameter ausgewählter Modelle mittels EA in einem automatisierten Workflow optimiert [21]. In [22] finden EA Anwendung in der Kalibrierung von Modellen in der Motorapplikation. Die Partikelschwarmoptimierung (PSO) als weiterer Vertreter der multikriteriellen Optimierung wurde zur Lösung nicht-konvexer Optimierungsaufgaben in [23] für die Entwicklung von Softwarefunktionen für den Antriebsstrang verwendet. Die verwendeten Optimierungsverfahren sind gut erforscht und gängige Praxis. Sie finden daher breite Anwendung als Werkzeug zur Parameteridentifikation [24].
Die Verfahren eignen sich dagegen nicht, um den Aufbau eines Modells an veränderliche Anwendungsfälle anzupassen. Die vorab definierten physikalischen Zusammenhänge limitieren den Einsatzzweck und die Flexibilität im Modellaufbau. Das Wissen erfahrener Entwickler_innen ist erforderlich, um die initialen Modelle zu erzeugen und die Startparameterkonfiguration vorzugeben. Die Fähigkeit des Modells zu Abbildung der physikalischen Zusammenhänge ist durch den Modellaufbau und die hinterlegten mathematischen Zusammenhänge vorab definiert.
Diesen Nachteil überwinden die Verfahren des ML. Aus der zugrundeliegenden Daten-

basis wird im Modelltraining eine mathematische Abbildung der Eingangsdaten auf die Ausgangsdaten erlernt. Diese datenbasierte Methodik ist einfach zu automatisieren. Die Modellgröße und -komplexität lassen sich steuern, um die geforderte Modellgüte zu erhalten [24]–[26].
In [27] werden unterschiedliche ML-Methoden eingesetzt. Dabei werden verschiedene Modelle verknüpft, um eine bessere Generalisierung auf den Zieldaten zu erreichen. Die entwickelten Modelle sind für stationäre Anwendungen wie die Regelung einer Gebäudeheizung vorgesehen. In [28] werden künstliche neuronale Netzwerke (KNN) verwendet, um komplexe Fluidprozesse zu modellieren. Die entwickelten Modelle werden zur Regelung von Fluidtemperaturen herangezogen.
Die Verwendung von ML-Modellen im prädiktiven PKW-TM ist nicht dokumentiert. In einer begleitenden Studie zu dieser Arbeit wurde die Modellierung der Motorleistung durch KNN anhand von Fahrzeugmessdaten untersucht [29]. Die Vorteile in Bezug auf kleine Modellgrößen und eine geringe Rechenkomplexität wurden deutlich. Die erreichte Genauigkeit konnte mit den im prädiktiven TM eingesetzten PE-Modellen gleichgesetzt werden. Die Modellbildung konnte automatisiert durchgeführt werden und zeichnete sich durch einfache Skalierbarkeit der Modelle aus. Die Interpretation der Modelle im Hinblick auf die abgebildeten physikalischen Zusammenhänge ist allerdings schwer bzw. oft nicht möglich.
In diesem Zusammenhang zielt die „KI-Strategie der Bundesregierung“ darauf ab, den Einsatz einer sicheren und vertrauenswürdigen KI zu fördern [30]. Zu diesem Zweck sollen unter anderem die Investitionen des Bundes in KI in den kommenden Jahren von drei auf fünf Milliarden Euro erhöht werden. Auf der anderen Seite verweist [31] auf Umfrageergebnisse, in denen führende Technologiekonzerne angeben, dass 75 % bis 85 % ihrer ML-Projekte nicht zu einem zufriedenstellenden Abschluss kommen. Gründe hierfür sind unter anderem die Qualität der Daten und der Software.
Das prädiktive TM eignet sich gut zur Untersuchung des Potenzials das ML-Modelle zur Substitution der derzeit verwendeten PE-Modelle bieten. In der Literatur werden lernende Algorithmen in Kombination mit einer vorausschauenden Temperaturregelung der Komponenten des Antriebsstrangs nicht eingesetzt. Für BEVs gibt es Ansätze zur vorausschauenden Batterietemperaturmodellierung anhand von *Recurrent Neural Network* (RNN) [32]. Allerdings betrifft das nur eine Modellierung der inneren Batterietemperatur. Der hier vorgestellte Ansatz umfasst eine universell einsetzbare Methodik, die sich damit befasst, die bisher verwendeten PE-Modelle aus dem vorausschauenden Regelkreis durch ML-Modelle zu ersetzen.
In dieser Arbeit wird eine Möglichkeit untersucht, wie Berechnungen von ML-Modellen plausibilisiert werden können.

1.2.2 Forschungsfragen

Aus der Motivation und der Einordnung der Themen in den aktuellen Stand der Technik ergibt sich der Forschungsbedarf dieser Arbeit. Das Ziel ist die Entwicklung einer

Methodik zur datenbasierten Modellerstellung für die modellbasierte Regelung im PKW-TM. Dadurch soll der manuelle Aufwand zur Kalibrierung der Applikationsparameter reduziert werden. Die kontinuierliche Übertragung von Fahrzeugdaten über die Kommunikationsschnittstelle des Fahrzeugs an ein Cloud-Backend soll eine durchgängig automatisierte Modellbildung in Form einer ML-Pipeline mit Trainingsdaten versorgen. Die Arbeit wird am Beispiel der modellbasierten Regelung im prädiktiven Thermomanagement entwickelt. Die bisher im prädiktiven TM verwendeten PE-Modelle sollen dafür durch datenbasierte Modelle ersetzt werden. Es ergeben sich drei Forschungsfragen:

- Kann der manuelle Kalibrierungsaufwand für Regelungsmodelle durch den Einsatz von ML-Modellen reduziert werden?
- Kann mit dem Einsatz von ML-Modellen eine vergleichbare Modellqualität erzielt werden?
- Können die Modellberechnungen der ML-Modelle mit geringem Rechenaufwand plausibilisiert werden?

Zur Beantwortung dieser Forschungsfragen werden geeignete ML-Modelle untersucht und ausgewählt. Durch die Entwicklung einer ML-Pipeline und Implementierung auf einem Cloud-Backend soll die Modellerstellung automatisiert und der manuelle Kalibirierungsaufwand reduziert werden. Die Modellqualität ist von der Datenqualität und der zulässigen Modellkomplexität abhängig. Eine standardisierte und automatisierte Datenvorverarbeitung soll eine bestmögliche Datengrundlage liefern. Skalierbare Rechenleistung und eine Kommunikationsschnittstelle zum Fahrzeug erzeugen einen kontinuierlich anwachsenden Trainingsdatensatz. Um die Modellberechnung während der Anwendung auf dem Fahrzeug plausibilisieren zu können, soll eine Methodik entwickelt werden, die es ermöglicht, nicht-konvexe Datenräume automatisch zu beschreiben und einzugrenzen. Die entwickelte Methodik soll auf begrenzten Steuergeräteressourcen effizient ausgeführt werden können, um jedes Modellergebnis auf seine Plausibilität zu prüfen.
Die Integration von ML-Methoden in das prädiktive TM stellt eine Neuheit gegenüber dem Stand der Technik dar. Die Kombination aus modellbasierter Regelung, dem Einsatz von automatisiert trainierten, datenbasierten Modellen und einer Plausibilisierung der Modellberechnungen im prädiktiven TM ist eine Neuerung. Durch den Einsatz des serienmäßig verfügbaren Kommunikationsmoduls (engl. *Telemetry Control Unit* (TCU)) moderner Fahrzeuge sowie eines skalierbaren Cloud-Backends werden die Vorteile einer cloudbasierten Datenbasis und eines Trainings auf performanten und skalierbaren Rechenknoten genutzt. Dadurch entfaltet sich das volle Potenzial der Methodik, da schnelle Zyklen in der Entwicklung und Anpassung der Softwarefunktionen realisiert werden können. Die Modellierungsaufgabe umfasst die Abbildung eines mehrdimensionalen Eingangsvektors $\boldsymbol{x}$ auf einen mehrdimensionalen Zielvektor $\boldsymbol{y}$. Die benötigten Modelle müssen in der Lage sein, eine große Variantenvielfalt abzudecken und sollen möglichst automatisiert anhand von Messdaten aus Versuchsfahrzeugen

erstellt werden können. Für den Einsatz auf Steuergeräten mit begrenzten Ressourcen liefert die Methodik eine optimierte Modellgröße und -güte. Darüber hinaus wird im Rahmen dieser Arbeit die Plausibilisierung der Modellausgänge in der Anwendung untersucht.
Soweit die aufgeführten Forschungsfragen beantwortet werden können, ergibt sich ein Mehrwert für den gesamten Entwicklungszyklus der Softwarefunktionen. Ausgehend von einem reduzierten manuellen Modellierungs- und Kalibrierungsaufwand, kann durch eine standardisierte Berechnung der optimalen Modellgüte eine automatisierte datenbasierte Modellentwicklung auf dem Cloud-Backend erfolgen. Zudem ergibt sich die Möglichkeit, Modelle basierend auf einer neuen Datengrundlage an veränderliche Umgebungsbedingungen anzupassen. Die Einführung der RPA ermöglicht die Plausibilisierung der datenbasierten Modelle im Fahrzeug. Die effiziente Anwendung auf dem Steuergerät eröffnet zudem weitere Möglichkeiten zur Absicherung von Softwarefunktionen. Der skalierbare Aufbau und die durchgängig autonome Modellierung anhand von Trainingsdaten ermöglichen die Übertragbarkeit der Methodik auf andere Problemstellungen.

1.2.3 Fallstudien zur Bewertung der Ergebnisse in der Anwendung

Die Methoden des maschinellen Lernens bieten eine Vielzahl von Modellarchitekturen mit unterschiedlicher Modellkomplexität. In Kapitel 4.1 werden verschiedene Architekturen untersucht und bewertet. Auf Basis dieser Bewertung werden für die Anwendung im prädiktiven TM KNN ausgewählt. Da die möglichen Netzwerkarchitekturen große Unterschiede in Modellgüte und -komplexität mit sich bringen, werden zwei Fallstudien mit in Frage kommenden Architekturen durchgeführt.

Fallstudie I: Um den geringen Speicherplatz und die begrenzte Rechenleistung des Steuergeräts auszunutzen, wird ein einfaches, mehrschichtiges *Feedforward Neural Network* (FNN) untersucht. Der Aufbau ohne Rückkopplungen und Faltungen hält die Komplexität gering und bietet gleichzeitig ein hohes Potenzial für eine ausreichende Modellgüte. Dies ist auf die Tatsache zurückzuführen, dass es sich (aufgrund mehrerer verdeckter Schichten und nichtlinearer Aktivierungsfunktionen) bereits um einen universellen Approximator handelt [24]. Zudem sind FNN mit Aktivierungsfunktionen in Form von *Rectified Linear Units* (ReLUs) nachweislich Lipschitz-stetig und können so mittels der in Kapitel 5 entwickelten RPA während der Inferenz überprüft werden.

Fallstudie II: Um zeitabhängige Zusammenhänge besser durch ein KNN abbilden zu können, werden RNN verwendet. Diese verfügen über eine Rückführung der Ergebnisse vorangegangener Zeitschritte. Um die Modellkomplexität und die Parameterzahl zu beschränken, werden vergleichsweise einfache, sogenannte *vanilla* RNN untersucht.

Komplexere *Long Short-Term Memory* (LSTM)-Netzwerke wurden in Vorstudien betrachtet und bringen für die geforderte Modellgüte keinen nennenswerten Vorteil im Vergleich zur stark steigenden Parameterzahl und Komplexität mit sich. Da ein RNN in der Zeitebene „ausgerollt“ werden kann, kann es in vielerlei Hinsicht wie ein FNN behandelt werden. Es wird zusätzlich untersucht, ob die RPA aus Kapitel 5 ebenfalls einen Mehrwert in der Anwendung auf ein RNN liefern kann.

In den *Fallstudien I & II* wird dieselbe Datengrundlage verwendet. Es wird jeweils eine Gitterstudie durchgeführt, um das bestgeeignete Modell zu ermitteln. Dafür werden dieselben Bewertungskriterien verwendet (vgl. Kap. 4.2.1).

1.3 Aufbau der Arbeit

Um die beschriebenen Forschungsfragen zu lösen, beinhaltet **Kapitel 2** eine Erläuterung der nötigen mathematischen Grundlagen. Dem folgt eine Analyse des Stands der Technik zum Zeitpunkt der Ausfertigung. Kernpunkte sind das prädiktive TM sowie die Methoden des maschinellen Lernens. Zur Einordnung werden das konventionelle und das modellbasierte, prädiktive TM beschrieben. Letzteres basiert auf Vorausschaudaten aus dem Navigationssystem und einer Modellierung der Komponenten des Antriebsstrangs. Dem folgt ein Überblick über die Methoden des ML und die Möglichkeiten zur Integration von ML-Modellen in das prädiktive TM. Das Kapitel schließt mit der Beschreibung des Gesamtsystems.

Die Datenvorverarbeitung und das *feature engineering* sind Bestandteile von **Kapitel 3**. Die Entwicklung einer Methodik zur automatisierten Definition von Trainingsdatenräumen wird beschrieben und bewertet. Der Fokus liegt auf der Definition von Trainings- und Testdatenräumen für ein Training robuster ML-Algorithmen und die Gewährleistung der Abdeckung sämtlicher kritischer Betriebspunkte. Zur Beschreibung der Datenraumgrenzen werden *One-Class Support Vector Machines* (OCSVMs) verwendet, die einem Teilgebiet des maschinellen Lernens zuzuordnen sind. Die vorgestellte Methodik zur Unterteilung und Eingrenzung der Datenräume ist essentieller Bestandteil der entwickelten RPA (vgl. Kap. 5).

Kapitel 4 befasst sich mit der Untersuchung geeigneter ML-Algorithmen zur Substitution der PE-Modelle. Hierfür werden zuerst der passende Einsatzbereich sowie der individuelle Nutzen verschiedener ML-Algorithmen dargestellt. Daraufhin werden verschiedene KNN-Architekturen eingeführt und Anwendungskriterien für deren Auswahl definiert. Die Definition geeigneter Bewertungskriterien zum Vergleich der einzelnen Modelle und zur objektiven Bewertung der Modellgüte schließt sich an. Mit diesen Kennzahlen erfolgen der Vergleich und die Auswahl geeigneter Netzarchitekturen.
Die Modelle werden in zwei verschiedenen *Fallstudien* trainiert und evaluiert. Diese

werden in Kapitel 4, basierend auf den Erkenntnissen der Untersuchung zu den verschiedenen Netzarchitekturen, eingeführt.

In **Kapitel 5** wird die RPA entwickelt. Diese dient zur Plausibilisierung der Berechnungen der ML-Modelle. Dabei werden bereits vorhandene Methoden für datenbasierte Modelle erörtert und weiterentwickelt. Die Aufbereitung der verfügbaren Datenbasis ist dafür essentiell. Die Einfassung des Datenraums in eine Entscheidungsgrenze (EG) sowie die Unterteilung in Unterdatenräume ermöglichen eine Bewertung der Modellausgänge in Abhängigkeit der zugehörigen Eingänge. Um diesen Zusammenhang abzusichern, werden Modelle mit Lipschitz-stetigem Verhalten aufgebaut. So kann eine Methodik zur Plausibilisierung der Modellberechnungen erstellt werden. Diese stellt geringe Anforderungen an Rechenzeit und Ressourcen, um online auf dem Fahrzeug eingesetzt werden zu können.

Die Validierung und Bewertung der entwickelten Methodik in der Anwendung wird in **Kapitel 6** durchgeführt. Die entwickelte ML-Pipeline zur Automatisierung von Datenverarbeitung und Training wird zu Beginn bewertet.
Die gesamte Methodik wird prototypisch in Fahrzeug und Cloud umgesetzt, um den Prozess der Aktualisierung der Netze und deren Anwendung auf einem Fahrzeugsteuergerät darzustellen. Die entwickelte Methodik wird in Bezug auf den Einsatz im PKW-TM validiert und der Einfluss auf die Modellbildung und Modellgüte beschrieben. Es folgt ein Vergleich zum Stand der Technik und die Beschreibung der sich bietenden Potenziale.

Kapitel 7 schließt mit einer Zusammenfassung der Arbeit und gibt einen Ausblick auf potenziell weitere Einsatzbereiche der Methodik und sich anschließende Forschungsthemen.

2 Grundlagen und Stand der Technik

Die Substitution der gegebenen PE-Modelle des prädiktiven TM durch ML-Modelle basiert auf dem aktuellen Stand der Technik. Um optimal geeignete Methoden für die Entwicklung von ML-Modellen für modellbasierte Regelalgorithmen auszuwählen, werden zunächst die relevanten Grundlagen beschrieben. Darauf folgt eine Einordnung in den Stand der Technik. Dabei liegt der Schwerpunkt auf dem Einsatz von ML im PKW-TM. Abschließend wird das Gesamtsystem aus Fahrzeug, Software, Kommunikationsschnittstellen und Cloud-Backend beschrieben. Damit werden die Systemgrenzen für die vorliegende Arbeit definiert.

2.1 Mathematische Grundlagen der Modellbildung

Zur Erstellung modellbasierter Regelalgorithmen können verschiedene mathematische Methoden zur Modellierung verwendet werden. Die Modelltypen werden in Tabelle 2.1 in drei Kategorien eingeteilt.

2.1.1 Modellierungsmethoden

Physikalische Modelle beschreiben die zu modellierenden Zusammenhänge basierend auf bekannten physikalischen Gesetzmäßigkeiten. Für die Entwicklung zuverlässiger Modelle ist ein vollständiges Verständnis der herrschenden Zusammenhänge erforderlich. Dadurch entstehen meist komplexe Modelle, die auf umfassendem Fachwissen und Expertise beruhen (sogenannte *white-box* Modelle). Diese hängen nicht von Messdaten ab. Die Modellparameter können den physikalischen Größen des Anwendungsfalls direkt zugeordnet werden [24, S. 15]. Der Einsatz ist auf Anwendungsfälle begrenzt, in denen die physikalischen Zusammenhänge vollständig bekannt sind und mathematisch beschrieben werden können. Diese Art von Modellen weist gute Extrapolationseigenschaften auf und ist sehr zuverlässig. Die erreichbare Genauigkeit wird vom Verständnis des zu modellierenden Prozesses begrenzt [24]. Die resultierende Modellkomplexität und -größe erfordern in der Regel einen hohen Rechenaufwand.
Um die Modellgröße und den Rechenaufwand zu reduzieren, werden vereinfachte physikalische Ersatzmodelle eingesetzt. Diese sogenannten *grey-box* Modelle verwenden empirisch ermittelte Faktoren und Annahmen sowie physikalische Zusammenhänge als Grundlage. Modellparameter können nicht mehr ohne weiteres physikalischen Größen zugeordnet werden. Die Kalibrierung dieser Parameter erfolgt mittels Messungen des zu modellierenden Systems.

Tabelle 2.1: Unterschiedliche Methoden der Modellbildung nach [24]

	Modelltyp		
	white-box	*grey-box*	*black-box*
Informationsquelle	physikalische Gesetzmäßigkeiten, Erkenntnisse, Domänenwissen	Domänenwissen & Messdaten	Messdaten
Vorteile	gute Extrapolationsfähigkeit, gute Erklärbarkeit, hohe Zuverlässigkeit	hybride Struktur ermöglicht hohen Detailgrad u. Berücksichtigung komplexer Prozesse	skalierbar, kann unbekannte Prozesse abbilden, Inferenz mit geringer Rechenzeit
Nachteile	zeitintensive Modellierung, detailliertes Domänenwissen erforderlich	modellbasierte Regelung, reduzierte Ersatzmodelle	nicht physikalisch nachvollziehbar, limitiert durch verfügbare Daten, keine verlässliche Extrapolation
Anwendung	offline Simulation mit hohem Detaillierungsgrad (z.B. Strömungssimulation)	Modelle mit reduzierter Größe auf embedded Hardware; hybride Modelle	komplexe Prozesse, kleine Modelle, lernende Modelle

Im Vergleich dazu basiert die Modellierung von sogenannten *black-box* Modellen auf Messdaten des zu modellierenden Zusammenhangs. ML-Methoden erstellen dabei mathematische Abbildungen eines Eingangsraums auf einem Ausgangsraum. Die verfügbaren Daten und deren statistische Verteilung limitieren die erreichbare Modellgüte des Modells [24], [25], [33]. Die Modellparameter können nicht physikalisch interpretiert werden. Ein ML-Modell lernt immer einen impliziten Zusammenhang zwischen Ein- und Ausgangsgrößen des Modells.

2.1.2 Statistische Methoden und Wahrscheinlichkeit

In diesem Abschnitt werden die statistischen Methoden und Grundlagen der Wahrscheinlichkeitstheorie beschrieben, die für die Analyse der Trainingsdaten und die Auswahl geeigneter Datenräume verwendet werden. Um datenbasierte Modelle im industrienahen Einsatz produktiv einzusetzen, müssen reale Messdaten aufgezeichnet und zur Modellbildung verwendet werden. Dazu ist eine umfangreiche Datenvorverarbeitung notwendig (vgl. Kap. 3). Ein wesentlicher Bestandteil davon ist die Analyse der Datenverteilung und die Bestimmung der statistischen Kenngrößen. Diese dienen der Beschreibung des Trainingsdatenraums $\mathcal{D}_{\text{ges}}$. Sie können genutzt werden, um den gültigen Einsatzbereich des erstellten Modells zu definieren.
Der Begriff der Wahrscheinlichkeit ist eine Erweiterung der Logik, um mit Unsicherheit umzugehen [34]. Er bietet formale Regeln, um die Wahrscheinlichkeit für das Eintreten einer Annahme zu ermitteln.

Zufallsgrößen

Eine Zufallsgröße X beschreibt alle Werte x_i aus einem reellen Datenraum R die X unter Zufallsbedingungen annehmen kann [35]. Es wird zwischen diskreten und stetigen Zufallsgrößen unterschieden. Setzt sich R aus endlich vielen Werten zusammen, liegt eine diskrete Zufallsgröße X vor. Diese wird durch diskrete Werte $[x_1, x_2, \ldots, x_j]$ beschrieben, deren Wahrscheinlichkeit $p_i = P(X = x_i)$ ist [35], [36]. Eine Zufallsgröße muss mit einer zugehörigen Verteilungsfunktion betrachtet werden, um die Wahrscheinlichkeit der möglichen Werte x_i bestimmen zu können [25].

Erwartungswert μ

Der Erwartungswert $\mu = E(X)$ beschreibt den Wert einer Zufallsgröße X, den diese im Mittel annehmen wird. Daher wird μ auch als Mittelwert bezeichnet. Für eine diskrete Zufallsgröße X berechnet sich der Erwartungswert aus:

$$\mu = E(X) = \sum_i x_i p_i. \tag{2.1}$$

Für eine stetige Zufallsgröße X berechnet sich der Erwartungswert folgendermaßen:

$$\mu = E(X) = \int_{-\infty}^{\infty} x f(x) dx. \tag{2.2}$$

Varianz und Standardabweichung

Die Varianz σ^2 einer Grundgesamtheit N wird mittels

$$\sigma^2 = \frac{\sum_{i=1}^{N}(x_i - \mu)^2}{N} \tag{2.3}$$

berechnet [33], [35]. Die Standardabweichung σ der Grundgesamtheit N ist daher

$$\sigma = \sqrt{\frac{\sum_{i=1}^{N}(x_i - \mu)^2}{N}}. \tag{2.4}$$

Hierbei ist μ das arithmetisches Mittel der Grundgesamtheit N. Eine endliche Menge an n Messwerten $[x_1, x_2, \ldots, x_n]$ wird als Stichprobe bezeichnet. Beispielsweise die Datengrundlage zum Training von datenbasierten Modellen. Die Varianz s^2 einer Stichprobe ist durch

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{x})^2 \tag{2.5}$$

beschrieben [33], [35]. Der Mittelwert $\overline{x}$ der Stichprobe ist das arithmetische Mittel

$$\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i. \tag{2.6}$$

Analog zur Gleichung 2.5 ist die Standardabweichung s der Stichprobe (auch Streuung genannt) beschrieben durch

$$s = \sqrt{s^2} = \sqrt{s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{x})^2}. \tag{2.7}$$

Mit diesen statistischen Maßzahlen (Erwartungswert und Varianz) sowie der zugehörigen Verteilungsfunktion lässt sich eine Beschreibung der Datenverteilung (vgl. Kap. 3) und der Initialisierung der Modellparameter (vgl. Kap. 4) durchführen.

Verteilungsfunktionen

Die Verteilungsfunktion gibt die Wahrscheinlichkeit einer Zufallsgröße X für jeden möglichen Wert x_i an, den diese Zufallsgröße annehmen kann. Im Allgemeinen kann in den Naturwissenschaften und in der Technik davon ausgegangen werden, dass bei zufällig aufgezeichneten Messdaten einer stetigen Zufallsgröße und einer genügend hohen Anzahl an Datensätzen eine Gaußverteilung vorliegt [24], [25], [33]. Die verfügbaren Datenpunkte stellen eine Stichprobe n dar. Steigt die Größe der Stichprobe, so nähert sich ihre Verteilung der Wahrscheinlichkeitsverteilung der Grundgesamt-

heit an. Die statistischen Momente geben in diesem Zusammenhang Aufschluss über die Verteilung der gesammelten Stichprobe. Gilt die Voraussetzung einer zufälligen Stichprobenentnahme, so können die statistischen Momente zur Ermittlung der zugehörigen Grundgesamtheit verwendet werden [33]. Diese statischen Momente sind der Erwartungswerte μ und die Varianz σ^2. Die Verteilungsfunktion der Zufallsgröße X wird durch

$$F(x) = P(X \leq x) \quad \text{für} \quad -\infty \leq x \leq \infty \tag{2.8}$$

beschrieben. Für eine stetige Zufallsvariable X wird die Wahrscheinlichkeit das X in einem endlichen Intervall liegt, durch die Wahrscheinlichkeitsdichte $f(t)$ beschrieben [35].
Die stetige Zufallsgröße X mit einer Verteilungsfunktion

$$P(X \leq x) = F(X) = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{(t-\mu)^2}{2\sigma^2}} \, dt \tag{2.9}$$

wird als normalverteilt bezeichnet. Daraus ergibt sich die Wahrscheinlichkeitsdichte der Normalverteilung

$$f(t) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(t-\mu)^2}{2\sigma^2}} \tag{2.10}$$

mit dem Maximum für $t = \mu$ [33], [35]. Eine normalverteilte Zufallsgröße X wird durch

$$X \sim \mathcal{N}(\mu; \sigma^2) \tag{2.11}$$

beschrieben. Betrachtet man eine uniform verteilte stetige Zufallsgröße X, so liegt eine Gleichverteilung mit einer Dichtefunktion $f(x)$ der Form

$$f(x) = \begin{cases} \frac{1}{b-a} & \text{für} \quad a \leq x \leq b, \\ 0 & \text{sonst}, \end{cases} \tag{2.12}$$

vor. Diese gilt für das endliche Intervall $[a, b]$. Eine uniform verteilte Zufallsgröße X ist durch

$$X \sim \mathcal{U}(a; b) \tag{2.13}$$

gekennzeichnet.

2.1.3 Optimierungsverfahren

Die Optimierung der Modellparameter von Regelungsmodellen für den Einsatz im PKW-TM ist ein entscheidender Prozess, um deren Modellgüte zu maximieren. Da im Allgemeinen abstrakte physikalisch motivierte Ersatzmodelle (*grey-box*) und in dieser Arbeit im Speziellen datenbasierte ML-Modelle (*black-box*) verwendet werden, muss die Modellparameteroptimierung anhand von Messdaten vorgenommen werden. Dabei

müssen nichtlineare Zusammenhänge und Lösungsräume berücksichtigt werden, um der Komplexität der zu modellierenden Zusammenhänge Rechnung zu tragen. Für die Optimierung der Netzwerkgewichte von KNN hat sich das Gradientenabstiegsverfahren etabliert. Dafür werden die vorhandenen Trainingsdaten in Eingangsdaten (engl. *features*) und Zieldaten (engl. *targets*) unterteilt. Daraufhin wird das Modell in Form seiner Beschreibung als mathematische Funktion auf die *features* angewendet. Die errechneten Ausgänge werden mit den *targets* verglichen und ergeben den aktuellen Modellfehler. Die Berechnung des Gradienten der Fehlerfunktion liefert eine Steigung. Deren Kehrwert zeigt in die Richtung des steilsten Abstiegs der Fehlerfunktion und weist auf ein Minimum. Dabei kann es sich sowohl um ein globales als auch um ein lokales Minimum handeln [24], [25]. Die Anwendung des Modells auf alle Trainingsdaten wird als Epoche bezeichnet. In der Regel werden mehrere Epochen durchlaufen, um die gewünschte Modellgüte zu erhalten. Folgende Gradientenabstiegsverfahrens sind zu unterscheiden:

- **Klassischer Gradientenabstieg:**
  Die mathematische Funktion des Modells wird einmal auf alle vorhandenen Trainingsdaten angewendet. Die berechneten Fehler der Ausgänge stellen die Fehlerfunktion dar. Der errechnete Gradient für jeden einzelnen Trainingspunkt wird verwendet, um den mittleren Gradienten aller Trainingsdatenpunkte und ihrer Fehler zu errechnen. Der mittlere Gradient wird verwendet, um die Parameter entgegen der Steigung in Richtung des erwarteten Minimum dieser Fehlerfunktion anzupassen. Das Ziel ist es, den Modellfehler durch diese Anpassungen zu reduzieren. Der Rechenaufwand für die Gradientenberechnung ist verhältnismäßig groß, da alle Trainingspunkte (und zugehörigen Fehler) gemeinsam berücksichtigt werden. Da die Modellparameter des Modells erst am Ende einer Epoche erneuert werden, ergibt sich eine langsamere Konvergenz als im Vergleich zu anderen Verfahren. Von Vorteil ist dagegen, dass durch die Berücksichtigung aller Datenpunkte eine gute Generalisierbarkeit erreicht werden kann. Die Zielfunktion oszilliert im Allgemeinen nicht so stark wie bei Verfahren mit kleineren Datenmengen pro Update (siehe nachstehende Verfahren).
- **Stochastischer Gradientenabstieg**
  Das Verfahren (engl. *Stochastic Gradient Descent* (SGD)) verhält sich genau gegenteilig zum klassischen Gradientenabstiegsverfahren. Es wird nur ein zufällig ausgewählter Trainingspunkt berücksichtigt. Für diesen wird der Fehler im Vergleich zu den Trainingszielen berechnet. Für die entstandene Fehlerfunktion wird der Gradient errechnet, um wie zuvor beschrieben, eine Anpassung der Parameter in Richtung des Minimums vorzunehmen. Die Modellparameter werden einmal pro Datenpunkt erneuert. Daher werden innerhalb einer Epoche ebenso viele Updates durchgeführt wie Datenpunkte vorhanden sind. Eine Epoche ist beendet, wenn jeder Datenpunkt einmal verwendet wurde. Dies führt zu einer schnellen Erneuerung der Modellparameter, was mit einer geringen Rechenleistung pro

Trainingsschritt einhergeht. Die Methodik führt zu einer sehr hohen Varianz in der Zielfunktion. Je nach Lernrate kann die Zielfunktion um das tatsächliche Minimum oszillieren, ohne dieses jemals exakt zu erreichen.

- **Batchwise Gradient Descent:**
  Die Gradienten werden für eine Untermenge aller Trainingsdaten (engl. *batch*) berechnet. Das Verfahren wird daher auch *mini-batch*-Verfahren genannt. Die Gradienten eines *mini-batch* werden für die Anpassung der Modellparameter verwendet. Eine Epoche ist beendet, wenn alle Trainingsdaten einmal als *mini-batch* verwendet wurden. Ein Vorteil ist der geringere Rechenaufwand für die Berechnung der Gradienten, da nur eine kleine Untermenge aller Datenpunkte pro *mini-batch* verwendet wird. Dadurch kann eine schnellere Anpassung der Modellparameter vor dem Ende einer Epoche stattfinden. Die Zielfunktion oszilliert nicht so stark wie beim SGD. Das Verfahren liefert einen guten Kompromiss zwischen schnellerer Konvergenz und Rechenaufwand.

Der Vollständigkeit halber seinen ebenfalls gängige Optimierungsverfahren wie EA und PSO erwähnt. Diese bieten sich an, wenn ein nichtlinearer Lösungsraum vorliegt. Die optimale Parameterkonfiguration kann damit gut angenähert werden[23], [24].

2.2 Stand der Technik

2.2.1 Konventionelles PKW-Thermomanagement

In der Auslegung eines Kühlsystems für einen PKW liegt der Fokus darauf, die benötigte Kühlleistung mit effizienten, kostengünstigen und kompakten Komponenten zu realisieren. Viele Kriterien müssen berücksichtigt werden, um eine optimale Anordnung und Dimensionierung zu erreichen. Diese Kriterien beinhalten die Geometrie der Kühler, die Leistungsaufnahme der Lüfter und Aktuatoren sowie Einflüsse auf Fahrleistung, c_W-Wert und Crashverhalten [14]. Die Temperaturen der Komponenten des Kühlsystems werden durch das PKW-TM geregelt. Durch die verschiedenen KKL wird sichergestellt, dass die Komponenten des Antriebsstrangs in ihren optimalen Temperaturbereichen betreiben werden. Dadurch kann Reibung vermindert, Schadstoffemissionen minimiert und der generelle Energieverbrauch reduziert werden.
In dieser Arbeit bezeichnet der Begriff „konventionelles Thermomanagement“ das *state-of-the-art* Verfahren zur Regelung der Temperaturen in den einzelnen KKL des Antriebsstrangs. Dieses kommt bisher in auf dem Markt erhältlichen Serienfahrzeugen zum Einsatz. Hierbei werden die elektrischen Aktuatoren, die zur Temperaturregelung verwendet werden, anhand von Kennlinien und Kennfeldern betrieben. Für eine Eingangsgröße wird über eine Kennlinie die zugehörige Ausgangsgröße bestimmt. Gleiches gilt für mehrere Eingangsgrößen und ein zugehöriges Kennfeld. Auf diese Weise werden für verschiedene Betriebspunkte Vorgaben definiert, zwischen diesen Stützstellen wird bei Bedarf interpoliert [14]. Durch die Art und Weise des Vorgehens

werden Ansteuerungen und Kreislaufabschnitte oftmals isoliert betrachtet und nicht mit allen Rückkopplungen aus dem Gesamtsystem. Ein Grund dafür ist, das bei Steigender Zahl von berücksichtigen Abhängigkeiten die Kennfelddimensionen steigen. Im Prozess der manuellen Kalibrierung der Applikationsparameter, nach teilweise subjektiven Kriterien [37], sind hoch dimensionale Kennfelder nicht gut zu handhaben. Es ist davon auszugehen, dass die Kapazität der Hochvoltbatterien weiter steigen wird. Damit wird ein höherer Energiegehalt auf kleinerem Bauraum einhergehen [38]. Um die Lebensdauer weiterhin gewährleisten zu können und die Energieeffizient weiter zu steigern, kommt dem TM eine kritische Rolle zu. Die Regelung der Aktuatoren entscheidet über die effiziente Nutzung der verfügbaren Energie. Aktuatoren sind elektrische Kühlmittelpumpen, Drehschieberventile, Lüfter, ein elektrisches Thermostat sowie, bei Hybridfahrzeugen, ein sogenannter Chiller. Der Chiller ist ein Wärmeübertrager, der bei Bedarf mittels elektrischer Energie die Kühlflüssigkeit aus dem ersten Niedertemperaturkühlkreislauf aufheizen kann, um so die Batterietemperatur zu erhöhen. Ist die Batterie zu warm, kann der Chiller mit Hilfe des Kältemittels aus dem Kreislauf der Klimaanlage die Batterie kühlen. All diese Aktuatoren benötigen Energie und bieten so Potenziale zur Verbesserung des Energieverbrauchs, der Reichweite und der Schadstoffemissionen des Fahrzeugs [14].

2.2.2 Prädiktives PKW-Thermomanagement

Die wesentlichen Wärmeströme in die Kühlkreisläufe resultieren aus hohen den Leistungsanforderungen an den Fahrzeugantrieb. Zusätzlich können extreme Temperaturen während der Fahrt oder beim Laden der Hybrid- oder Traktionsbatterie das Kühlsystem stark beanspruchen. Zu jeder Zeit muss der Betrieb aller Komponenten in einem optimalen Temperaturfenster gewährleistet werden. Für diese und weitere Anwendungsfälle kann ein prädiktives TM durch die Kenntnis der vorausliegenden Fahrtstrecke die kritischen Komponenten rechtzeitig konditionieren oder entlasten [16]. Um ein prädiktives PKW-TM realisieren zu können, bedarf es zweier Grundvoraussetzungen. Zum einen sind Informationen über die vorausliegende Fahrtstrecke (MPP) erforderlich. Zum andern bedarf es eines Fahrzeugmodells, um die gegebenen Vorausschauinformationen zu verarbeiten. Anhand der Modelle lässt sich im Voraus berechnen, welche Antriebsleistung das Fahrzeugs für die anstehende Fahrtstrecke benötigt. Aus der Antriebsleistung kann auf den Wärmestrom $\dot{Q}_{\text{in}}$ der Antriebsstrangkomponenten in den KKL geschlossen werden. Die wichtigsten Komponenten sind dabei Hochvoltbatterie, Verbrennungsmotor, Chiller, Kühlmittelpumpen, Klimakompressor, Wärmetauscher und Öl-Kühler. Zusätzlich kann für einen bevorstehenden Ladestopp die für ein Schnellladen nötige Batterietemperatur eingeregelt werden.
Das prädiktive TM hat das Ziel, die voraussichtlich benötigte Leistung für die kommende Fahrtstrecke zu berechnen [16]. Dies wird anhand von Informationen über die wahrscheinlichste Fahrstrecke (MPP) des Fahrzeugs ermittelt. Dazu werden die Vorausschaudaten als Eingänge der PE-Modelle des Antriebsstrangs verwendet. Diese

berechnen die benötigte Leistung. Die Leistung in Form von Drehmoment und Drehzahl hat direkten Einfluss auf die Temperaturentwicklung des Motors und der Hochvoltbatterie. Das Kühlsystem muss diese und weitere Komponenten in ihrem optimalen Temperaturbereich halten und Wärme an die Kühlflüssigkeit und die Umgebungsluft abführen. Sind all diese Informationen vorhanden, kann vorausschauend gekühlt werden, um die Komponenten des KKL länger in einem optimalen Betriebspunkt zu halten. Diese Methodik verringert Reibungen und Kraftstoffverbrauch. Sie erhöht zudem die Lebensdauer der Komponenten, da die thermische Alterung verringert werden kann [16]. Die Modelle werden auf dem CPC in Echtzeit ausgeführt. Durch diese *embedded* Anwendung sind sie in Größe und Genauigkeit begrenzt. Für jede Fahrzeugvariante muss, abhängig von Motor, Batterie und Baureihe, manuell eine optimale Parametrierung dieser Modelle ermittelt werden. Das heißt, alle Modellparameter, die einen Einfluss auf Genauigkeit und Modellgüte haben, werden manuell anhand von Messungen aus dem Zielfahrzeug auf die tatsächlich auftretenden Temperaturen kalibriert.
In dieser Arbeit werden die verwendeten physikalischen Ersatzmodelle als PE-Modelle bezeichnet, um sie von den ML-Modellen zu unterscheiden.
Das bisher im prädiktiven TM verwendete PE-Modell des Verbrennungsmotors, besitzt inklusive der verwendeten Motorkennfelder und Kennlinien über 1000 Parameter.

2.2.3 Vorausschaudaten und Fahrzeughorizont

Die verwendeten Vorausschaudaten basieren auf dem ADAS-Horizont Konzept basierend auf der ADASIS. Der ADAS-Horizont stellt dabei digitale Karteninformationen für die voraussichtliche Fahrtstrecke zur Verfügung [39]. Der sogenannte MPP basiert auf der ADASIS und liefert Horizont-Daten zur wahrscheinlichsten Fahrstrecke des Fahrzeugs [39]. Dieser Horizont beinhaltet eine beschränkte Auswahl an Informationen. Dazu gehören beispielsweise die voraussichtliche Fahrtstrecke, Informationen über die aktuelle Straßenklasse k_{St} (Autobahn, Bundesstraße, Landstraße, ...), Bebauung b_{St} (städtischer Bereich oder außerorts), die Steigung der Fahrtstrecke a_{HRC} (engl. *slope*), Geschwindigkeitslimits auf der Fahrtstrecke (v_{lim}) und der Kurvenradien (r_{St}). Die ADAS-Daten werden vom ADAS-Provider des Navigationssystems in einer Ringspeicherlogik bereitgestellt und durch den HRC in positionsbasierte Arrays umgewandelt. Die Arrays beinhalten die verfügbaren ADAS-Daten für Geschwindigkeitslimits, Steigungen, Bebauung und Kurvenradien und ein zugehörige Positionsangabe in Metern. Die betrachteten Informationen über die vorausliegende Fahrtstrecke liegen anfänglich als Ereignisse in positionsbasierten Vektoren vor. Ein zusätzlicher Positionsvektor ermöglicht die Zuordnung der Ereignisse zu den prädizierten Streckenabschnitten vor dem Fahrzeug. Anhand der verfügbaren ADAS-Daten wird eine prädizierte Fahrzeuggeschwindigkeit für die einzelnen Streckenabschnitte ermittelt. Damit können die positionsbasierten Arrays in zeitbasierte Arrays umgerechnet werden. Dadurch entsteht ein zeitbasierter Vorausschauvektor für jedes Signal [16], [39]. Diese sind notwendig, um von den PE-Modellen des prädiktiven TM verarbeitet werden zu können.

2.2.4 Methoden des maschinellen Lernens

In dieser Arbeit werden Methoden des ML eingesetzt. Diese können nach geltender Definition dem Feld der KI zugeordnet werden. Eine genau Abgrenzung der KI ist derzeit nicht eindeutig möglich, da die KI eine Vielzahl an Werkzeugen und Methoden umfasst. Systeme, in denen diese Methoden zum Einsatz kommen, verwenden in der Regel noch weitere Werkzeuge und Verfahren [40]. Die Anlehnung der KNN an natürliche Neuronen und deren Verknüpfung im menschlichen Gehirn verheißt zwar große Ähnlichkeit, die KNN stellen in der realen Anwendung jedoch ein äußerst abstraktes mathematisches Ersatzmodell dar [25], [40]. In den neuronalen Knotenpunkten finden gewichtete mathematische Operationen statt, welche die natürlichen Vorgänge im menschlichen Hirn nur annähern können [24], [25].
Nach *German Standardization Roadmap on Artificial Intelligence* [41] wird die KI in verschiedene KI-Methoden unterteilt:

- Problemlösen, Suche, Optimierung, Planen und Entscheidungsfindung
- Wissensrepräsentation und Inferenz
- *Machine Learning*
- Hybride Lernverfahren

Die Unterteilung richtet sich nach Russell und Norvig [41] und gliedert die KI einerseits in Methoden und andererseits in Fähigkeiten. Dabei zeigen diese Methoden in der Reihenfolge ihrer Auflistung immer mehr Gemeinsamkeiten mit dem gemeinhin bekannten Verständnis der menschlichen Intelligenz. Der Begriff Inferenz (engl. *inference*) ist im englischen Sprachgebrauch eine weitverbreitete Bezeichnung für die Ausführung von trainierten ML-Modellen. Das Gebiet der hybriden Lernverfahren verwendet Methoden aus den anderen drei Gebieten und zielt auf eine ganzheitliche KI ab. Nach der Einteilung in verschiedene KI-Fähigkeiten fällt die Anwendung von KNN zur Substitution der PE-Modellen im prädiktiven TM unter die Kategorie *Gedächtnisse und Modelle* [41].
Die Fähigkeit, Zusammenhänge aus Daten zu abstrahieren, ist dem ML zuzuschreiben. ML-Modelle können auf Grundlage von Messdaten die zugrundeliegenden Beziehungen zwischen Eingangs- und Ausgangsdaten durch eine mathematische Funktion z.B. in Form eines KNN repräsentieren. Die Methoden des Maschinellen Lernens lassen sich nach [42] in drei Kategorien unterteilen:

- *Supervised Learning* eignet sich zur Klassifikation und Regression. Datensätze beinhalten Eingangsdaten und zugehörige Zielwerte. Die Methodik findet beispielsweise Anwendung in der Bilderkennung, Text- und Sprachverarbeitung, Zeitreihenprädiktion oder Modellierung komplexer Ein- und Ausgangsbeziehungen.
- *Unsupervised Learning* eignet sich zur Ermittlung von Mustern oder Clustern in Datensätzen ohne klare Kategorisierung. Diese Verfahren werden eingesetzt, um

Cluster in Datensätzen zu finden oder die Verteilung der Daten durch ein Modell möglich genau zu beschreiben.
- Beim *Reinforcement Learning* lernt das Modell aus der Interaktion mit seinem Umfeld. Ziel des Trainings ist die Maximierung einer Belohnung für richtige Interaktionen. Der Ansatz eignet sich bspw. zum Erlernen von Spielen oder regelbasierten Prozessen.

Für die vorliegende Arbeit werden Methoden des *supervised learnings* angewendet. Die Trainingsdaten bestehen aus Eingangsvektoren (engl. *features*) und zugehörigen Zieldatensätzen (engl. *targets*). Das heißt, der erwartete Modellausgang für jeden Trainingseingang ist bekannt. Das können kategorisierte Bilddateien oder Datensätze für Klassifikationsmodelle sein. Eine andere Möglichkeit zur Anwendung von *supervised learning* ist die Erstellung von Regressionsmodellen. Damit können z.B. die Ein- und Ausgangsbeziehungen von technisch-physikalischen Prozessen durch eine mathematische Funktion approximiert werden. Im *supervised learning* wird eine Zielfunktion (bzw. Kostenfunktion) minimiert, um den Fehler zwischen dem berechneten Modellausgang $\hat{\boldsymbol{y}}$ und *target* $\boldsymbol{y}$ zu minimieren. Dadurch wird das Modell mit der besten Anpassung an die Trainingsdaten ermittelt [24], [43].
ML-Methoden eignen sich zur nichtlinearen Systemidentifikation. Ihr Vorteil ist, dass trotz der individuellen Unterschiede der einzelnen Systeme viele mit ein und demselben Ansatz aus dem ML beschrieben werden können. Sie eignen sich gut für einen Einsatz in der Industrie, da eine weitgehend automatisierte und robuste Modellbildung realisiert werden kann [24], [44]. Von Polynommodellen über einfache FNN bis hin zu tiefen RNN gibt es eine Vielzahl verfügbarer Methoden, um nichtlineare Systeme abzubilden. KNN haben den Vorteil, dass sie auch stark nichtlineares Verhalten abbilden können [45]. Die steigende verfügbare Rechenleistung von Fahrzeugsteuergeräten führt dazu, dass auch komplexe KNN online ausgeführt werden können.
In der industriellen Anwendung muss dabei stets die Balance zwischen Rechenzeit, Ressourcenverbrauch und Modellgüte gefunden werden.

2.3 Integration von ML-Modellen in das prädiktive Thermomanagement

Der Aufbau des prädiktiven TM eignet sich gut zur Integration datenbasierter Modelle. Die Verwendung zur vorausschauenden Regelung entkoppelt die Modelle von der kritischen reaktiven konventionellen Regelung. In der derzeit verwendeten Logik ist eine Rückfallebene implementiert, welche die Möglichkeit ausbleibender ADAS-Horizontdaten jederzeit einkalkuliert. Durch die Kombination von ML-Modellen mit einer Methodik zu Plausibilisierung (vgl. Entwicklung der RPA in Kap. 5) können implausible Berechnungen erkannt und durch diese Rückfallebene abgefangen werden. In diesem Fall wird die prädiktive Regelung für den aktuellen Rechenschritt des Steuergerätetaktes zurückgesetzt. Im nächsten Rechentakt verarbeiten die Modelle

den erneuerten ADAS-Horizont.
Der MPP liefert Daten für Streckeninformationen über 8 km vor dem Fahrzeug. Ein Update des MPP findet mit einer Frequenz von 10 Hz im Rechentakt des Steuergeräts statt. Somit kann jederzeit mit einem aktuellen ADAS-Horizont gerechnet werden.
In der vorliegenden Arbeit wird die Substitution mehrerer PE-Modelle durch ML-Modelle untersucht. Dabei sollen die bereits beschriebenen Vorteile in der automatisierten Modellbildung mit einer höheren Modellgüte für das prädiktive TM kombiniert werden. Dadurch soll ein Mehrwert für die Entwicklung effizienter, prädiktiver modellbasierter Regelalgorithmen geschaffen werden. Eine derartige Verknüpfung aus Methoden des ML mit modellbasierten Regelalgorithmen und der Anwendung im PKW-TM ist so noch nicht durchgeführt worden. Zudem existiert derzeit kein effizienter Algorithmus zur Plausibilisierung der Modellberechnungen im Rechentakt des Steuergerätes. Hierfür wird in dieser Arbeit eine Lösung durch Verwendung von OCSVMs vorgestellt. Diese ermöglicht die Entwicklung der RPA für die Modellausgänge, basierend auf den zum Training verwendeten Fahrzeugmessdaten. Da auf dem Fahrzeugsteuergerät nur begrenzt Ressourcen zur Verfügung stehen, müssen die derzeit verwendeten PE-Modelle stark vereinfacht werden. Die Verwendung von ML-Modellen bietet die Möglichkeit, eine hohe Modellgüte mit relativ kleinen Modellstrukturen zu realisieren. Diese können automatisiert erstellt werden. Die Modellbildung mittels einer ML-Pipeline auf dem Cloud-Backend kann in kurzer Zeit eine Vielzahl von Modellvarianten generieren. Damit kann die beste Modellvariante für den Einsatz auf der Zielplattform ermittelt werden. Die Durchführung einer Gitterstudie oder komplexerer Optimierungsmethoden liefert systematisch die beste Modellvariante. Im Laufe der Fahrzeugentwicklung kommt es regelmäßig vor, dass andere Gewerke (wie zum Beispiel die Getriebeapplikation oder die Betriebsstrategie der Hochvoltbatterie) Änderungen in den Antriebsstrang einbringen. Auf diese Änderungen muss das Modellverhalten der Fahrzeugmodelle im prädiktiven TM angepasst werden. Für den Einsatz der PE-Modelle ist hierfür eine erneute, manuelle Kalibrierung der Applikationsparameter notwendig. Im Falle der ML-Modelle kann durch das Training mit aktuellen Fahrzeugmessdaten das neue Verhalten erlernt und in das Modell aufgenommen werden. Daher birgt der Einsatz von datenbasierten Modellen für das prädiktive TM während der Entwicklung des Fahrzeugs großes Potenzial.

2.4 Beschreibung des Gesamtsystems

Das in dieser Arbeit betrachte Gesamtsystem besteht aus einem Erprobungsfahrzeug, einer Kommunikationsschnittstelle und einem Cloud-Backend. Das Erprobungsfahrzeug ist ein Vorserienfahrzeug mit Verbrennungsmotor und Hochvolt (HV)-Batterie. Da dieser Plug-In-Hybrid sowohl über einen konventionellen Antriebsstrang mit Verbrennungsmotor als auch über einen elektrischen Antrieb verfügt, liegt ein komplexes

System vor welches sich gut dafür eignet den Mehrwert der entwickelten Methoden zu überprüfen.
Das Fahrzeug ist eine *Mercedes-Benz* S-Klasse der Baureihe 223 mit einem sechs Zylinder Ottomotor (*M*256 Modellpflege) sowie einer E-Maschine mit bis zu 110 kW zusätzlicher Antriebsleistung. Die integrierte Hochvoltbatterie ermöglicht rein elektrische Fahrten von bis zu 100 km.
Das Fahrzeug ist mit Messhardware ausgestattet und zeichnet Signale, die über den *Controller Area Network* (CAN)-Bus fließen, sowie interne Steuergerätsignale auf. Die für die Anwendung nötigen Messsignale werden nicht über zusätzliche Sensorik aufgezeichnet, sondern durch die serienmäßig verbauten Sensoren. Dadurch ist sichergestellt, dass die entwickelte Methodik im Serienfahrzeug angewendete werden kann. Mit der Messhardware wird ein initialer Datensatz für die Entwicklung der ML-Pipeline und ML-Modelle aufgezeichnet. Die Messdaten werden mit Hilfe des serienmäßig verfügbaren TCU an ein Cloud-Backend übermittelt. Dort wird im Rahmen dieser Arbeit die ML-Pipeline entwickelt und implementiert (vgl. Kap. 3). Dadurch kann die Gesamtmethodik im Cloud-Backend validiert werden. Das Cloud-Backend basiert auf einer *Microsoft-Azure*-Plattform. Dort werden *Microservices* zum Senden, Empfangen und Aufbereite der Messdaten implementiert. Die Messdaten werden in einem *Datalake* gesammelt und die im Rahmen dieser Arbeit entwickelte ML-Pipeline wird auf diesem Cloud-Backend implementiert.
Nach der Sammlung der Daten und der Ausführung der ML-Pipeline übermittelt das Cloud-Backend die trainierten Modelle mit fixierten Modellparametern zurück an das Zielfahrzeug. Die Verbindung mit der TCU wird über eine fest verbaute SIM-Karte realisiert. Die durchgängige Automatisierung der kompletten Prozesskette ist essentiell zur Schaffung eines Mehrwerts in der Entwicklung von Fahrzeugsoftwarekomponenten. Dadurch kann der manuelle Aufwand zur Konfiguration der Applikationsparameter der PE-Modelle reduziert werden.

3 Automatisierte Datenvorverarbeitung und Definition zulässiger Datenräume

Im vorliegenden Kapitel wird eine vollständige Automatisierung der Datenvorverarbeitung entwickelt. Zusätzlich wird eine Methodik erarbeitet, welche es ermöglicht den vorhandenen Trainingsdatenraum mit einer einhüllenden EG zu beschreiben. Diese Arbeitsabläufe bilden die Grundlage der ML-Pipeline die zur Erstellung und Auswahl der ML-Modelle verwendet wird. Die automatisierte Methodik wurde im Rahmen der Untersuchungen zum Patent angemeldet und offengelegt [102]. Sie definiert den Einsatz in Fahrzeug und Cloud-Backend sowie die zugehörige Kommunikation zum Datenaustausch.
Die systematische Entwicklung eines automatisierten Arbeitsablaufs für die Erstellung von ML-Algorithmen (vgl. Zielsetzung der Arbeit in Kap. 1.2) verlangt eine automatisierte Beschreibung von Datenräumen beliebiger Dimension und Größe. Die zur Entwicklung des ML-Modells verwendeten Eingangsdaten und Zielgrößen stellen eine mehrdimensionalen Punktewolke dar. Durch gegebene technische Bedingungen und physikalische Zusammenhänge kann diese Punktewolke eine konkave, teilweise lückenhafte Form aufweisen. Diese ist mathematisch nur schwer exakt und automatisiert zu beschreiben. Eine adäquate Beschreibung ändert sich je nach Dimension der Ein- und Ausgangsgrößen. Um solch variable Datenräume automatisiert in guter Näherung zu beschreiben, wird in diesem Kapitel eine Methodik vorgestellt. Diese ermöglicht eine automatisierte Erfassung der verwendeten Datenpunkte und erstellt eine einhüllende EG. Die entwickelte Methodik kann anschließend dazu verwendet werden neue Datenpunkte auf ihre Zugehörigkeit zum beschriebenen Datenraum hin zu überprüfen. Dies wird zur Plausibilisierung der Modellausgänge im Anwendungsfall verwendet (vgl. Kap. 5).

3.1 Methodik und Datenanalyse

Zur Erstellung der Datenbasis für das Training der ML-Modelle werden Messdaten aus einem Versuchsfahrzeug (vgl. Kap. 2.4) aufgezeichnet und aufbereitet. Die vorliegenden Daten werden in einen Trainingsdatensatz $\mathbb{S}_{\text{train}}$, einen Validierungsdatensatz $\mathbb{S}_{\text{vali}}$ und einen Testdatensatz $\mathbb{S}_{\text{test}}$ unterteilt. Für das Training unterscheidet man zwischen den Eingangsgrößen (engl. *features*) und den Zielgrößen (engl. *targets*) des ML-Modells. Diese werden durch die verwendeten Eingangssignale bzw. durch die zu bestimmenden Modellausgänge definiert. Der Vektor $\boldsymbol{x}$ stellt die Modelleingänge dar. Der Vektor $\boldsymbol{y}$

beschreibt die Zielgrößen. Es gilt:

$$\boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_j \end{pmatrix}, \qquad \boldsymbol{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_k \end{pmatrix}; \tag{3.1}$$

$$j \in (3, 4, \cdots, 10), \qquad k \in (1, 2, 3). \tag{3.2}$$

Hierbei ist die Anzahl j der Eingangsgrößen im Vektor $\boldsymbol{x}$ abhängig von der Anzahl verwendeter Signale. Es werden mindestens drei Signale verwendet wie unten beschreiben repräsentieren diese Fahrzeuggeschwindigkeit v_{Fzg}, Streckensteigung a_{St} und Fahrzeugmasse m_{Fzg} (vgl. Tab 3.2). Weitere Eingangssignale werden durch *feature engineering* (vgl. Kap. 3.2.3) erzeugt und durch eine Gitterstudie in Kap. 4 ausgewählt. Dabei wurden in dieser Arbeit ein Maximum von 10 Eingangsgrößen nicht überschritten, da die Redundanz der im *feature engineering* zusätzlich erzeugten Signale keinen weiteren Mehrwert liefert. Die Anzahl k der Zielgrößen des Vektors $\boldsymbol{y}$ wird im Zuge der Gitterstudien in jeder der beiden *Fallstudien I & II* variiert, um die optimale Modellstruktur zu ermitteln. Die Anzahl der Eingangs- und Zielgrößen definiert die Dimensionen der Ein- und Ausgangsdatenräume $\mathcal{D}_{\mathrm{ein}}$ und $\mathcal{D}_{\mathrm{aus}}$.

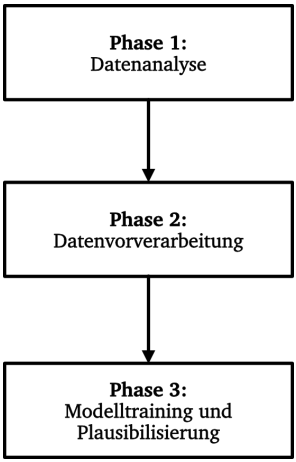


Abbildung 3.1: Darstellung des verwendeten Prozesses zur Datenanalyse und Aufbereitung. Angelehnt an CRISP-ML (Q) [31].

3.1.1 Methodisches Vorgehen

Das methodische Vorgehen orientiert sich an dem von Studer, Bui, Drescher et al. entwickelten Prozessmodell für maschinelles Lernen, dem *Cross Industry Standard Process for Machine Learning with Quality Assurance Methodology* (CRISP-ML(Q)) [31]. Dabei handelt es sich um eine Weitereinwicklung des von Chapman, Clinton, Kerber et al. eingeführten *Cross Industry Standard Process for Data Mining* (CRISP-DM) Prozessmodells welches in der Industrie seit Jahren zu Anwendung kommt [31], [46]. Dabei trägt das erweiterte CRISP-ML(Q) Modell dem Fakt Rechnung, dass im Zuge eines ML-Projekts anders als beim *data mining* nicht nur Muster und Informationen aus einer vorliegenden Datengrundlage ermittelt werden müssen, sondern datenbasierte Modelle erstellt werden. Diese sind in der Anwendung wechselnden Einflüssen ausgesetzt und müssen überprüft und ggf. weiterentwickelt werden können. Die erste und zweite Phase des CRISP-DM Prozessmodells wurden im CRISP-ML(Q) zur Phase 1 *Business and Data Understanding* zusammengefasst und eine Methodik zur Qualitätsabsicherung (Q) hinzugefügt. Für die vorliegende Arbeit wurden die in Abbildung 3.1 dargestellten Phasen aus dem CRISP-ML(Q) abgeleitet. Sie kommen, eingebettet in eine ML-Pipeline (vgl. Kap. 4.2), zur Anwendung. Folgenden Prozessschritte werden in den einzelnen Phasen durchgeführt:

Phase 1: Datenanalyse

1. Bewerten der Fahrzeugmessdaten
2. Auswahl geeigneter Fahrzeugsignale

Phase 2: Datenvorverarbeitung

1. Beschreibung der gegebenen Datenmenge mittels mathematischer Funktionen zu Begrenzung des Datenraums
2. Unterteilung in Trainings-, Test- und Validierungsadaten
3. Unterteilung in Unterdatenräume

Phase 3: Modelltraining und Plausibilisierung

1. Aufteilen in zulässige Eingangs- und Ausgangsdatenräume
2. Modellierung (Training der KNN)
3. Evaluation: Automatisierte Auswahl des besten Modells
4. Plausibility Assessment: Algorithmus zur Zuordnung neuer Datenpunkte zu den definierten (Unter-)Datenräumen (als Erweiterung von CRISP-ML(Q))

Die oben aufgeführten Punkte werden im Folgenden detailliert beschrieben. Die Datenanalyse, Auswahl und Vorverarbeitung der Messdaten haben direkten Einfluss auf die Entwicklung der ML-Modelle. Das sich daran anschließende Training der Modelle (vgl. Kap. 4) ist in der resultierenden Genauigkeit der erzeugten Modelle direkt abhängig von Menge, Verteilung und Qualität der Daten (vgl. Tab. 2.1 und Kap. 2.2.4).

Abschließend wird in Kapitel 3.3 eine Methodik zur Beschreibung der Datenräume und Bewertung neuer Datenpunkte eingeführt. Dies passt sich in die Phase 3 des CRISP-ML(Q) ein und erweitert diesen um ein weiteres Werkzeug zur Qualitätsbewertung der erstellten Modelle. Die mathematische Beschreibung des zulässigen Datenraums (engl. *domain of definiton*), ausgehend von den vorhandenen Trainingsdaten, wird verwendet, um die Modellberechnungen im Anwendungsfall zu plausibilisieren (vgl. Kap. 5 und Kap. 6).

Die Datenanalyse beinhaltet die Sammlung, Bewertung und Auswahl der Daten. Dieser

Algorithmus 1 Vorverarbeitung der Messdaten

Eingabe: Messdaten aus Erprobungsträger

1: Messdaten als **.mf4*-Datei laden
2: Parsen der ausgewählten Signale und laden in Pandas *DataFrame*
3: Überprüfen ob *NaNs* vorhanden, wenn ja Datenpunkt löschen
4: Filtern ausgewählter Signale um Rauschen zu verringern
5: Merkmalsgenerierung (engl. *feature engineering*)
6: Berechnen der statistischen Daten des *DataFrames*
7: Abspeichern des *DataFrames* im **.parquet*-Format
8: Laden der nächsten Messdatei

Ausgabe: Vorverarbeitete Dateien im **-parquet*-Format

Schritt lehnt sich an Phase 1 des CRISP-ML(Q) Modells an.
Die aufgezeichneten Fahrdaten aus dem Versuchsfahrzeug liegen im **.mf4*-Format vor. Daraus werden diejenigen Signale ausgewählt, welche für ein umfassendes Training und Validieren des ML-Modells nötig sind. Die Gesamtmenge aller verwendeten Datenpunkte spannt den Trainingsdatenraum $\mathcal{D}_{\text{ges}}$ auf. In der vorliegenden Arbeit ist ein Datenpunkt als Zeitpunkt einer Messung definiert. Alle zu diesem Zeitpunkt aufgezeichneten Signalgrößen bilden einen Vektor im Raum. Dieser wird anschließend in der Datenvorverarbeitung in Eingangsvektoren $\boldsymbol{x}$ und Zielvektoren $\boldsymbol{y}$ aufgeteilt.
Während der Datenvorverarbeitung (Phase 2) werden die relevanten Signale aufbereitet (vgl. Kap. 3.2). Dabei werden sie auf fehlende oder fehlerhafte Datenpunkte überprüft. Daran schließt sich eine Filterung und das sogenannte *feature engineering* an (vgl. Kap. 3.2.3). Da datenbasierte Modelle nur Zusammenhänge erlernen können, die in den Trainingsdaten repräsentiert werden (vgl. Kap. 2.2.4), kommt der Datenanalyse und -aufbereitung eine kritische Rolle im Prozess der Modellbildung zu. Fehlerhafte Aufzeichnungen und Rauschen können zu Fehlinterpretationen führen. Nach Methoden der statistischen Lerntheorie gilt die Annahme, dass die Aufzeichnung der Datengrundlage, der sogenannte Datengenerierungsprozess, einer Wahrscheinlichkeitsverteilung folgt. Somit gilt, dass die einzelnen Datenpunkte unabhängig voneinander sind und dass Trainings- und Testdatensatz der selben Wahrscheinlichkeitsverteilung p_{D} unterliegen [25]. Dabei ist nach [25] die Erfüllung zweier Anforderungen während des

Trainings entscheidend dafür, wie gut ein ML-Modell die Zielwerte der Trainings- und Testdaten berechnet:

- minimieren des Trainingsfehlers
- minimieren der Differenz zwischen Trainings- und Testfehler

Um diese Faktoren zu minimieren müssen sowohl **Überanpassung** (engl. *overfitting*) als auch **Unteranpassung** (enlg. *underfitting*) vermieden werden. Überanpassung wird durch eine ausreichend große Datengrundlage und passende Modellgröße vermieden. Unteranpassung wird durch ein ausreichend großes Modell vermieden, welches in der Lage ist, alle für den Anwendungsfall wichtigen Zusammenhänge aus den Trainingsdaten zu „lernen“ [24], [25]. Da die Modellgröße durch die Optimierung der Hyperparameter (engl. *hyperparameter tuning*) (vgl. Kap. 4.2.3) angepasst werden kann, spielt im ersten Schritt die Schaffung der Datengrundlage die wichtigste Rolle. Nach dem *No free Lunch Theoreme* [47] gilt, dass ein ML-Modell niemals die Gesamtheit aller Wahrscheinlichkeitsverteilungen mit ausreichender Genauigkeit erlernen kann, sich jedoch auf eine dem Anwendungsfall entsprechende Wahrscheinlichkeitsverteilung p_{Daten} trainieren lässt. Das *No Free Lunch Theorem* wurde von Wolpert und Macready in [47] beschrieben. Aus ihrer Untersuchung und den Bewertungen von Kochenderfer und Wheeler in [48] ergibt sich, dass die Kenntnis des vorliegenden Anwendungsfalls und der zugehörigen Wahrscheinlichkeitsverteilung des Datenraums unerlässlich für die Auswahl eines passenden ML-Modells sind. Daher ist ein erfolgreiches Training des ML-Modells direkt abhängig vom Domänenwissen, einer Analyse des Datenraums und der Wahl des Trainingsalgorithmus und der Zielfunktion (siehe *hyperparameter tuning* in Kap. 4.2.3).
Die Untersuchungen von Kochenfelder und Wheeler bestätigen die Wahl eines ML-Modells zur Erfüllung der vorliegenden Zielstellung, da sie in der Lage sind aus einem gut aufbereiteten Datensatz relevante Zusammenhänge abzuleiten. Dabei kann das Vorgehen automatisiert und die Bewertung verschiedener Modelle anhand skalarer Größen eindeutig geordnet werden. Dafür wird die verfügbare Datengrundlage mittels statistischer Analysen und dem Einsatz von OCSVM genau erfasst und beschrieben. Dieses Vorgehen wird im Folgenden ausgeführt.
Die Auswahl der Messgrößen (CAN-Bus Signale sowie interne Steuergerätesignale einer Messung aus dem Fahrzeug) definiert zu Beginn des Projekts die maximal mögliche Dimension und Größe des Trainingsdatenraums. Da die Speicherkapazität während der Fahrzeugmessungen begrenzt ist, gilt es die Auswahl der Messgrößen an den Anwendungsfall anzupassen. Eine größere Anzahl gemessener Signale erhöht die Dimension aber bietet eine umfassendere Information über die physikalischen und systemischen Zusammenhänge. Daher wird die Datengrundlage so breit wie möglich aufgezeichnet. Im vorliegenden Anwendungsfall ist jedoch ein Modell zu trainieren, dessen Modelleingänge x_i im Anwendungsfall ausschließlich aus ADAS-Vorausschaudaten aus dem Fahrzeughorizont bestehen oder daraus abgeleitet wurden. Das heißt dynamische, fahrerabhängige Eingaben können nicht als Eingangsgrößen x_i für das Training ver-

Tabelle 3.1: Übersicht verwendeter Signale aus dem Vorausschauhorizont (MPP).

Signalname	Formelzeichen	Beschreibung
Geschwindigkeitslimit	v_{lim}	diskrete Klassen in km/h
Streckensteigung	a_{HRC}	Steigung in %
Straßenklasse	k_{St}	Landstraße, Autobahn etc.
Bebauung	b_{St}	vorhanden ja/nein
Kurvenradius	r_{St}	Radius in m

wendet werden, da diese nicht als ADAS-Horizont vorliegen.
Um die Genauigkeit der Modellausgänge zu erhöhen, müssen auch die zugehörigen Trainingsdaten eine hohe Genauigkeit aufweisen. Dazu werden Fahrzeugmessdaten mit einer Abtastrate von 1 Hz verwendet. Höhere Abtastraten wurden ebenfalls untersucht, sind aber nicht zielführend für die betrachteten Zielgrößen, da der verwendete ADAS-Horizont nur sehr grob aufgelöst ist (vgl. Kap. 2.2.3). Hochfrequente Abtastraten der Messsignale erzeugen keinen nutzbaren Mehrwert für die Modellberechnungen im prädiktiven TM und erhöhen die Datenmenge unnötig. Die thermische Trägheit der Komponenten des Antriebsstrangs kann damit in ausreichender Genauigkeit modelliert werden.
Die im MPP verfügbaren und mittels des HRC aufbereiteten Daten aus dem ADAS-Horizont (vgl. Kap. 2.2.3) umfassen die für den MPP prädizierten Signale (vgl. Tab. 3.1). Aus dieser eingeschränkten Datenbasis werden die Eingangsgrößen x_i für das Modell gewählt. Das Ziel ist es mit den trainierten ML-Modelle den Wärmestrom $\dot{Q}_{\text{in}}$ in den Motorkühlkreislauf möglichst genau zu berechnen (vgl. Kap. 1.2 und Kap. 4). Der Wärmestrom $\dot{Q}_{\text{in}}$ (vgl. Gl. 4.26) ist für die vorausschauende Regelung und Ansteuerung der Aktuatoren im Hochtemperatur (HT)-Kühlkreislauf die entscheidende Basis. Aus $\dot{Q}_{\text{in}}$ werden Regelgrößen wie die Kühlmitteltemperatur $T_{\text{mot,aus}}$ am Motorausgang ermittelt.
Die Modelleingänge $\boldsymbol{x}$ sowie die Zielwerte $\boldsymbol{y}$ werden aus den verfügbaren Messdaten für das Training der ML-Modelle ausgewählt. Für lineare Modelle wird im Normalfall eine Korrelationsanalyse durchgeführt, um sicherzustellen, dass die Eingangsgrößen linear unabhängig sind [24]. Das Training von KNN mit nichtlinearen Aktivierungsfunktionen (vgl. Kap. 4.1.4) stellt eine nichtlineare Modellierung dar. Aus diesem Grund sind Korrelationsanalysen nicht ausreichend. Sie ermöglichen es linear abhängige Signale zu identifizieren und diese auszusortieren ohne Informationen für das Training zu verlieren. Damit kann zu Beginn die Dimensionalität des Eingangsdatenraums reduziert werden [24]. Zusätzlich eignet sich die Durchführung einer Sensitivitätsanalyse [103]. Wie von Nelles in [24] vorgeschlagen bietet die Option alle verfügbaren Eingangsgrößen zu verwenden eine Möglichkeit sofern die Anzahl der verfügbaren

Eingänge nicht zu groß ist. Da im vorliegenden Fall nur wenige Größen aus dem Vorausschauhorizont verwendet werden können, wird dieser Ansatz verfolgt. Zur zusätzlichen Bewertung der gewählten Größen wurden Kovarianz- und Autokorrelationsanalysen durchgeführt. Die Autokorrelationsanaylse eines Signals zeigt dabei wie stark abhängig es von vorangegangenen Ereignissen ist. Signale mit hoher Dynamik wie zum Beispiel das Motordrehmoment M_{mot} weisen dabei eine geringe Autokorrelation auf, da bereits geringe Veränderungen an der Fahrpedalstellung oder durch Gangwechsel abrupte Drehmomentabfälle erzeugen könne. Diese sind nicht aus der längeren Historie des Signalverlaufs abzuleiten und kündigen sich meist nur mit kurzem Vorlauf an. Im Vergleich dazu ist der Temperaturverlauf des HT-KKL stark abhängig von vorangegangenen Wärmeströmen und Fahrmanövern und weist somit eine hohe Autokorrelation auf.
Sind alle kritischen Wechselwirkungen und Abhängigkeiten ausgeschlossen kann mit einer reduzierten und aussagekräftigen Datengrundlage das Training der Modelle vorgenommen werden.

3.1.2 Datenerhebung und Bestimmung relevanter Messgrößen

Der Trainingsdatenraum für die Erstellung der ML-Modelle wird in dieser Arbeit durch Messungen aus einem Erprobungsfahrzeug (vgl. Kap. 2.4) definiert. Diese Messungen wurden über den Zeitraum mehrere Monate auf verschiedenen Erprobungsfahrten im In- und Ausland aufgezeichnet. Nur so konnte ein ausreichend große Datenbasis mit einem Versuchsträger aufgezeichnet werden.
Die Beschränkung auf ein Fahrzeug hat den Vorteil, dass keine Daten aus verschiedenen Erprobungsträgern kombiniert werden. Das ist zur Schaffung einer Datenbasis wichtig, da nicht gewährleistet werden kann, dass derselbe Hard- und Softwarestand für verschiedene Erprobungsträger über einen Zeitraum mehrere Monate unverändert vorliegt. Somit ist sichergestellt, dass die Datenbasis konsistent ist und vergleichbare Werte für vergleichbare Fahrsituationen aufgezeichnet werden. In dieser Arbeit liegt der Fokus auf Realdaten aus dem Straßenverkehr, um so einen Trainingsdatensatz zu erzeugen der bestmöglich das erwartete Anwendungsgebiet für den Einsatz der ML-Modelle widerspiegelt. In den *Fallstudien I & II* (vgl. Kap. 1.2.3) werden unterschiedliche Netzarchitekturen überprüft. Die Kombination aus Ein- und Ausgangsgrößen spannt einen multidimensionalen Datenraum auf. Dieser wird in den Eingangsdatenraum $\mathcal{D}_{\text{ein}}$ und den zugehörigen Ausgangsdatenraum $\mathcal{D}_{\text{aus}}$ unterteilt [104].
Dabei beinhaltet der Eingangsdatenraum $\mathcal{D}_{\text{ein}}$ alle aufgezeichneten Werte der Eingangsgrößen $\boldsymbol{x}$. Die aufgezeichneten Werte der Zielgrößen $\boldsymbol{y}$ werden durch den Ausgangsdatenraum $\mathcal{D}_{\text{aus}}$ beschrieben. Es gilt also

$$\boldsymbol{x} \in \mathcal{D}_{\text{ein}}, \tag{3.3}$$

$$\boldsymbol{y} \in \mathcal{D}_{\text{aus}}. \tag{3.4}$$

Die Ausgangsgrößen sind die Zielgrößen, welche in der Inferenz durch das ML-Modell möglichst genau berechnet werden sollen. Während des Trainings werden die Modellberechnungen $\hat{\boldsymbol{y}}$ kontinuierlich mit den Zielgrößen $\boldsymbol{y}$ abgeglichen, um eine Optimierung der Modellparameter (Gewichts- und Biaswerte) durchzuführen (vgl. Kap. 4.2). Alle Messgrößen, die nicht direkt den Zielgrößen entsprechen oder von ihnen linear abhängig sind, eignen sich potentiell als Eingangsgrößen für das ML-Modell. Auch hier kann die Anzahl j der Modelleingänge x_i je nach Zielsetzung und Modellarchitektur stark variieren. Die Eingangsgrößen sind für beide Fälle variabel und werden auf ihre Tauglichkeit hin untersucht. Dabei gibt es drei feste Eingangsgrößen, welche direkt aus den verfügbaren Streckendaten des ADAS-Horizonts und dem Fahrzeugaufbau entnommen werden können. Dabei handelt es sich um die Fahrzeuggeschwindigkeit v_{Fzg} bzw. das Geschwindigkeitslimit der wahrscheinlichsten Fahrtstrecke v_{lim}, die Steigung der Fahrtstrecke a_{St} und die Fahrzeugmasse m_{Fzg}. Es handelt sich bei den Messdaten aus dem Fahrzeug um gemessene Realwerte aus dem Fahrbetrieb. Im Anwendungsfall wird das ML-Modell jedoch mit ADAS-Horizontdaten v_{lim} und a_{HRC} aus dem ADAS-Horizont in Form des prädizierten Eingangsvektors $\boldsymbol{x}_{\mathrm{HRC}}$ gespeist. Diese sind deutlich gröber aufgelöst. Die Fahrzeuggeschwindigkeit wird ersetzt durch die gegebenen Geschwindigkeitslimits entlang der Strecke. Das Modell wird mit fein aufgelösten Fahrzeugmessdaten trainiert und getestet. In der Inferenz wird das Modell auf die grob aufgelösten ADAS-Signale $\boldsymbol{x}_{\mathrm{HRC}}$ angewendet. Das Training mit den fein aufgelösten Fahrzeugdaten wurde bewusst gewählt und dem Training mit grob aufgelösten ADAS-Signalen vorgezogen. Ein reines Training nur mit Vorausschaudaten schließt sich aus zwei Gründen aus.

1. Das Training mit Realdaten hoher Auflösung ermöglicht eine bessere Bewertung der Modellgüte der trainierten Modelle. Werden reale Fahrsituationen aufgezeichnet und als Eingangsgrößen für das Modell verwendet, können die errechneten Modellausgänge $\hat{\boldsymbol{y}}$ direkt mit den tatsächlich aufgezeichneten Messsignalen der Zielgrößen $\boldsymbol{y}$ verglichen werden. Somit kann eine direkte Aussage über die Modellgüte getroffen werden. Dazu wird ein Maß für die Abweichung der berechneten Ausgänge $\hat{\boldsymbol{y}}$ im Vergleich zu den gemessenen Zielgrößen $\boldsymbol{y}$ berechnet. Die Modelle können dadurch an die physikalisch-technischen Zusammenhänge angepasst werden. Ein Vergleich mit ADAS-Signalen als Eingangsgrößen $\boldsymbol{x}_{\mathrm{HRC}}$ führt zu einem Vergleich der berechneten Ausgänge $\hat{\boldsymbol{y}}$ mit aufgezeichneten Messsignalen, da die in $\boldsymbol{y}$ definierten Zielgrößen wie die Komponenten des $\dot{Q}_{\mathrm{in}}$ (vgl. Gl. 4.26) nicht im ADAS-Horizont enthalten sind.
2. Das Aufzeichnen der Vorausschauvektoren ist äußerst speicherintensiv. Für jedes Signal muss zu jeden Zeitpunkt der Messung zusätzlich ein 200 Datenpunkte umfassendes Array aufgezeichnet werden. Dies beinhaltet die Vorausschauinformationen des ADAS-Horizonts für die kommenden 8 km. Aufzeichnungen dieser Art wurden speziell für diese Arbeit in gesonderten Messkampagnen

durchgeführt, um eine ausreichenden große Test- und Vergleichsbasis für die Inferenz der Modelle zu erstellen. Eine kombinierte Aufzeichnung all dieser Informationen im alltäglichen Erprobungsbetrieb ist nicht möglich. Die verwendete Software und Speicherkapazitäten limitieren die Anzahl an Messgrößen sowie das Datenvolumen.

In Tabelle 3.2 sind die ausgewählten Eingangs- und Zielgrößen aufgeführt. Als skalares Fehlermaß wird in dieser Arbeit die mittlere quadratische Abweichung bzw. deren Quadratwurzel (engl. *Root Mean Squared Error* (RMSE)) verwendet (vgl. Kap. 4.2.1).

Tabelle 3.2: Übersicht über verfügbare Eingangs- und Zielgrößen

Eingangsgrößen x	**Zielgrößen y**
Fahrzeuggeschwindigkeit v_{Fzg} in km/h	Reibleistung $\dot{Q}_{\mathrm{Reib}}$ in W
Streckensteigung a_{St} in %	Wärmestrom Zylinderkopf $\dot{Q}_{\mathrm{Zyl}}$ in W
Fahrzeugmasse m_{Fzg} in kg	Wärmestrom Getriebeöl $\dot{Q}_{\mathrm{Oel}}$ in W

3.2 Datenvorverarbeitung

In Phase 2 des CRISP-ML(Q) findet die Datenvorverarbeitung statt. Sie besteht aus folgenden drei Schritten:

- Auswahl und Zusammenführen der Daten
- Datenbereinigung (engl. *data cleansing*)
- Merkmalsgenerierung (engl. *feature engineering*)

Diese Schritte beinhalten alle Operationen (vgl. Algorithmus 1), welche benötigt werden um den finalen Datensatz zu erstellen [46]. Dieser besteht aus Trainings-, Test- und Validierungsdatensatz. Die ML-Pipeline ist damit nach dem Prinzip *extract, load and transform (ELT)* aufgebaut welches die Rohdaten erst auf dem Cloud-Backend sammelt und anschließend weiterverarbeitet und transformiert. Diese steht im Gegensatz zu einem anderen gängigen Prinzip welches *extract, transform and load (ETL)* befolgt [49].
Um Skalierbarkeit und eine hohe Wiederholgenauigkeit zu erhalten, werden alle Daten in einem automatisierten Arbeitsablauf, der in Kapitel 4 entwickelten ML-Pipeline (vgl. Abb. 4.4), verarbeitet. Messungen mit fehlenden Werten oder Signalen werden ignoriert. Es findet kein Auffüllen mit Ersatzwerten statt, um die realen physikalischen Abhängigkeiten nicht zu verfälschen. Stark dynamische und verrauschte Signale wie die gemessene Steigung a_{St} werden mittels eines Savitzky-Golay Filters [50] gefiltert. Zudem findet ein Resampling aller verwendeten Signale auf eine Frequenz von 1 Hz statt. Die im Fahrzeug aufgezeichneten Signale liegen auf verschiedenen Datenbussen

(*Controller Area Network with Flexible Data-Rate* (CAN-FD), Ethernet, FelxRay) und werden in Frequenzen zwischen 1 Hz und 100 Hz abgetastet. Hierbei sind im Fahrbetrieb hochdynamische und sicherheitsrelevante Größen wie die Bremspedalstellung oder die Stellung des Fahrpedals in hohen Frequenzen verfügbar. Im TM herrscht die hohe thermische Trägheit vieler Komponenten vor. Dies in Kombination mit den Eingangsdaten aus dem Vorausschauhorizont im Betrieb ermöglicht ein *down sampling* der Messgrößen auf 1 Hz. Diese Transformation reduziert gleichzeitig die Datenmenge und entfernt große Mengen an identischen Datenpunkten, die während Konstantfahrten in großer Zahl aufgezeichnet werden. Da die verwendeten Signale keine hohe Dynamik aufweisen, spricht dies ebenfalls für eine geringere Datenrate. Eine hohe Abtastfrequenz würde dementsprechend die Menge weitgehend identischer Datenpunkte erhöhen und die Datenverteilung stark in Richtung der Konstantphasen verschieben. Im *Feature Engineering*-Schritt werden zusätzliche Signale durch Kombination und Aggregation aus der vorhandenen Datenbasis erstellt. Zudem werden alle Eingangs- und Zielgrößen für das Modelltraining standardisiert. Hierbei werden alle Datenpunkte auf einen Mittelwert von $\mu = 0$ und eine Varianz von $\sigma^2 = 1$ (engl. *zero mean and unit variance*) skaliert. Diese Art der Skalierung wird in der Literatur bevorzugt für KNN mit ReLUs [25]. In dieser Arbeit werden alle Signale auf diese Weise standardisiert. Die entwickelten ML-Modelle benötigen für die Inferenz daher Eingangsgrößen, welche auf Grundlage von μ_{train} und σ^2_{train} des Trainingsdatenraums standardisiert wurden. Der vorgestellte automatisierte Arbeitsablauf ist eine Weiterentwicklung der in [103] erstmals skizzierten Methodik.

3.2.1 Auswahl und Zusammenführen der Daten

Eingangsgrößen können nur aus dem Pool der verfügbaren Vorausschaudaten (vgl. Tab. 3.1) gewählt werden. Die Auswahl der Eingangsgrößen beschränkt sich somit auf die Messgrößen für die Fahrzeuggeschwindigkeit v_{Fzg} in km/h, die Steigung a_{St} der Fahrstrecke in Prozent sowie auf die Fahrzeugmasse m_{Fzg} in kg. Die Fahrzeugmasse ist nicht im ADAS-Horizont enthalten, sondern auf dem CAN-Bus verfügbar und wird mit einer Frequenz von 10 Hz aktualisiert. Für die Streckenlänge eines Vorausschau-Arrays mit 200 Datenpunkten wird diese Größe als konstant behandelt. Eine mögliche, abrupte Änderung der Fahrzeugmasse wird bei der nächsten Aktualisierung des Vorausschauvektors eingepflegt. Der HRC aktualisiert den Vorausschauvektor ebenfalls mit einer Frequenz von 10 Hz. Somit wird eine Modellberechnung mit einer veralteten Fahrzeugmasse im nächsten Rechenschritt überschrieben. Das *feature engineering* wird im Anschluss genutzt um weitere Eingangsgrößen zu erstellen (vgl. Kap. 3.2.3) und diese dem Eingangsvektor $\boldsymbol{x}$ hinzuzufügen. Zur Auswahl der Zielgrößen des Zielvektors $\boldsymbol{y}$ werden die drei gesuchten Komponenten (vgl. Tab. 3.2) des gesamten Wärmestroms $\dot{Q}_{\text{in}}$ in den KKL verwendet. Dafür werden in einer Gitterstudie (vgl. Kap. 4) sowohl Modelle mit allen $k = 3$ Zielwerten, als auch einzelne Modelle für jeden der Zielwerte des Vektors $\boldsymbol{y}$ trainiert und bewertet. Im Vergleich zum Modellbasierten Ansatz substi-

tuieren die trainierten ML-Modelle mehrere, hintereinander geschaltete, PE-Modelle. Dadurch kann die Anzahl der verknüpften Modelle von einem hohen zweistelligen Bereich auf maximal drei Modelle reduziert werden. Dies resultiert in einem geringeren Speicherbedarf des Gesamtsystems. Die modellbasierte Regelung setzt auf einzelnen Modelle, welche aus den gegebenen Eingangswerten, die Motorleistung als Drehmoment M_{mot} und Drehzahl N_{mot} berechnen. Zudem wird mittels eines Ersatzmodells des Getriebes der Getriebegang geschätzt. Diese Modellausgänge werden, basierend auf Kennlinien und Kennfeldern, für die betrachtete Motor-/Baureihenkombination sowie weiteren bauteilspezifischen Parametern und Konstanten berechnet [16]. Die Ergebnisse dieser Modelle werden im nächsten Schritt in thermische Modelle eingespeist, welche den Wärmestrom in Motor, Kühlflüssigkeit und Getriebeöl anhand der berechneten Motorleistung schätzen.

3.2.2 Bereinigen der Daten

Auf den oben beschriebenen Prozessschritt folgt die Bereinigung der Messdaten und steht im Einklang mit dem CRISP-ML(Q) Prozess. Dazu werden im Folgenden verschiedene Arbeitsschritte durchgeführt, um aus den oben auswählten Messgrößen Trainingsdaten hoher Qualität zu erzeugen. Generell gilt, dass der Vorgang der Datenbereinigung stark von den zugrundeliegenden Daten und dem Anwendungsfall abhängt. Folgende Arbeitsschritte werden in der Datenbereinigung durchgeführt.

- Abtastrate reduzieren um somit gleiche/ähnliche Datenpunkte zu vermeiden.
- Ungültige oder leer Datenpunkte löschen.
- Ausreißer erkennen und entfernen.
- Filtern der Daten.
- Entfernen von Datenpunkten, Streckenabschnitten oder Fahrmanövern basierend auf kritischen Randbedingungen
- *optional:* Ausgleichen und reduzieren der Datenverteilungen mittels Methoden der statistischen Versuchsplanung.
- Visualisieren der bereinigten Daten und vergleichen mit Datengrundlage.
- bewerten statistische Momente des Datensatze.

Im Zuge der Datenbereinigung findet zuerst das, in Kapitel 3 beschriebene, Resampling statt. Im Anschluss daran werden verrauschte Signale gefiltert um das Signal-Rauschverhältnis (engl. *Signal-to-Noise Ratio* (SNR)) zu reduzieren. In einem weiteren Schritt werden korrumpierte und fehlende Datenpunkte identifiziert und entfernt. Hierbei kann für ein FNN welches mit diskreten, zeitunabhängigen Datenpunkten trainiert wird anders verfahren werden als für ein RNN ((vgl. *Fallstudien I & II*)) welches mit zusammenhängenden Zeitreihen trainiert wird. Werden Datenpunkte entfernt, wird die betroffene Messung an dieser Stelle in zwei Teile gespalten um keine Sprünge im zeitlichen Verlauf darzustellen. Dies ist vor allem für *Fallstudie 2* von Bedeutung.

In dieser Arbeit wird darauf verzichtet korrumpierte und fehlende Messdaten durch Ersatzwerte zu bereinigen. Grund dafür ist der Anspruch realistische Fahrsituationen und Zustandskombinationen für das Training zu verwenden. Generell gibt es jedoch Verfahren welche je nach Datenlage und Anwendungsfall bspw. den Mittelwert, Median, einen interpolierten oder konstanten Wert als Ersatzwert verwenden [51]. Werden hingegen die Messpunkte ohne zeitlichen Bezug verwendet, können diese Datenpunkte einfach aus dem Trainingsset entfernt werden, da die Datenpunkt später zufällig aus dem verfügbaren Trainingsdatensatz gezogen werden.
Im Falle einer zu großen oder nicht normal verteilten Datenbasis kann eine Methode der statistischen Versuchsplanung (engl. *Design of Experiments* (DoE)) wie ein Sobol-Versuchsplan oder das Latin-Hypercube-Sampling eingesetzt werden. Mit dieser Methodik lässt sich beispielsweise eine angestrebte Datenverteilung zugrundelegen. Die Messdatenpunkte welcher dieser Verteilung am besten entsprechend werden (in reduzierter Zahl) als neue Datenbasis gewählt [24], [45], [52], [53].

Filter und Resampling

Die vorliegenden Daten werden aus verschiedenen Gründen gefiltert. Das Steigungssignal a_{St} wird im Fahrzeug sehr verrauscht aufgezeichnet und im Zuge der Datenbereinigung mittels eines Savitzky-Golay Filters geglättet. Dieses Tiefpassfilter arbeitet im Zeitbereich und erzeugt ein rauscharmes Signal. Das ist für das Training der ML-Modelle entscheidend, da starkes Rauschen viele unterschiedliche Datenpunkte zu identischen Betriebspunkten liefern kann. Zudem wird das Modell in der Anwendung konkrete Steigungsverläufe pro Streckenabschnitt als Eingangsgröße a_{HRC} erhalten. Auch das aufgezeichnete Geschwindigkeitssignal v_{Fzg} wird in verschiedenen Stufen geglättet und mit einem Hystereseband belegt, um so verschieden mögliche Geschwindigkeiten zu erzeugen welche dann in einer Gitterstudie untersucht werden. (vgl. Kap. 3.2.3).
Hochfrequente Messungen werden auf eine vorgegebene Frequenz reduziert (engl. *down sampling*). Somit kann die Informationsdichte und die Datenmenge individuell auf den Anwendungsfall angepasst werden. Beispielsweise herrschen unterschiedliche Anforderungen an die Informationsdichte für das Signal der Kühlmitteltemperatur am Motorausgang im Vergleich zum Signal der Fahrzeugbeschleunigung. Während sich die Temperatur auf Grund thermischer Trägheit und einer großen thermischen Masse des Motorblocks nur langsam ändert, stellt das Signal der Fahrzeugbeschleunigung einen hochdynamischen Prozess dar, der feiner abgetastet werden muss, um Änderung zu dokumentieren.

Aufbereiten der Daten aus dem ADAS-Horizont

Die prädizierten Streckendaten in Form des MPP aus dem HRC müssen aufbereitet werden, damit sie für eine Bewertung der trainierten Modelle verwendete werden

können. Die Inferenz wird mit Eingängen $\boldsymbol{x}_{\mathrm{HRC}}$ aus dem sogenannten ADAS-Horizont überprüft. Die Eigenschaften des ADAS-Horizonts sind in Kapitel 2.2.3 beschrieben. Da die Daten vom HRC als positionsbezogener Ereignisvektor mit einem zugehörigen Positionsvektor bereit gestellt werden, muss eine Transformation in zeitbasierte Signale stattfinden [16]. In den aufgezeichneten Fahrzeugmessdaten liegen die Informationen zum MPP in Form von Arrays mit 200 Einträgen pro Zeitpunkt der Messung vor. Für jeden Zeitpunkt der Messung existiert für jedes Signal ein Array mit 200 Einträgen (vgl. 3.1), welches die zum Zeitpunkt der Messung vorhandenen Informationen zum MPP beinhaltet. Diese Arrays werden im Zuge der Datenaufbereitung in zeit-basierte Datenpunkte umgewandelt. Diese können als Zeitreihendaten $\boldsymbol{x}_{\mathrm{HRC}}$ zur Inferenz der trainierten ML-Modelle verwendet werden. Dieser Schritt wird durchgeführt, um eine Evaluation der ML-Modelle anhand der im Steuergerät vorliegenden Eingangsdaten durchführen zu können. Dafür wurden Testdaten aufgezeichnet, welche sowohl die gefahrene Fahrstrecke mit allen relevanten Eingangs- und Zielgrößen beinhalten, als auch die zu jedem Zeitschritt der Messung im Fahrzeug verfügbaren Informationen zum MPP. Dies ist wie oben erwähnt sehr speicherintensiv und kann daher nicht bei der Aufzeichnung aller Trainingsdaten durchgeführt werden.

Die Aufbereitung der Horizontinformationen erfordert eine Umwandlung der positionsbezogenen in zeitbasierte Datenpunkte. Das Array eines Signals enthält nur dann einen neuen Eintrag, sofern im Horizont ein neues Ereignis eintritt. Diese Methodik wird im HRC ausgeführt um eine größtmögliche Datenkompression zu gewährleisten. Ein Ereignis ist stets eine Veränderung eines der verfügbaren Signale (vgl. Tab. 3.1) des MPP. Kern ist dabei der Positionsvektor, der 200 Einträge für die vorausliegende Strecke s_{HRC} in Metern bereithält. Ändert sich ein Wert wie die Steigung a_{HRC} oder das Geschwindigkeitslimit v_{lim} an einer Position, so enthält der Positionsvektor zu diesem Ereignis den Eintrag als Abstand zur aktuellen Position. Dazu passend enthält das Array der Geschwindigkeit $\boldsymbol{v}_{\mathrm{lim}}$ den Eintrag für das Geschwindigkeitslimit an der selben Position im Array. Ändert sich in dieser Zeit kein weiterer Wert wie bspw. die Steigung, so erhält das zugehörige Array $\boldsymbol{a}_{\mathrm{HRC}}$ einen neuen Eintrag. Dieser ist identisch mit dem vorherigen Arrayeintrag.
Die 200 Einträge des Positionsvektors repräsentieren Abstände des Fahrzeugs zum nächsten Ereignis im HRC. Die Zusammensetzung des Vorausschauvektors wird mit einer Frequenz von 10 Hz erneuert. Dadurch werden Änderungen der Fahrstrecke oder unerwartete Abbiegemanöver schnell erfasst. Anhand der bekannten Beziehung von Geschwindigkeit und Wegstrecke

$$v_{\mathrm{lim}} = \frac{s_{\mathrm{pos}}}{t_{\mathrm{St}}}, \tag{3.5}$$

$$t_{\mathrm{St}} = \frac{s_{\mathrm{pos}}}{v_{\mathrm{lim}}}, \tag{3.6}$$

kann die benötigte Fahrzeit t_{St} für jeden der Streckenabschnitte in $\boldsymbol{s}_{\mathrm{pos}}$ bestimmt werden [16], [39].

3.2.3 Feature Engineering

Um die vorhandene Datengrundlage mit zusätzlichen Informationen zu erweitern werden mittels *feature engineering* weitere Signale erstellt. Dafür werden vorhandene Signale miteinander kombiniert, durch Transformation und Aggregation werden neue Signale erzeugt. Ziel ist es eine Auswahl an zusätzlichen Eingangsgrößen x_i zu erhalten, um im Verlauf des Trainings der ML-Modelle (vgl. Kap. 4) mittels einer Gitterstudie die besten Eingangswerte für den gegebenen Anwendungsfall zu erhalten. Transformationen stellen Rechenoperationen in Bezug auf ein Signal dar. Z.B. die Bildung des natürlichen Logarithmus oder die Anwendung eines Filters auf ein vorhandenes Signal. Als Aggregation bezeichnet man etwa Erstellung eines neuen Signals auf Basis eines oder mehrere Ursprungssignale. Beispielsweise können statistische Werte oder Ableitungen über ein definiertes Fenster von n Zeitschritten verwendet werden. Um historische Informationen aus den vorangegangenen Zeitschritten der Messung zu erhalten werden verschiedene Signale aus den Mittelwerten über die letzten $5, 10, 20$ und 50 Zeitschritte erzeugt. Eine Differenzierung des Signals über die selben Zeitfenster erzeugt weitere Signale. Folgende Methoden werden verwendet: Glättung des Signals in verschiedenen Stufen mittels Savitzky-Golay Filter, eingrenzen des Signals in ein Hystereseband, unterteilen des Wertebereichs in Klassen (sogenannte *bins*), Differentiation des Signals nach der Zeit, Bildung des gleitenden Mittelwertes über n vorangegangene Zeitschritte (damit kann dem Signal eine Information aus der Historie mitgegeben werden). Diese zusätzlichen Eingangssignale enthalten Informationen über vorangegangene, durchschnittliche Geschwindigkeit, den Energieverbrauch, Fahrverhalten, Steigungsabschnitte und vieles mehr ohne dass dafür ein KNN mit rekurrenten Strukturen nötig ist. Diese sogenannten RNN betrachten jeweils eine größere Spanne (engl. *window*) der Eingangsgröße, um den zeitlichen Zusammenhang zu berücksichtigen. Ein weiteres Ziel des *feature engineering* die Signale aus den Fahrzeugmessungen so zu verarbeiten, dass sie nahe am diskreten Verlauf der Vorausschaudaten aus dem HRC liegen. Damit kann ein Training auf Basis dieser Signale mit einem Training mit Signalen mit realem Datenverlauf verglichen werden. Dies wird verwendet, da ein Training nur mit grob aufgelösten ADAS-Daten nicht sinnvoll ist, wie oben ausgeführt wurde (vgl. Kap. 3.2.2). Eine Gitterstudie dient beim Training der ML-Modelle (vgl. Kap. 4) dazu die am besten geeignete Kombination von Eingangssignalen zu ermittelt. Tabelle 3.3 fasst die verwendeten Methoden der Merkmalsgenerierung zusammen.

3.2.4 Skalieren der Daten

Im Anschluss an das *feature engineering* werden die Daten in Trainings-, Validierungs- und Testset aufgeteilt. Da sowohl die Eingangs- als auch die Zielgrößen auf verschiede-

Tabelle 3.3: Übersicht verwendeter Methoden der Aggregation und Transformation zur Erstellung zusätzlicher Signale im *feature engineering*

Methodik	Parameter	Signale
Filtern	Filterfenster $w \in (5, 11, 21, 51, 101, 201)$ Filterordnung $n_\mathrm{F} \in (2, 3, 5)$	$a_\mathrm{St}, v_\mathrm{Fzg}$
Mittelwert	über n Zeitschritte $n \in (5, 10, 20, 50)$	$a_\mathrm{St}, v_\mathrm{Fzg}$
gemittelte Ableitung	über n Zeitschritte $n \in (5, 10, 20, 50)$	$a_\mathrm{St}, v_\mathrm{Fzg}$
Hystereseband	l_hyst in km/h gilt für $-l_\mathrm{hyst} < v_\mathrm{Fzg} < l_\mathrm{hyst} \mid l_\mathrm{hyst} \in (2, 5, 10)$	v_Fzg
Wertebereiche entfernen	$N_\mathrm{mot} < 0, v_\mathrm{Fzg} = 0$	$N_\mathrm{mot}, v_\mathrm{Fzg}$
Gangzustände entfernen	Gang $S_\mathrm{Gtr} \notin (10, 11, 12, 13)$	S_Gtr

nen Skalen liegen können würden Signale mit größeren Absolutwerten das Training dominieren. Sie führen zu einem starken Bias des ML-Modells hin zu großen Wertebereichen wohingegen Signale mit kleinen Absolutwerten weniger stark berücksichtigt werden [24], [31]. Um diese Effekte zu vermeiden, ist es gängige Praxis die Eingangs- und Zielgrößen des Trainingssets zu skalieren. Dabei werden in der Literatur vor allem zwei Verfahren angeführt. Die gängigste Variante ist das Normieren auch Min-Max-Skalierung genannt (engl. *min-max-scaling*) [54]. Hierbei werden alle Größen des Datensatzes mittels ihrer jeweiligen globalen Minima und Maxima auf einen Wertebereich zwischen 0 und 1 normiert. Eine weitere Methode ist das Standardisieren der Datenpunkte. Dieses passt den Wertebereich auf eine Verteilung mit dem Mittelwert null und einer Varianz von eins an. Für beide Verfahren wird die Skalierung auf Basis des Trainingsdatensatzes durchgeführt, Validierungs- und Testdaten werden dann anhand der selben Verhältnisse skaliert. Dies gilt ebenfalls im Anwendungsfall für die Eingangsgrößen. Um diese mit dem ML-Modell verwenden zu können müssen sie ebenfalls entsprechend skaliert werden. Es folgt einer genauere Beschreibung der beiden gängigen Methoden zur Skalierung der Daten.

Normierung

Eingangs- und Zielgrößen werden anhand ihrer globalen Minima und Maxima auf einen Wertebereich zwischen 0 und 1 skaliert (vgl. Gl. 3.7). Das Verfahren bietet den Vorteil, dass weitere neu aufgenommene Messdaten basierend auf den selben Minima und Maxima normiert werden können, solange diese nicht die globalen Minima und Maxima des bereits vorhandenen Datenraums unter- bzw. überschreiten. Um gegen diese Möglichkeit eine größere Sicherheit zu erzeugen können die Minima und Maxima mit zusätzlichen Toleranzwerten belegt werden. Diese können anhand der physikali-

schen Grenzen oder auch mittels eines umfassenden Systemverständnisses ermittelt werden. Die Normierung der Daten berücksichtigt allerdings nicht die Verteilung und den Mittelwert der Datenmenge, welche sich durch das zusätzliche aufnehmen von Messdaten ändern können, auch wenn Minima und Maxima erhalten bleiben. Darüber hinaus kann ein einziger Ausreißer (engl. *outlier*) in der zu Grunde liegenden Datenbasis dazu führen, dass die globalen Minima oder Maxima unverhältnismäßig verschoben werden und der Rest der Datenpunkte in einem nicht repräsentativen Bereich normiert wird [54].

$$x_{i_{\mathrm{nrm}}} = \frac{x_i - x_{i_{\mathrm{min}}}}{x_{i_{\mathrm{max}}} - x_{i_{\mathrm{min}}}} \tag{3.7}$$

Gleichung 3.7 zeigt die Normierung am Beispiel einer Eingangsgröße x_i aus dem Eingangsvektor $\boldsymbol{x}$. Dabei ist $x_{i_{\mathrm{nrm}}}$ die normierte Eingangsgröße.

Standardisierung

Das Verfahren der Standardisierung (engl. *standardization*) ist weniger anfällig für Ausreißer in den Daten. Der Grund hierfür ist die Subtraktion des Mittelwert $\overline{x}_i$ von allen Datenpunkte einer Eingangsgröße $\boldsymbol{x}$, dadurch ergibt sich eine Verteilung um den Mittelwert $\overline{x}_i = 0$. Um eine Varianz von $v = 1$ zu erhalten werden die Datenpunkte durch die Standardabweichung σ_i der Größe dividiert [55], [56]. Gleiches gilt für die Skalierung der Zielgrößen y_l.

$$x_{i_{\mathrm{std}}} = \frac{x_i - \overline{x}_i}{\sigma_i} \tag{3.8}$$

In Kapitel 4.1.4 wird die Verwendung von ReLUs zur Aktivierung der Neuronen definiert. Dafür müssen alle Eingangs- und Zielgrößen standardisiert vorliegen um bestmögliche Ergebnisse zu erzielen [25], [26]. Daher werden skalierte Eingangs- und Zielgrößen im folgenden nur noch als x_i und y_l bezeichnet. Das Subskript für die Form der Skalierung entfällt. Der Prozess der Skalierung findet nach der Aufteilung der Datenmenge in Trainings-, Test- und Validierungsdaten statt. Dabei werden alle Datensätze entsprechend den statistischen Werten des Trainingssets skaliert.

3.2.5 Ablegen und Aufrufen der Daten

Am Ende der Datenvorverarbeitung (Phase 3) werden die erzeugten Trainingsdaten im spaltenorientierten **.parquet*-Format abgelegt um für das Training minimale Zugriffszeiten zu gewährleisten. Das verwendete Dateiformat wird unter anderem in verteilt arbeitenden Systemen eingesetzt, um die Zugriffszeiten zu reduzieren [57]. Besonderes für die Implementierung des Modelltrainings auf einer Cloud-Instanz bietet sich das **.parquet*-Format an, da die Leistungsfähigkeit einer Cloudumgebung nur durch das Verteilen der Operationen auf eine Vielzahl von Rechenkernen (*Central Processing Units* (CPUs) und *Graphics Processor Units* (GPUs)) ausgeschöpft werden kann. Dafür

ist ein kompatibles Dateiformat unumgänglich. Das **.parquet*-Format kann direkt als *Spark Dataframe* für das verteilte Rechnen auf Cloudressourcen geladen werden. Für das verteilte Rechnen auf skalierbaren Clustern wird die *Apache Spark Enigne* verwendet, welche verteiltes Rechnen, Datenanalyse und maschinelles Lernen ermöglicht. Es existieren Programmierschnittstellen (engl. *Application Programming Interfaces* (APIs)) für Programmiersprachen wie *Python, SQL, R* oder *Scala*. All diese Sprachen sind im Bereich des *Data Engineering* weit verbreitet [58]. **.parquet*-Dateien können ebenso von klassischen *Python* Bibliotheken geladen werden. Ein Beispiel hierfür ist die *Pandas* Programmbibliothek, welche die Dateien direkt in einen *Pandas Dataframe* lädt. Die genannten Vorteile und die Kompatibilität über verschiedene Systeme hinweg machen das **.parquet*-Format für den Einsatz auf Desktoprechnern, Remote-Servern und Cloud-Instanzen gleichermaßen anwendbar. Die verwendete ML-Plattform *TensorFlow* verwendet für das Training der Modelle beispielsweise *Pandas Dataframes*. Aus diesen Gründen werden die Daten nach dem *precprocessing* im **.parquet*-Format abgespeichert.

3.3 Methodik zur Beschreibung von Datenräumen zur Modellbewertung

Das Training von ML-Algorithmen ist stark abhängig von der Verteilung, Skalierung und Menge der Datenpunkte. Diese haben direkten Einfluss auf die Fähigkeit des Modells zu generalisieren. Die Anzahl der Eingangs- und Zielgrößen eines KNN definiert die Dimension des Datenraums. In dieser Arbeit wird die Verwendung von mindesten drei und maximal zehn Eingangsgrößen x_i untersucht (vgl. Gl. 3.1). Dem gegenüber stehen bis zu drei Zielgrößen y_l, welche durch die ML-Modelle berechnet werden. Für den Ein- und Ausgangsdatenraum gilt

$$\mathcal{D}_{\text{ein}} \subseteq \mathbb{R}^{\text{j}}; \forall \text{j} \in [3, 10], \tag{3.9}$$

$$\mathcal{D}_{\text{aus}} \subseteq \mathbb{R}^{\text{k}}; \forall \text{k} \in [1, 3]. \tag{3.10}$$

Jeder Eingangsvektor $\boldsymbol{x}$ und der zugehörige Zielvektor $\boldsymbol{y}$ liegen somit in den gegebenen Ein- und Ausgangsdatenräumen

$$\boldsymbol{x} \in \mathcal{D}_{\text{ein}}, \tag{3.11}$$

$$\boldsymbol{y} \in \mathcal{D}_{\text{aus}}. \tag{3.12}$$

Die Trainingsdaten spannen einen mehrdimensionalen Datenraum auf, welcher in seiner hohen Dimensionalität nicht alleine durch graphische Abbildung und Vergleiche beschrieben werden kann (vgl. Gl. 3.9 u. 3.10). Die Verteilung der Datenpunkte im Raum beeinflusst die Fähigkeit des Modells über den gesamten Anwendungsraum generalisieren zu können. Gibt es Bereiche mit Häufungen oder einer sehr geringen

Zahl an Datenpunkten, besteht für diese Bereiche die Gefahr der Über- oder Unteranpassung des Modells. Eine Gleichverteilung der Trainingsdaten in Bezug auf den Einsatzbereichs des KNN ist anzustreben [24], [25]. Eine Häufung von Trainingsdaten verschiebt den Fokus der Anpassung im Training auf die überrepräsentierten Datenpunkte und erschwert es so eine gute Genrealisierung des ML-Modells im gesamten Einsatzdatenraum zu erreichen. Gleiches gilt für die Bewertung der Modellberechnungen. Mit einem homogen im Einsatzdatenraum verteilten Testdatensatz kann die Modellgüte des KNN für den definierten Anwendungsbereich überprüft werden. Anhand des Testdatensatzes wird das Gütekriterien für die Bewertung der trainierten ML-Modelle berechnet (vgl. Kap. 4.2.1).
Um die Berechnungen der ML-Modelle nach der Inferenz plausibilisieren zu können wird in Kapitel 5 die RPA eingeführt. Für die Anwendung der RPA ist eine mathematische Beschreibung der Ein- und Ausgangsdatenräume nötig um ein robustes Verhalten der ML-Modelle zu gewährleisten. Eine Verbindung aller vorverarbeiteten **.parquet*-Dateien aus dem *preprocessing* zu einem Gesamtdatensatz $\mathbb{S}_{\text{ges}}$ ist der erste Schritt.

3.3.1 Datenaufteilung für Training und Bewertung der Modelle

Für das Modelltraining wird der Gesamtdatensatz $\mathbb{S}_{\text{ges}}$ in einen Trainings-, Validierungs- und Testdatensatz aufgeteilt. Es entstehen drei Datensätze.
Der Testdatensatz $\mathbb{S}_{\text{test}}$ wird mit dem in der Literatur empfohlenen Anteil von 25 % vom Gesamtdatensatz $\mathbb{S}_{\text{ges}}$ abgetrennt [24], [25], [45]. Für *Fallstudie I* werden die Datenpunkte randomisiert entnommen. Für *Fallstudie II* werden zusammenhängende Messungen entfernt, da die RNN auf zusammenhängende Zeitreihen angewendet werden (vgl. Kap. 4.1). Da einzelne, lange Messfahrten oft große Teile an Konstantfahrten beinhalten, wurden Messungen mit einer repräsentativen Datenverteilung ausgewählt. Dazu wurden die statistischen Momente (vgl. Kap. 2.1) des Gesamtdatensatzes berücksichtigt. Zudem kann anhand von Violinen-Diagramme die Datenverteilung einzelner Signale visuell verglichen werden. Anhand des Testdatensatz $\mathbb{S}_{\text{test}}$ wird die Generalisierungsfähigkeit des ML-Modell nach dem Training bewertet [24], [25]. Die Auswahl des bestens ML-Modells in den Gitterstudien wird anhand der für $\mathbb{S}_{\text{test}}$ ermittelten Modellgüte durchgeführt (vgl. Kap. 4.2.1).
Die verbleibenden Datenpunkte des Gesamtdatensatzes $\mathbb{S}_{\text{ges}}$ werden im Verhältnis 80 : 20 in einen Trainingsdatensatz $\mathbb{S}_{\text{train}}$ (80 %) und einen Validierungsdatensatz $\mathbb{S}_{\text{vali}}$ (20 %) aufgeteilt.
Das Modelltraining wird mit $\mathbb{S}_{\text{train}}$ durchgeführt. Anhand der Verlustfunktion werden die Modellparameter optimiert (vgl. Kap. 4.2.2). Der Validierungsdatensatz $\mathbb{S}_{\text{vali}}$ wird nicht zum Training des Modells verwendet. Er dient als unabhängiger Datensatz zur Überprüfung der Generalisierungsfähigkeit des Modells während des Trainings [24], [25]. In der Praxis wird während des Modelltrainings, in einem festgelegten Rhythmus (beispielsweise alle 10 Epochen), ein Validierungsschritt durchgeführt. Die Verlust-

funktion für $\mathbb{S}_{\text{train}}$ und $\mathbb{S}_{\text{vali}}$ wird verglichen. Weichen Trainingsverlustfunktion und Validierungsverlustfunktion über mehrere Epochen stark voneinander ab, wird eine Überanpassung erkannt und das Training vorzeitig abgebrochen (engl. *early stopping*) [24], [25], [59].
Bei der Verwendung von realen Messdaten aus dem Fahrzeug kann nicht davon ausgegangen werden, dass alle möglichen Betriebszustände mit derselben Wahrscheinlichkeitsdichtefunktion repräsentiert sind [104], [60]. Für die *Fallstudien 1 & 2* werden die Messdaten in ihrer gegebenen Verteilung verwendet, um alle verfügbaren Informationen für das Modelltraining nutzbar zu machen. Der Trainingsdatensatz ist zwangsläufig unvollständig, da im realen Fahrbetrieb immer nur ein Subset des möglichen Eingangsdatenraums erfasst werden kann [104], [60] Die Zielgrößen sind die Komponenten des gesamten Wärmestroms in den KKL. Der gesamte Wärmestrom $\dot{Q}_{\text{in}}$ setzt sich aus drei Komponenten zusammen. Die erzeugte Reibleistung der Kolben im Motorblock $\dot{Q}_{\text{Reib}}$, der Wärmestrom $\dot{Q}_{\text{Zyl}}$ aus dem Zylinderkopf in den KKL sowie der Wärmestrom $\dot{Q}_{\text{Oel}}$ aus dem Getriebeölkühler in den KKL. Da es sich dabei um einen Energieeintrag handelt, kann dieser nicht im Fahrzeug als Signal gemessen werden. Es handelt sich um eine Regelgröße, welche im Fahrzeug durch die physikalischen Modelle bereitgestellt wird. In [16] wurde nachgewiesen, dass die Berechnung dieses Energieeintrags sehr gut mit genauen Simulationsmodellen und Windkanalmessungen übereinstimmt. Da die physikalischen Modelle im Fahrzeug die exakten Signalwerte wie Motordrehzahl N_{mot} und Motordrehmoment M_{mot} mit einem Motorkennfeld kombinieren, erreichen die Berechnungen eine hohe Genauigkeit für den Anwendungsfall. Die drei Komponenten des gesamten Wärmestroms $\dot{Q}_{\text{in}}$, werden als einzelne Ausgangsgrößen zu den Zielgrößen des Modelltrainings. Die Notwendigkeit eines standardisierten Testdatensatzes wird ebenfalls in der *German Standardization Roadmap on Artificial Intelligence* gefordert [41].

3.3.2 OCSVMs zur Definition des Datenraums

Um das Modelltraining vollständig mittels einer ML-Pipeline automatisieren zu können, werden die Ein- und Ausgangsdatenräume anhand von OCSVMs mit einer einhüllenden EG (Hyperebene) beschrieben. Diese wird zum einen zur Klassifikation neuer Messdaten benötigt. Zum anderen basiert die in Kapitel 5 entwickelte Methodik zur Plausibilisierung der ML-Modellberechnungen (RPA) auf der Definition valider Ausgangsdatenräume. Die Logik zur Bewertung einer plausiblen Modellberechnung definiert *Backwards Reachable Sets* (BRSs) (deutsch: rückwirkend erreichbare Datensätze) [61]. Die Entwicklung und Anwendung dieser Methodik wird in Kapitel 5 beschrieben. Grundlegend wird ein zulässiger Datenraum erzeugt, dessen Grenze eine EG (engl. *decision boundary*) darstellt, um plausible von implausiblen Modellberechnungen zu unterscheiden. Im einfachsten Fall kann der zulässige Datenraum für die Modellierung durch Minima und Maxima der Ein- und Ausgangsgrößen definiert werden. Resultat ist ein Hyperquader mit $n = j + k$-Dimensionen für n unterschiedliche Einflussgrößen.

Diese ergeben sich aus dem Eingangsvektor $\boldsymbol{x}$ und dem Vektor der Zielgrößen $\boldsymbol{y}$ (vgl. Gl. 3.1). Die Anzahl aller Ein- und Ausgangsgrößen eines KNN definieren somit die Dimension des Datenraums. Im realen Anwendungsfall geben physikalisch und technische Beschränkungen der einzelnen Signale zusätzliche Grenzen des Datenraums vor. Zudem können Abhängigkeiten der Signale untereinander und die Logik der Regelalgorithmen weitere Bedingungen (engl. *constraints*) hinzufügen. Somit ist ein Datenraum im Anwendungsfall selten durch klare, linear verlaufende Grenzen bestimmt. Konvexe Datenräume ohne derartige Grenzen können beispielsweise mit dem *Convex-Hull*-Algorithmus mit ausreichender Genauigkeit beschrieben werden [45]. Das Auffinden von Hohlräumen innerhalb der Hülle ist mit dem *Convex-Hull*-Algorithmus nicht möglich. Für komplexere, konkave Datenräume eignet sich der Algorithmus nicht. Da in dieser Arbeit ein automatisiertes Erfassen und Beschreiben der gesammelten Messdaten erfolgen soll, wird ein Algorithmus gewählt der auch mit konkaven und teilweise nicht gefüllten Datenräumen umgehen kann. Eine mathematische Definition der möglichen Datenräume ist vorab nicht möglich. Ein Grund dafür ist, dass in der Gitterstudie zum Training der ML-Modelle verschiedenste Ein- und Ausgangssignalkombinationen untersucht werden, um das beste Modell für den gesammelten Datensatz zu ermitteln. Dieser kann je nach Anwendungsfall und Dauer der Messkampange eine unterschiedliche Größe und Verteilung aufweisen. Um das Ziel einer automatischen Datensammlung und Modellerstellung (vgl. Kap. 1.2) zu erfüllen, werden OCSVM zur Beschreibung der mehrdimensionalen Datenräume verwendet. OCSVMs wurden von Schölkopf, Williamson, Smola et al. in [62] vorgestellt. Sie sind bis zu einer Dimension des Eingangsraums von $n = 10$ anwendbar und stellen einen effizienten Algorithmus zur Klassifizierung von Datenpunkten als *innerhalb* oder *außerhalb* einer EG dar. Nach dem Training auf einem Datensatz können sie schnell und ohne großen Speicherbedarf ausgeführt werden. OCSVMs führen eine Einklassen-Klassifizierung aus. Anderes als *Support Vector Machine*s (SVMs), welche binäre Datenklassen separieren können, berechnen OCSVMs eine Einhüllende, sogenannte Hyperebene, welche die gegebenen Datenpunkte umschließt. Diese Hyperebene wird als EG verwendet um neue Daten als *innerhalb* oder *außerhalb* des eingehüllten Datenraums zu klassifizieren. Die EG wird ermittelt, indem die zu beschreibenden Datenpunkte in einen multidimensionalen Suchraum $\mathcal{Z}$ transformiert werden, welcher mittels einer Hyperebene linear separiert werden kann. Es existiert also eine Funktion

$$\phi : \mathcal{X} \rightarrow \mathcal{Z} \tag{3.13}$$

die den Originaldatenraum $\mathcal{X}$ in den Suchraum $\mathcal{Z}$ transformiert (engl. *mapping*) [62]. Dabei entstehen zwei Halbräume, einer beinhaltet (nahezu) alle Datenpunkte, wohingegen der zweite (nahezu) keine Datenpunkte beinhaltet. Dabei werden die Datenpunkte solange in immer höher dimensionale Datenräume transformiert, bis es eine entsprechende Hyperebene gibt. In diesen hoch dimensionalen Suchräumen müssten komplexe Vektoroperationen durchgeführt werden, um die Hyperebene aufzuspannen und zu

berechnen. Diese Berechnung erfordern eine hohe Rechenleistung. Jedoch kann mit Hilfe des sogenannten *kernel trick* dieser hohe Rechenaufwand drastisch reduziert werden. Diese Methode erlaubt es die Punktprodukte der hoch dimensionalen Vektoren nicht im Suchraum berechnen zu müssen, sondern diese durch eine Kernelfunktion zu ersetzen, welche im original Datenraum berechnet werden kann [63]. Die errechnete Hyperebene stellt die EG dar, welche verwendet wird, um neue Datenpunkte als innerhalb oder außerhalb des zulässigen Datenraums zu klassifizieren. Die Berechnung der EG bezeichnet man als Training der OCSVM. Für das Training ist die gesamte Datengrundlage nötig. Danach kann die EG mittels einiger weniger Stützvektoren (engl. *support vectors*) aufgespannt werden und bietet somit einen effizienten, speicher- und rechenzeitgünstigen Klassifizierungsalgorithmus. Für das Training der OCSVM müssen verschiedene Hyperparameter identifiziert werden, welche den Bruchteil der vernachlässigten/ignorierten Datenpunkte und der erstellten Stützvektoren definieren. Diese Größen beeinflussen direkt die Form der EG (vgl. Abb. 3.2) im original Datenraum [64]. Dabei sind die beiden wichtigsten Hyperparameter γ und ν. Der Wert γ definiert die Form der Hyperebene, welche zwischen einer Hypersphäre und einer beliebigen konkaven Form variiert werden kann. Dabei ist γ ein Kernel-Koeffizient, der ja nach verwendetem Kernel angepasst werden muss. Um den Anteil an Datenpunkten zu definieren, welche als Ausreißer ignoriert werden sollen, muss ν variiert werden. Gleichzeitig definiert ν auch den Anteil an Datenpunkten, welche als Stützvektoren verwendet werden, um die Hyperebene aufzuspannen. Geometrisch ist eine Hyperebene ein Unterraum mit einer Dimension weniger als der umgebende Raum (Suchraum). Die Hyperebene lässt sich folgendermaßen beschreiben:

$$\omega \cdot \mathbf{z} + b = 0 \tag{3.14}$$

Der Vektor ω und der Biaswert b beschreiben die Hyperebene im Suchraum indem $\mathbf{z}$ ein Vektor zu einem Datenpunkt in diesem Suchraum darstellt. Für einen zweidimensionalen Suchraum ist Gl. 3.14 die Geradengleichung mit $\omega = (m, -1)$ und $\mathbf{z} = (z_1, z_2)$. Hierbei ist m die Geradensteigung. Im eindimensionalen Raum definiert Gleichung 3.14 einen Punkt, eine Gerade im zweidimensionalen Raum und eine Ebene im dreidimensionalen Raum. Für den Suchraum $\mathcal{Z} \in \mathbb{R}^n$ stellt Gl. 3.14 also eine Hyperebene der Dimension $n-1$ dar.
Nach Cortes und Vapnik ist eine optimale Hyperebene dadurch definiert, dass eine lineare EG mit maximalem Abstand (engl. *maximal margin*) zwischen den Stützvektoren der zu unterscheidenden zwei Klassen (für SVM) vorliegt [65]. Dafür wurde nachgewiesen, dass nur ein kleiner Teil der Trainingsdaten nötig ist, um als Stützvektoren (engl. *support vectors*) für eine optimale EG zu dienen [65]. Im Zuge des Trainings einer SVM wird die optimale Hyperebene gesucht, welche die Daten bestmöglich (linear) separiert [63], [65]. Es wird also eine Lösung für Gl. 3.14 gesucht, derart, dass die Hyperebene den größtmöglichen Abstand zu allen Datenpunkten hat. Im Falle einer SVM findet eine binäre Klassifizierung zwischen zwei Datengruppen statt. Der Abstand

der Hyperebene muss zwischen beiden Datengruppen maximiert werden [65]. Für eine OCSVM wird der Abstand zum Ursprung der Hyperebene maximiert. Folgendes Optimierungsproblem muss gelöst werden:

$$\omega \in \mathcal{Z}, \xi \overset{\min}{\in} \mathbb{R}^l, \rho \in \mathbb{R} \qquad \frac{1}{2} \parallel \omega \parallel^2 + \frac{1}{\nu l} \sum_i \xi_i - \rho, \tag{3.15}$$

$$abhg.von \qquad (\omega \cdot \phi(\mathbf{z}_i)) \geq \rho - \xi_i, \xi_i \geq 0 \tag{3.16}$$

Es handelt sich um die Optimierung eines *soft margin classifiers*, da die Schlupfvariable ξ (engl. *slack variable*) in der Optimierung zu kleine Abstände (engl. *margins*) der Stützvektoren zur Hyperebene bestraft. Durch das Aufsummieren der individuellen Fehler ξ_i in Gl. 3.15 wird ein maximaler Abstand der EG bei gleichzeitig minimalem Fehler gesucht. Ziel ist es, den Trainingsdatensatz mit einer möglichst geringen Anzahl an Fehlern zu separieren [65]. Der Parameter $\nu \in (0, 1]$ muss im Zuge des Trainings der OCSVM ebenfalls optimiert werden. Er kontrolliert den Anteil an ignorierten Ausreißern (engl. *outliers*) sowie den Anteil an verwendeten Stützvektoren in Bezug auf den Datensatz im Suchraum $\mathcal{Z}$. Der Parameter $l \in \mathbb{N}$ gibt die Anzahl der Datenpunkte an. Zusätzlich kann ein fixes Offset ρ definiert werden, welches durch die *slack variable* ξ_i ausgeglichen werden kann [62].
Die, durch das Training der OCSVM erzeugte, EG kann beliebig komplexe, mehrdimensionale Datenräume beschreiben. Dabei können auch nicht belegte Bereich innerhalb des Datenraums erkannt und ausgeschlossen werden. Beim Training der OCSVM muss eine Optimierung für das Anpassen der Hyperparameter (*hyperparameter tuning*) durchgeführt werden. Dabei gilt es einen Kompromiss zwischen einer *losen* und einer *straffen* EG zu finden (vgl. Kap. 5.2.2). Eine *lose* EG folgt der Form des Datenraums nur grob und schließt fast alle Datenpunkte ein. Eine *straffe* EG folgt der Form des gegebenen Datenraums möglichst eng, dafür werden einige Datenpunkte exkludiert (vgl. Abb. 3.2). In Kapitel 5.2.2 wir die Optimierung der Straffheit (engl. *tightness*) nach [66] angewendet, um den Datenraum mittels OCSVM optimal zu beschreiben. Eine trainierte OCSVM benötigt nur einen Bruchteil aller vorhandenen Trainingsdaten als Stützvektoren, um die EG zu definieren. Dadurch stellt eine OCSVM einen effizienten Algorithmus dar, um neue Daten als *innerhalb* oder *außerhalb* der trainierten EG zu klassifizieren. Im Vergleich dazu benötigen Clusteringverfahren wie *k-nearest Neighbour* oder *Decision Trees* die gesamte Datenbasis, um eine Klassifizierung durchzuführen. Die im Vergleich dazu geringe Größe der EG einer trainierten OCSVM bietet damit einen signifikanten Vorteil für Anwendung direkt auf dem Fahrzeugsteuergerät. Sie kann dadurch im Anschluss an Modellberechnungen der KNN ausgeführt werden, um diese zu plausibilisieren (vgl. Kap. 5). Die Verwendung einer OCSVM bietet sich an, da Speicher und Rechenleistung auf den Steuergeräten knapp bemessen sind und die Modelle echtzeitfähig sein müssen [15].
Um die Modellberechnungen zu bewerten, wird eine Definition des wahrscheinlichsten

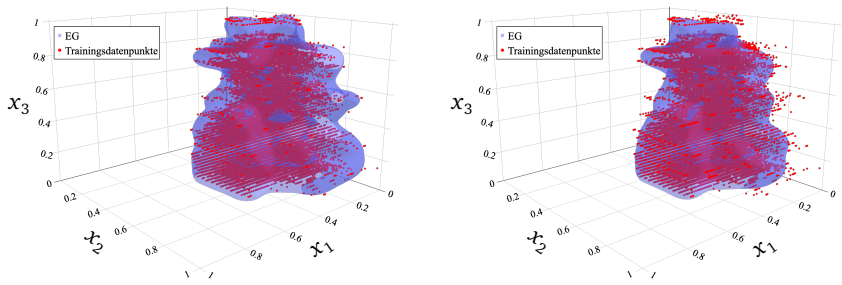


Abbildung 3.2: Vergleich zweier OCSVMs mit unterschiedlichen Hyperparametern ν und identischem γ. Diese führen jeweils zu einer anderen EG (blau) zur Beschreibung eines $\mathcal{D}_{\text{bw,sub}} \in \mathbb{R}^3$.
links: $\nu = 0.001$, $\gamma = 75$; rechts: $\nu = 0.01$, $\gamma = 75$.

Ausgangsdatenraums $\mathcal{D}_{\text{aus}}$ in Abhängigkeit der zugehörigen Eingangsdatenräume $\mathcal{D}_{\text{ein}}$ anhand einer OCSVM erstellt (vgl. Kap. 5).

3.3.3 Unterteilung in Unterdatenräume

Um die Genauigkeit der zulässigen Datenräume zu steigern, wurde in dieser Arbeit ein Methode entwickelt, die eine feingranulare Unterscheidungen des zulässigen Ein- und Ausgangsdatenraums ermöglicht. Einzelne Modellberechnungen können so mit einem kleineren zulässigen Bereich im Eingangsdatenraum (dem BRS vgl. Kap. 5.2.1) rückverknüpft werden. Dazu wurden Unterdatenräume $\mathcal{D}_{\text{aus,sub}}$ zur Unterteilung des zulässigen Ausgangsdatenraums $\mathcal{D}_{\text{aus}}$ erzeugt. Es wurden zwei verschiedene Verfahren zur Unterteilung dieser Datenräume verwendet (vgl. Kap. 5).
Den Unterdatenräumen $\mathcal{D}_{\text{aus,sub}}$ des zulässigen Ausgangsdatenraums $\mathcal{D}_{\text{aus}}$ können anhand der Trainingsdaten die zugehörigen Datenpunkte im zulässigen Eingangsdatenraum zugeordnet werden. Diese Menge an Datenpunkten ist eine Untermenge des zulässigen Eingangsdatenraums $\mathcal{D}_{\text{ein}}$ und wird als BRS $\mathcal{D}_{\text{bw,sub}}$ bezeichnet. Dieses Set wird durch alle gemessenen Eingangsvektoren $\boldsymbol{x}$ aus dem Eingangsdatenraum $\mathcal{D}_{\text{ein}}$ ($\boldsymbol{x} \in \mathcal{D}_{\text{ein}}$) definiert, welche den Zielvektoren $\boldsymbol{y}$ aus dem Unterdatenraum $\mathcal{D}^c_{\text{aus,sub}}$ zugeordnet werden können. Dabei ist $c \in (0, 1, 2, ...N)$ mit N die Anzahl der erzeugten Unterdatenräume. Es gilt also:

$$\mathcal{D}^c_{\text{bw,sub}} \subseteq \mathcal{D}_{\text{ein}} \tag{3.17}$$

$$\mathcal{D}^c_{\text{aus,sub}} \subseteq \mathcal{D}_{\text{aus}} \tag{3.18}$$

Die Anzahl N der Unterdatenräume $\mathcal{D}_{\text{sub}}$ entscheidet über die Granularität der Unterteilung des Datenraums. Sie wurde in [104] vorgestellt und wird in Kapitel 5 näher beschrieben. Kann eine Modellberechnung dem zulässigen Ausgangsdatenraum $\mathcal{D}_{\text{aus}}$ zugeordnet werden, kann sie auch einem zulässigen Unterdatenraum $\mathcal{D}^{c}_{\text{aus,sub}}$ zugeordnet werden. Diesem kann ein BRS $\mathcal{D}^{c}_{\text{bw,sub}}$ gegenübergestellt werden. Befindet sich der Eingangsvektor, der zur fraglichen Modellberechnung geführt hat, in diesem Set, so kann die Modellberechnung als plausibel eingestuft werden. Die Herleitung dazu beruht auf dem Lipschitz-Kriterium und wird in Kap. 5 ausgeführt.

3.3.4 Einflussbewertung der Trainingsdatenverteilung

Die Verteilung der Messdaten ist ausschlaggebend für die Generalisierung des Modells auf unbekannten Daten. Nur durch eine den realen Zusammenhängen entsprechende Verteilung der Trainingsdaten kann das Modell all Zusammenhänge in gleicher Qualität erlernen. Besteht ein Bias hinsichtlich einer oder mehrere Betriebspunkte wird das Modell diese tendenziell besser abbilden als die unterrepräsentierten Betriebspunkte. Daher ist ein *Big-Data*-Ansatz meist vielversprechend, da durch die schiere Masse an Daten zu allen möglichen Kombinationen die Wahrscheinlichkeit erhöht wird, dass das Modell alle Zusammenhänge abbilden kann. Auch bei einem *Big-Data*-Ansatz muss auf die zum Anwendungsfall passende Verteilung geachtet werden, um einen Bias in den Trainingsdaten gering zu halten [24], [45].
In Bezug auf das Training von KNN ist eine größere, und gleichmäßig verteilte Messdatenmenge von Vorteil. Dies hat sich in Untersuchungen zu dieser Arbeit in Bezug auf die Repräsentation des Gangzustandes in den Trainingsdaten gezeigt [29].
Es ist nicht möglich einen Datensatz ohne Bias zu erstellen, alleine durch die eingeschränkte Kenntnis aller möglichen Fahrsituationen und Wechselwirkungen im Antriebsstrang kann dies für diese Arbeit nicht angenommen werden.
Wie in Kapitel 2.1 erläutert, kann näherungsweise eine Gaußverteilung zur Beschreibung des Anwendungsfalls angenommen werden. Der Trainingsdatenraum wird mit Hilfe von OCSVMs eindeutig beschreiben. Auf Grundlage der Datenbasis werden die statistischen Momente berechnet und mit denen eine Gaußverteilung verglichen. In dem durch die EG definierten Datenraum, wird eine möglichst gleichverteilte Datenmenge angelegt, um alle Betriebspunkte gleichermaßen im Training zu repräsentieren. Für die Erstellung eines Datenraums mit uniform verteilten Datenpunkten kann ein Verfahren der statistischen Versuchsplanung angewendet werden. Für das Training von KNN empfehlen sich raumfüllende Verfahren wie S-optimale Versuchspläne oder das *Latin-Hypercube-Sampling* [45]. Beide Verfahren erzeugen im definierten Versuchsraum (innerhalb der EG) möglichst raumfüllend verteilte Datenpunkte. Damit sollen alle möglichen Parameterkombinationen gleich gewichtet werden.
Durch diese Verfahren können raumfüllend Datenpunkte mit maximalem minimalen Abstand zueinander innerhalb der mathematisch konvexen EG erstellt werden. Anwendung finden diese Pläne z.B. in der Motorapplikation zu Definition der anzufahrenden

Versuchspunkte, um bestmöglich ein Kennfeld zu untersuchen und abzudecken. Um die Trainingsdaten an die durch die raumfüllenden Versuchspläne vorgegebene Verteilung anzupassen, wird jedem Versuchsdatenpunkt der nächstgelegene Trainingsdatenpunkt zugeordnet. Dieses Vorgehen erzeugt eine dem ursprünglichen Versuchsplan angenäherte Verteilung realer Messdaten und eine Reduktion der Trainingsdatenmenge [24], [67]. Eine weiter Möglichkeit wäre die Verwendung komplett synthetisch erstellter Trainingsdaten als raumfüllende Versuchspunkte innerhalb der EG. Diese Methodik wurde in Testszenarien überprüft und ist nicht Teil dieser Arbeit. Da durch die zugrundeliegende Gaußverteilung der aufgezeichneten Messdaten eine Reduktion auf einen raumfüllenden Versuchsplan die Datenmenge reduziert, wird vor allem in Bereichen stark unterrepräsentierter Betriebspunkte weiter eine geringe Datenmenge verfügbar bleiben. Aus diesem Grund wurden in dieser Arbeit auf eine Reduktion und Anpassung der Trainingsdaten an einen raumfüllenden Versuchsplan verzichtet. Damit soll der bestmögliche Informationsgehalt der Datengrundlage für das Modelltraining verfügbar gemacht werden.

3.3.5 Einfluss der Ein- und Ausgangsdimensionen

Die Anzahl der Modelleingänge bestimmt die Dimension des Eingangsdatenraums, analog dazu wird die Dimension des Ausgangsdatenraums durch die Anzahl der Modellausgänge bestimmt. Diese Datenräume gilt es nach den oben beschriebenen Prinzipien mit einer EG zu umgeben, um die nötige Plausibilisierung (vgl. Kap. 5.4) der Modellausgänge durchführen zu können.
Für einen zweidimensionalen Datenraum kann diese EG noch in einem $2D$-Koordinatensystem zusammen mit den entsprechenden Datenpunkten dargestellt und überprüft werden. Bei der EG handelt es sich hierbei um eine Hyperebene im zweidimensionalen Raum mit beliebiger Form. Ähnlich gestaltet sich das Vorgehen für einen drei dimensionalen Datenraum. Auch hierfür können alle Datenpunkte und die EG in einem $3D$-Koordinatensystem dargestellt werden. Einzig die EG ist eine Hyperebene im dreidimensionalen Raum. Je nach Anordnung und Verteilung der Datenpunkte im Raum kann eine visuelle Beurteilung der Form und des Verlaufs der EG durch die Entwickler_innen bereits schwierig werden. Dies ist insbesondere der Fall, wenn Hinterschneidungen oder Hohlräume mit berücksichtigt werden müssen.
Für die Anwendung auf die *Fallstudien I & II* sind allerdings Datenräume höherer Dimension möglich. Für die Ein- und Ausgangsdatenräume gilt

$$\mathcal{D}_{\text{ein}} \subseteq \mathbb{R}^j \quad \forall j \in \mathbb{N}; \tag{3.19}$$

$$\mathcal{D}_{\text{aus}} \subseteq \mathbb{R}^k \quad \forall k \in \mathbb{N}. \tag{3.20}$$

Eine Darstellung in der vierten Dimension könnte mittels einer Farbskala stattfinden, allerdings entfällt trotz allem die Möglichkeit die EG passend darzustellen. Dieser Nachteil erfordert ein Verfahren um die Form der trainierten EG anhand von Kenn-

zahlen direkt zu bewerten. Dieses wird in Kapitel 5.2.2 beschrieben. Ein Vorteil davon ist, dass somit ein Training (und die zugehörige Optimierung) sowie die Auswahl der am besten geeigneten EG automatisiert durchgeführt werden kann. Die Dimensionen der Ein - und Ausgangsdatenräume beeinflussen also direkt die Art und Weise wie die zugehörige EG überprüft, bewertet und ausgewählt werden kann.
Ein weiterer direkter Einfluss ist die Höhe der Dimension bzw. die Menge an Vektoreinträgen der Ein- und Ausgangsvektoren. Studien bescheinigen dem Verfahren der SVMs und OCSVMs eine gute Genauigkeit für bis zu ca. 10-dimensionalen Datenräumen [68], [69]. Je höher jedoch die Dimension der zu verarbeitenden Daten, desto höher die Trainingszeit und desto empfindlicher reagieren die SVMs auf eine Änderung der Hyperparameter. Der sogenannte Fluch der Dimensionalität (engl. *curse of dimensionality*) sorgt für sehr dünn besetzte Datenräume in hohen Dimensionen und führt leicht zu einer Überanpassung. Ab dieser Obergrenze der Dimensionalität können die OCSVMs nicht mehr nutzbringend eingesetzt werden. Allerdings spielt diese Obergrenze im Zuge dieser Arbeit keine Rolle, da die maximale Anzahl j an Eingangssignalen (Dimension des Eingangsvektors $\boldsymbol{x}$) höchstens $j_{max} = 10$ erreicht. Für den Zielvektor $\boldsymbol{y}$ gilt $k_{max} = 3$.
Zur Überprüfung der EG können Kontrollpunkte erstellt werden, welche per Definition entweder *innerhalb*, *außerhalb* oder auf der Grenze der EG liegen sollen [68], [69]. Dies ermöglicht die punktuelle Überprüfung des Verlaufs der Hyperebene, kann jedoch nur mit einer endlichen Zahl an Kontrollpunkten, meist in physikalisch eindeutig zu beschreibenden Betriebspunkten durchgeführt werden. Diese Methode bietet eine Möglichkeit zur Überprüfung hoch-dimensionaler Datenräume.

4 Untersuchung geeigneter ML-Modelle zur Substitution der PE-Modelle

Im Rahmen dieser Arbeit werden zwei verschiedene neuronale Netzwerkarchitekturen untersucht und bewertet, ob sie sich zur Substitution der PE-Modelle eignen. Das vorliegende Kapitel gibt einen Überblick über mögliche Architekturen und begründet die Auswahl von FNN und RNN. Diese werden in zwei Fallstudien detailliert überprüft. Dazu erfolgt die Erstellung geeigneter Trainingsabläufe und Hyperparameter. Zur Ausführung des Trainings wird eine ML-Pipeline entwickelt, um die Modellerstellung durchgängig automatisiert durchführen zu können. Durch die Automatisierung des Modelltrainings wird eine Methodik entwickelt, die es ermöglicht, die zur modellbasierten Regelung benötigten Modelle an sich ändernde Hard- und Softwarestände anzupassen. Gleichzeitig kann der manuelle Aufwand zur Kalibrierung von Appilkationsparametern reduziert werden. Das Ziel ist es mit den erzeugten ML-Modellen eine höhere Modellgüte bei einer, mit den PE-Modellen vergleichbaren Ressourcenauslastung zu erreichen.

Im Automobilbereich sind die meisten Rechenprozesse auf den Steuergeräten dem herrschenden Rechentakt unterworfen. Für diese Prozesse muss Echtzeitfähigkeit gewährleistet werden können [15]. Diese Anforderung kann die zulässige Rechenzeit limitieren, da das Modell innerhalb einer vorgegebenen Taktzeit ausgeführt werden und die Modellausgänge bereitstellen muss. Daher kommen auf den Steuergeräten oft vereinfachte Ersatzmodelle zum Einsatz, welche die Anforderungen an die Modellgüte zufriedenstellend erfüllen. Gleichzeitig können diese speicherplatzschonend so*wie in* ausreichend geringer Rechenzeit ausgeführt werden. Die benötigte Datengrundlage und deren Speicherplatzbedarf, haben ebenfalls einen großen Einfluss. Um ML-Modelle im Fahrzeug verwenden zu können, wird ein möglichst geringer Ressourcenverbrauch auf der späteren Zielhardware angestrebt. Für den gegebenen Anwendungsfall des vorausschauenden PKW-TM ist dies das Antriebsstrangsteuergerät CPC. Um ein Modell auf diesem Mikrocontroller (μC) auszuführen, muss die Logik in *C-Code* vorliegen und echtzeitfähig sein. Der *C-Code* wird mittels einer standardisierten Abfolge von Softwarewerkzeugen, der sogenannten Tool-Kette (engl. *toolchain*), in Steuergerätecode umgewandelt. Nur so kann die Berechnung der Zielgrößen im vorgegebenen Steuergerätetakt stattfinden. Diese Bedingungen müssen bei der Wahl der Modellierungsmethodik berücksichtigt werden und sprechen für den Einsatz von ML-Methoden zur Bildung von datenbasierten Modellen. Grundlegend gilt, ein lineares Modell sollte immer die erste Wahl sein, um eine datenbasierte Modellierung durchzuführen. Die

Tabelle 4.1: Gegenüberstellung von ML-Modellen und PE-Modellen

Merkmale	**PE-Modelle**	**ML-Modelle**
Komplexe Zusammenhänge	Expertenwissen nötig, ggf. komplexe u. große Modelle	datenbasiert, Modellgröße ist nicht abhängig von Komplexität
Anzahl der Modellparameter	tendenziell geringer - sonst nicht überschaubar	groß und nicht mit physikalischen Werten in Verbindung
Skalierbarkeit	Komplexität begrenzt durch Verständnis der physikalischen Zusammenhänge	fast unbegrenzt skalierbar
Inkrementelles Lernen	kann nur definierten Anwendungsfall lösen	kann mittels aktueller Daten nachtrainiert werden
Datenmenge/ Messungen	meist nur Validierungsmessung nötig	große Datenmengen nötig
Genauigkeit	abhängig von Systemverständnis und Modellierungsaufwand	abhängig von Daten, Modellgröße, Training
Erklärbarkeit	physikalische Zusammenhänge bekannt und mathematisch beschrieben	*black-box* Modell, meist eingeschränkte Erklärbarkeit
Generalisierung	gut im Definitionsbereich	gut im Datenraum der Trainingsdaten

geringe Komplexität und Parameterzahl machen sie zu beliebten Modellen, die sogar geringe Nichtlinearitäten abbilden können [24]. Liegen nur wenige und verrauschte Messdaten vor, können lineare Modelle eine ausreichend gute Modellgüte liefern. Allerdings können nichtlineare Modelle deutliche Genauigkeits- und Leistungsvorteile mit sich bringen. Zudem sind sie in der Lage, deutlich komplexere Zusammenhänge

effizienter abzubilden [24]. Mittels datenbasierter Modellbildung können ML-Modelle erstellt werden, welche den in Kapitel 3 beschriebene Lösungsraum umfassend und effizient abbilden können. Da dies eine Vielzahl von möglichen Modellparameterkombinationen hervorbringt, muss eine große Zahl an Modellen erstellt und bewertet werden. Dafür wird ein automatisiertes Vorgehen eingeführt. Die Auslagerung der Datensammlung, des Trainings und der Modellbewertung auf eine skalierbare Plattform (wie eine Cloud-Instanz) bietet ein variables Entwicklungsumfeld für diese Arbeit. Dadurch ergeben sich weitere Vorteile. Die Cloud-Anbindung der aktuellen Fahrzeuggeneration wird genutzt, um neue Messdaten direkt in den bereits vorhandenen Trainingsdatensatz auf der Cloud einzuspeisen. Nach dem Training mit dem aktualisierten Datensatz werden die Modelle über diese Schnittstelle auf das Fahrzeug übertragen. So ist es möglich, ein Update der bestehenden Modelle durchzuführen, sobald eine neue Datengrundlage und Modelle mit höherer Modellgüte vorliegen. Die Verwendung von datenbasierten Modellen bietet die Möglichkeiten, den Entwicklungsprozess weiter zu automatisieren und die Modellqualität anhand eindeutiger Kennzahlen zu bewerten. Für diese Arbeit werden KNN untersucht. Diese werden basierend auf der gegebenen Messdatenbasis modelliert. KNN sind eine Unterklasse der ML-Modelle (vgl. Kap. 2.2). Sie bieten eine Vielzahl verschiedener Architekturen für die unterschiedlichsten Anwendungsfälle (vgl. Kap. 4.1.2). Diese Architekturen können in ihrer Komplexität beliebig skaliert werden. Die variablen Hyperparameter (wie Modelltiefe n, Aktivierungs- und Zielfunktion) ergeben eine immense Vielfalt an verfügbaren Modellstrukturen. Diese ML-Modelle gründen auf einer breiten wissenschaftlichen Basis, ermöglichen eine einfache Automatisierbarkeit der Modellbildung und bieten die Möglichkeit komplexe Zusammenhänge in verhältnismäßig kleinen Modellen abzubilden. Im Modellbildungsprozess, dem sogenannten Training, werden die Modellparameter in einem iterativen Prozess optimiert. Folgenden Vorteile ergeben sich aus dem Einsatz von KNN-Modellen:

- komplexe oder unbekannte Zusammenhänge können ohne manuelle Kalibrierung der Modellparameter modelliert bzw. trainiert werden,
- das Training verschiedener Modelle und die Auswahl des besten Modelle kann automatisiert werden,
- Potenzial zur Verringerung der Modellgröße und des Ressourcenverbrauchs gegenüber PE-Modellen (vgl. Tab. 4.1),
- Objektive Bewertung der Modellgüte auf Basis der Testdaten,
- kompatibel mit *Continuous Integration / Continuous Deployment* (CI/CD) Pipelines und *Big-Data*-Ansätzen,
- hohe Genauigkeit bei ausreichender Trainingsdauer und Datenbasis [24],
- hohe Informationsdichte, bezogen auf die Parameterzahl [24],
- geringe Sensitivität auf Rauschen, da die Parameteridentifikation über alle Trainingsdaten stattfindet [24],
- Inferenz kann im Vergleich zu anderen ML-Modellen effizient und schnell durch geführt werden [24].

Alle untersuchten Modelle werden in einem automatisierten Workflow auf einer Cloud-Instanz modelliert und evaluiert. Das Update auf dem Fahrzeug schließt sich daran an. In Kapitel 4.1 werden verschiedene Netzarchitekturen und geeignete Einsatzbereiche verglichen, bewertet und ausgewählt. Darauf folgt in Kapitel 4.2 die Beschreibung des Trainings. Die Auswahl der passenden Lernverfahren, Trainingsdaten und Zielgrößen steht im Vordergrund. Anschließend werden in Kapitel 4.3 die in Kapitel 3 entwickelten Methoden zur Eingrenzung und Analyse von Trainingsdatenräumen angewendet, um die Modellgüte der trainierten KNN zu optimieren. Dazu wird die vorhandene Datenbasis analysiert, um die Datenverteilung im Datenraum zu bewerten. Zusätzlich wird ein Verfahren zur synthetischen Datenerzeugung vorgestellt. Ergebnisse der erzeugten ML-Modelle werden in Kapitel 4.4 diskutiert. Hierbei werden die zwei verschiedenen Fallstudien betrachtet (vgl. Kap. 1.2). Zudem werden die Vorzüge und die Notwendigkeit einer Plausibilisierung der Modellberechnungen erörtert. Diese wird im Anschluss in Kapitel 5 eingeführt.

4.1 Einsatzbereiche und Varianten künstlicher neuronaler Netzwerke

KNN sind dem Feld des ML zuzuordnen (vgl. Kap 2.2.4). Dieses stellt eine Unterklasse der KI dar. Bei KNN handelt es sich um datenbasierte Modelle. Inspiriert durch die Verknüpfung der Neuronen des menschlichen Gehirns werden künstliche Neuronen bzw. Knoten des KNN erstellt. Die Reizübertragung am synaptischen Spalt wird als Aktivierungsfunktion des KNN abgebildet. Durch die Verkettung einfacher mathematischer Funktionen können damit KNN beschrieben werden. Die Logik wird zu einem Großteil durch Matrizenoperationen abgebildet. Die Aktivierung der künstlichen Neuronen ergibt sich aus der Summe aller Eingangswerte eines solchen Knotens. Diese Summanden werden vorab mit individuellen Gewichtungsparametern multipliziert (vgl. Abb. 4.1). Durch die Verwendung stetig differenzierbarer Funktionen kann das Gradienten-Abstiegsverfahren für das sogenannte Training (vgl. Kap. 4.2) eingesetzt werden, in welchem die Modellparameter optimiert werden. In der Praxis unterscheiden sich KNN und ihre künstlichen Neuronen in vielen Punkten grundlegend von ihrem Vorbild aus der Natur. Zum einen stellt die schichtweise Verknüpfung von einfachen mathematischen Einheiten eine starke Vereinfachung der Vernetzungen im menschlichen Gehirn dar, zum anderen ist die Idee der sprunghaften Aktivierung bei Erreichung eines Aktionspotenzials durch die weit verbreitete Verwendung von ReLUs (vgl. Kap. 4.1.4) einer teilweise stetigen, unbegrenzt wachsenden Aktivierung gewichen. Viele weitere Unterschiede zu dieser starken Vereinfachung des menschlichen Gehirns sind in der Fachliteratur dargelegt [70]–[72].

Da die Ressourcen auf dem Fahrzeugsteuergerät stark limitiert sind (wenige Kilobyte für eine Funktion), werden in dieser Arbeit KNN mit möglichst hoher Modellgüte und geringer Parameterzahl entwickelt. Ziel ist ein optimaler Kompromiss aus Genauigkeit und Ressourcenbedarf. Dafür wird eine automatisierter Trainingsablauf in Form einer

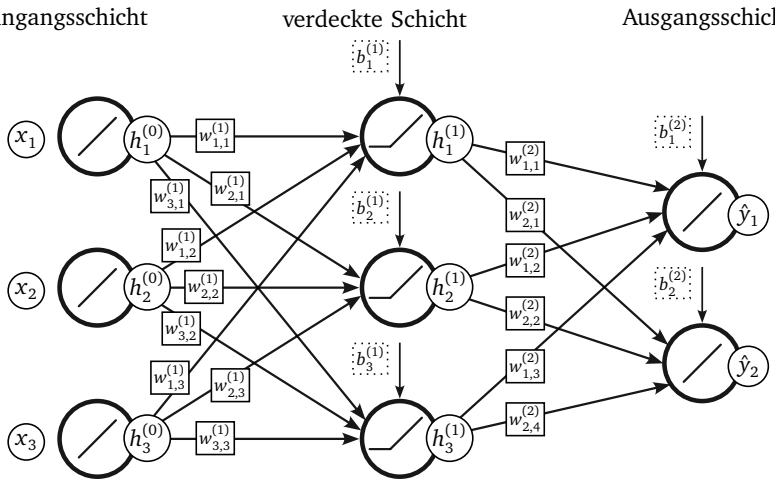


Abbildung 4.1: Einfache Darstellung eines MLP mit einer verdeckten Schicht.

ML-Pipeline aufgebaut. Die in Kapitel 3 beschriebene Datenerhebung und Aufbereitung sind Bestandteile dieser Automatisierung. In diesem Unterkapitel wird die Eignung unterschiedlicher Netzwerkarchitekturen untersucht und bewertet, um daraus die am besten geeignete Modellkonfiguration abzuleiten. Hierfür werden die nötigen Kriterien zur Anwendung des KNN definiert. Darauf aufbauend werden Bewertungskriterien eingeführt, die es erlauben, die verschiedenen Optionen miteinander zu vergleichen. Hierfür wird unter anderem die Modellgüte der trainierten Modelle bewertet, deren Definition wird in Kapitel 4.2.1 beschrieben. Anhand dieser Kriterien und Kennzahlen kann ein Vergleich der Architekturen durchgeführt und eine Auswahl getroffen werden.

4.1.1 Anwendungskriterien für datenbasierte Modelle

Für das prädiktive TM wurden in [16] PE-Modelle verwendet, um trotz begrenzter Ressourcen auf dem Steuergerät eine modellbasierte Regelung implementieren zu können. Um einen technischen Mehrwert zu erzielen, müssen KNN trainiert werden, welche die PE-Modelle in Güte, Genauigkeit und Ressourceneffizienz ersetzen können. Die vorhandenen PE-Modelle in Form von *Matlab Simulink*-Modellen werden im Rahmen dieser Arbeit in *Python*-Codes überführt. Dadurch wird eine direkte Vergleichbarkeit zu den entwickelten KNN geschaffen (vgl. Kap. 4.4). Die Grundlage dieser Modelle bilden bekannte physikalische Zusammenhänge. Die verschiedenen Modellparameter

Tabelle 4.2: Vergleich verschiedener Modellierungsmethodiken nach [24]

Methodik	Parameterzahl	Trainingsaufwand	Generalisierung
KNN	−	−−	++
Decision Tree	−−	−	+
SVM	+	−	+
Polynommodelle	+	+	+
k-nearest Neighbour	−−	+	−
Gaußprozessmodelle	+	−	++
Lookup-Table	−−	++	−−
Markov-Ketten-Modell	−−	−	−

können jedoch nicht mehr eindeutig ihren physikalischen Einflussgrößen zugeordnet werden, da dieser Ansatz die Modelle auf ein Ersatzmodell reduziert und keine direkte Entsprechung aller Modellparameter zu den zugrunde liegenden Gleichungen mehr möglich ist. Anhand der Modellparameter kann das Modellverhalten manuell kalibriert werden, um die in Messungen vorliegenden Ausgänge zu erhalten. Diese Modelle liefern ein deterministisches Verhalten für alle Eingangsgrößen innerhalb des Definitionsbereichs. Je nach zulässiger Modellgröße können so einfache oder komplexe Ersatzmodelle erzeugt werden. Voraussetzung ist die umfassende Kenntnis der zu modellierenden Zusammenhänge. Im Vergleich dazu haben datenbasierte Modelle den Vorteil, dass sie nicht vollständig erfasste und komplexe Zusammenhänge durch das Training mit einer ausreichend diversen Datengrundlage *erlernen* können. Dies führt unmittelbar zu einem wesentlichen Nachteil, da ein sogenanntes *black-box* Modell entsteht, welches keine Rückschlüsse auf die zugrunde liegenden physikalischen Zusammenhänge gewährt. Für das Training der KNN spielt neben der Verfügbarkeit der einzelnen Signale in einer Messdatei die Menge der verschiedenen Messungen eine wichtige Rolle. Nur mit einer ausreichend großen, gleichmäßig im Versuchsraum verteilten Datenbasis (vgl. Kap. 3) kann ein repräsentatives Modell trainiert werden [73], [74]. Die Abtastrate der Messungen limitiert zusätzlich die erreichbare Modellgüte (vgl. Kap. 4.4.3). Datenbasierte Ansätze ermöglichen in vielen Fällen die Erstellung von kleinen und effizienten Modellen. Extrapolationen über die Grenzen der verwendeten Trainingsdaten hinaus sind in der Anwendung zu vermeiden, da nicht gewährleistet werden kann, dass das Modell für Eingänge außerhalb des Trainingsdatenraums gut generalisiert [24], [25], [75], [76]. Ein nach dem Training fixiertes Modell (konstanter Parameter) liefert für identische Eingangswertkombinationen bei jeder Ausführung deckungsgleiche Ergebnisse. KNN können nicht als deterministisch im Sinn der Interpolation zwischen bekannten Testdatenpunkten und unbekannten Modelleingängen

Tabelle 4.3: Anwendungskriterien von PE-Modellen und ML-Modellen

Eigenschaften	PE-Modell	ML-Modell
vollkommenes Systemverständnis nötig	ja	nein
große Datengrundlage nötig	nein	ja
deterministisch	ja	nein
automatisiertes Erstellen u. Kalibrieren	schwer	ja
automat. Nachtrainieren	nein	ja

betrachtet werden. Die Generalisierbarkeit kann grundsätzlich mit einer ausreichend homogen verteilten Testdatenmenge überprüft werden. Absolute Sicherheit, dass eine physikalische Funktion $\boldsymbol{y}_p = f_p(\boldsymbol{x})$ für alle möglichen Eingangsgrößen korrekt abgebildet wird, ist jedoch nicht zu erreichen. Dafür müssten im Training alle theoretisch möglichen Eingangsdatenpunkte im Voraus verwendet werden. Dies ist in der Anwendung unrealistisch und widerspricht dem Vorteil der datenbasierten Modelle. Dieser ermöglicht es, unbekannte Zusammenhänge aus gegebenen Beobachtungen dieser Vorgänge zu erlernen. Zudem würde ein stark überangepasstes Modell erstellt werden, welches schlechte Generalisierungsqualitäten auf unbekannten Daten hätte.
Um die Robustheit der Modell abzusichern, kann die Stetigkeit bestimmter Netzwerkarchitekturen in den Grenzen der verwendeten Trainingsdaten auch für unbekannte Eingangsgrößen bewiesen werden. Diese Lipschitz-Stetigkeit kann beispielsweise für FNN hergeleitet werden [77]. Die Stetigkeit betrifft den Zusammenhang zwischen Eingangs- und Ausgangsgrößen. Kleine Änderungen in den Eingangswerten führen zu kleinen Änderungen in den Ausgangswerten und somit zu einem stetigen Verhalten. Dies ist die Grundlage für die in Kapitel 5 eingeführte RPA von FNN-Berechnungen und wird dort näher beschrieben.
Dieser Arbeit liegt die Forderung nach ressourcenschonenden Modellen zugrunde, welche automatisiert trainiert werden können. Bei Bedarf sollen sie mit einer neuen Datengrundlage automatisiert nachtrainiert werden können. Die Möglichkeit, gewisse Modelle als Lipschitz-stetig zu definieren und deren Berechnungen zu plausibilisieren (vgl. Kap. 5), lässt die Wahl auf FNN fallen. Für RNN kann eine Lipschitz-Stetigkeit mit erhöhtem Aufwand ebenfalls erzeugt werden [78]. Tabelle 4.3 fasst die genannten Fakten zusammen. In dieser Gegenüberstellung weisen datenbasierte Modelle zur Erfüllung der wissenschaftliche Zielstellung dieser Arbeit einige Vorteile gegenüber manuell kalibrierten, PE-Modellen auf.

Tabelle 4.4: Übersicht über eine Auswahl verschiedener KNN Architekturen. Zusammenstellung nach [25].

KNN	Parameterzahl	Trainingsaufwand	Anwendungsbeispiele
FNN	−−	−	Regression, Klassifikation
RNN	+	+	Regression, Vorhersagemodelle
NARX	++	++	Zeitreihenvorhersage
LSTM	+(+)	+(+)	NLP
CNN	++	++	Bilderkennung
GAN	++	++	Deep Fakes
Auto-Encoder	++	++	Regression, Klassifikator

4.1.2 Übersicht und Auswahl der Netzwerkarchitekturen

Die Anzahl der verdeckten Schichten eines KNN definiert die Tiefe des Modells. Durch sogenanntes tiefes Lernen (engl. *deep learning*) können komplexe Zusammenhänge und zeitliche Abhängigkeiten gelernt und durch tiefe KNN repräsentiert werden. Von *deep learning* spricht man, wenn mehr als eine verdeckte Schicht existiert. Ein FNN mit mindestens einer verdeckten Schicht wird auch als mehrschichtiges Perzeptron (engl. *Multilayer Perceptron* (MLP)) bezeichnet. Beide Begriffe werden äquivalent verwendet [25]. Ein FNN definiert eine Abbildung $\boldsymbol{y} = f(\boldsymbol{x}, \theta)$ eines Eingangsvektors $\boldsymbol{x}$ auf einen Vektor aus Zielgrößen $\boldsymbol{y}$. Dafür werden die Modellparameter θ im Training optimiert, um die gewünschte Näherung zu erhalten [25]. Je nach Anwendungsfall stehen unterschiedliche Netzarchitekturen zur Verfügung (vgl. Tabelle 4.4). Schon ein MLP mit einer verdeckten Schicht ist ein universeller Funktionsapproximator [24, S. 251]. Beachtenswert ist, dass das MLP eine multidimensionale Funktion alleine mittels einer Kombination mehrerer, eindimensionaler und standardisierter (Aktivierungs-)Funktionen näherungsweise beschreiben kann. Die Genauigkeit dieser Näherung kann durch eine Erhöhung der Neuronenzahl verbessert werden. Dabei gilt, dass mehrere verdeckte Schichten das KNN performanter und komplexer machen. Müssen für ein MLP mit nur einer Schicht noch eine Vielzahl an Neuronen verwendet werden, so kann durch die Erweiterung um weitere Schichten die gesamte Parameterzahl verringert und die Genauigkeit gesteigert werden. Die Generalisierbarkeit wird durch eine höhere Anzahl an verdeckten Schichten und aufgrund weniger Neuronen pro Schicht verbessert [24], [25]. Die Eigenschaft als universeller Approximator führt jedoch nicht dazu, dass die approximierte Funktion auch gut auf unbekannten Daten generalisiert. Um dies zu erreichen, müssen im Training verschiedenen Aspekte wie die Beschaffenheit der Datengrundlage, *early stopping*, Methoden zur Regularisierung etc. beachtet werden, um eine Über- oder Unteranpassung zu vermeiden (vgl. Kap. 3.3.4 & 4.2.4).

Im Vergleich zu FNN besitzen RNN eine Rückkopplung. Dabei handelt es sich um eine Rückführung bereits berechneter Werte für die Berechnung des nachfolgenden Ausgangswertes. Man unterscheidet zwischen interner und externer Rückführung der Ein- und Ausgänge. RNN verwenden historische Informationen aus zuvor berechneten Ergebnissen als Eingang zur Berechnung des aktuellen Zeitschritts. Dadurch eignen sie sich zur Modellierung von Zeitabhängigkeiten. Diese Funktion fehlt FNN. Dies führt zu einer höheren Anzahl an Netzparametern, abhängig von der Länge der historischen Information und deren Dimension. Die Berücksichtigung historischer Informationen ist sowohl für FNN als auch für RNN möglich. Für FNN muss dazu der Eingangsvektor im *feature engineering* erweitert werden (vgl. Kap. 3.2.3). Für RNN fällt die Datenaufbereitung im Allgemeinen umfangreicher aus. Ein Grund hierfür ist, dass anstatt einzelner Zeitpunkte pro Zeitschritt jeweils Zeitfenster einer vorher zu definierenden Länge (engl. *window*) in das Modell eingespeist werden. Alle beschriebenen Maßnahmen erweitern zwangsläufig die Anzahl der Eingangswerte und somit die gesamte Zahl der Netzwerkparameter.
Für die Beschreibung weiterer Netzwerkarchitekturen wie *Generative Adversarial Network*s (GANs), welche auf der Fälscher-und-Polizisten-Methodik basieren um Muster, Regeln oder Datenanordnungen zu erlernen, GRUs, Auto-Encodern und *Convolutional Neural Network*s (CNNs) wird auf die Literatur verwiesen [25], [26]. Tabelle 4.4 gibt einen Überblick über eine Auswahl an KNN und vergleicht deren Parameterzahlen, Trainingsaufwände und Anwendungsgebiete.
Neben klassischen RNN (vgl. Abb.: 4.2) gibt es verschiedene weitere rekurrente Architekturen. Dazu gehören *Nonlinear Autoregressive Network with Exogenous Inputs* (NARX)-Modelle, welche eine Rückführung des vorher berechneten Modellausgangs zur Berechnung des aktuellen Zeitschritts als zusätzlichen Eingang nutzen. Dabei wird das Modell in der sogenannten *open-loop* Konfiguration trainiert. Die Inferenz wird in der *closed-loop* Konfiguration durchgeführt. Dies ermöglicht eine Bereitstellung der genauen historischen Modellausgänge im Training, anstatt mit den Modellschätzungen zu trainieren. Diese kommen nur in der Anwendung des Modells in der *closed-loop* Konfiguration zur Anwendung. Die *open-loop* Konfiguration bietet den Vorteil, dass die NARX-Modelle wie FNN trainiert werden können. Dabei vergrößert sich lediglich die Anzahl der Eingangsparameter um die Anzahl zurückgeführter Ausgangsparameter. Im Rahmen dieser Arbeit hat sich die Verwendung von NARX-Modellen als nicht geeignet erwiesen, da sie nach dem Training bei Inferenz mit ADAS-Horizontdaten im Vergleich zu FNN und RNN keine zufriedenstellende Genauigkeit liefern konnten.
LSTM stellen eine Weiterentwicklung der klassischen RNN dar, vorgestellt von Hochreiter und Schmidhuber im Jahr 1997. Diese Netzwerkarchitektur ermöglicht es sowohl, Lang- als auch Kurzzeitabhängigkeiten zu erlernen. Bis dahin stellte dies einen Schwachpunkt der RNN dar, da auf Grund des *vanishing-gradient*-Problems das Erlernen komplexer Zusammenhänge zwischen weit entfernten Zeitschritten schwierig war. LSTM eignen sich daher besonders für die Sprachverarbeitung (engl. *natural language processing*) sowie für andere, zeitreihenabhängige Probleme. Sie sind *state-of-the-art* in

moderner Sprach- und Textverarbeitungssoftware, beispielsweise auf Mobiltelefonen. Um das am besten geeignete Modelle für eine automatisierte Modellierung auf Basis von Fahrzeugmessdaten zu erstellen, werden verschiedene Netzwerkarchitekturen für den Anwendungsfall im vorausschauenden TM untersucht (vgl. Kap. 1.2). Um die vorhanden PE-Modelle zu ersetzen, müssen datenbasierte Regressionsmodelle verwendet werden. Je nach Komplexität des zu modellierenden Zusammenhangs eignen sich unterschiedliche Modelle. Nach Goodfellow, Bengio und Courville empfiehlt es sich, mit einem möglichst einfachen Modell zu starten und dann inkrementell die Anzahl der Modellparameter und die Komplexität der Architektur zu steigern, bis die gewünschte Modellgüte und -größe erreicht ist. Dazu werden zwei Fallstudien (vgl. Kap. 1.2.3) durchgeführt. In diesen werden ein einfaches FNN sowie ein komplexeres RNN untersucht.
Anhand der gegenübergestellten Vor- und Nachteile der einzelnen Netzwerkarchitekturen wurden für die Fallstudien (vgl. Kap. 1.2.3) zwei Architekturen ausgewählt.

- **FNN:** Durch ihre einfache Architektur können Modelle mit vergleichsweise kleiner Parameterzahl erstellt werden. Die Betrachtung einzelner, zeitunabhängiger Datenpunkte ermöglicht es, den Trainingsdatensatz als Punktewolke mit OCSVMs zu beschreiben und in Unterdatenräume $\mathcal{D}_{\text{sub}}$ zu unterteilen (vgl. Kap. 3). Damit wird eine genaue Beschreibung der Zusammensetzung und Verteilung der Daten im Datenraum durchgeführt. Diese Beschreibung ist essentiell für die Entwicklung der in Kapitel 5 eingeführten RPA der Modellausgänge.
  Fallen in den erstellten Unterräumen Defizite auf, können gezielt zusätzliche Daten aufgezeichnet und der Datenraum damit aufgefüllt werden. Eine Gleichverteilung der Trainingsdaten im Datenraum wird angestrebt.
- **RNN:** Durch interne oder externe Rückführungen können diese Netzwerke zeitliche Abhängigkeiten erlernen. Dies ist für die Prädiktion einer zusammenhängenden Fahrstrecke von Vorteil. Die ADAS-Daten liefern in der Anwendung einen zusammenhängenden Vorausschauvektor $\boldsymbol{x}_{\text{HRC}}$. RNN können Zusammenhänge zwischen Streckensteigung und Geschwindigkeitslimits aus der Vergangenheit für die Berechnung der Motorleistung berücksichtigen. Dabei werden mit einem Anstieg in der Parameterzahl und dadurch höherer Modellkomplexität potenziell genauere Modelle möglich. Die Architektur der RNN ermöglicht es, zeitliche Abhängigkeiten zu berücksichtigen. Zusätzlich kann die Anzahl der Zeitschritte, die das RNN in die Zukunft prädizieren soll, variiert werden. Dadurch können sowohl der Wert für den aktuellen Zeitpunkt des Eingangsvektor sowie weitere, in der Zukunft liegende Zeitpunkte ermittelt werden (engl. *multi-step prediction*). Je mehr zukünftige Zeitschritte berechnet werden sollen, desto ungenauer und schwieriger wird die Vorhersage. Für das *Fallbeispiel II* wird keine *multi-step-prediction* durchgeführt, sondern der zum Zeitschritt des Eingangsvektors passende Ausgangsvektor berechnet. Es findet also keine Prädiktion in die Zukunft statt. Diese Informationen werden durch den Vorausschau - Vektor $\boldsymbol{x}_{\text{HRC}}$

Tabelle 4.5: Hyperparameter Sets für die Gitterstudien mit definierten Intervallen und Grenzen

Parameter	Wertebereich
Anzahl Epochen n_e	$n_e \in \{100, 200, \ldots, 500\}$
Initiale Lernrate ϵ_0	$\epsilon_0 \in 10e^{-3}\,\{0.1, 0.25, 0.5, 1, 2.5, 5, 10\}$
Größe *mini-batch* n_{mb}	$n_{mb} \in \{32, 64, \ldots, 1028\}$
Anzahl Neuronen pro Schicht n_n	$n_n \in \{2, 4, 8, \ldots, 512\}$
Anzahl verdeckter Schichten n_h	$n_h \in \{2, 4, 8, \ldots, 64\}$
Aktivierungsfunktion $\Phi(\cdot)$	ReLU, Sigmoid, tanh
Optimierungsalgorithmus	*Adam* oder *RMSprop*
Fehlermaß	MSE
Initialisierung nach	*Glorot* oder *He*
Anzahl Eingangsparameter j	$j \in \{3, 4, \ldots 10\}$
Anzahl Ausgangsparameter k	$k \in \{1, 3\}$
Anteil Drop-Out Neuronen	0 %; 25 %; 75 %
Regularisierung via Strafterm	l_1- oder l_2-Norm
Early stopping	immer

aus dem ADAS-Horizont geliefert (vgl. Kap. 2.2.3).

Kriterien, die im *hyperparameter tuning* variiert werden, sind in Tabelle 4.5 dargestellt. Die aufgeführten Hyperparameter finden für beide Modellstrukturen Anwendung. Für die RNN werden weitere Hyperparameter wie die Anzahl der Zeitschritte n_{rek} der zurückgeführten Werte benötigt. Diese werden im folgenden Kapitel 4.1.3 dargestellt.

4.1.3 Vergleich von vorwärts- und rückgekoppelten KNN

Um die ML-Modelle auf dem Fahrzeugsteuergerät in einem vorgegebenen Rechentakt auszuführen, muss das in *C-Code* übersetzte Modell möglichst klein sein, um wenig Speicherplatz zu belegen. Zudem muss es im vorgegebenen Raster von 10 Hz des Steuergerätes ausgeführt werden können. Vorwärtsgekoppelte Modelle können sehr groß werden, um komplexe Probleme abzubilden. Die Vorteile dieser Modellarchitektur werden in der Anwendung mit den auftretenden Nachteilen verglichen. In *Fallstudie II* werden RNN untersucht. Diese können Abhängigkeiten aus vorangegangenen Modellein- und -ausgängen lernen.

$$\mathbf{W} = \begin{pmatrix} w_{1,1} & \cdots & w_{1,j} \\ \vdots & \ddots & \vdots \\ w_{n_n,1} & \cdots & w_{n_n,j} \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} b_1 \\ \vdots \\ b_{n_n} \end{pmatrix} \tag{4.1}$$

Gleichung 4.1 stellt die allgemeine Form der Gewichtsmatrix $\mathbf{W} = w_{n_n,j}$ und des Biasvektors $\mathbf{b}$ für eine verdeckte Schicht mit n_n Knoten (Neuronen) eines MLP dar. Jedes Neuron kann als standardisierte Einheit einer Schicht gesehen werden, welche parallel zu weiteren Neuronen agiert. Jede dieser Einheiten führt eine Vektor-zu-Skalar-Operation durch [25, Kap. 6, S. 165]. Die erste verdeckte Schicht eines tiefen FNN errechnet aus dem Eingangsvektor $\boldsymbol{x}$, der Gewichtsmatrix $\mathbf{W}^{(1)}$ und dem Biasvektor $b^{(1)}$ einen Ausgangsvektor $\mathbf{h}^{(1)}$ mit

$$\mathbf{h}^{(1)} = \Phi(\mathbf{W}^{(1)T}\boldsymbol{x} + b^{(1)}). \tag{4.2}$$

Der Biasvektor $\boldsymbol{b}$ kann in die Gewichtsmatrix $\boldsymbol{W}$ integriert werden, indem dem Eingangsvektor ein konstanter Parameter $x_0 = 1$ angefügt wird:

$$\mathbf{W} = \begin{pmatrix} w_{1,0} & w_{1,1} & \cdots & w_{1,j} \\ w_{2,0} & w_{2,1} & \cdots & w_{2,j} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n_n,0} & w_{n_n,1} & \cdots & w_{n_n,j} \end{pmatrix}, \quad \boldsymbol{x} = \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_j \end{pmatrix}, \quad x_0 = 1. \tag{4.3}$$

Der Biaswert $\boldsymbol{b}$ wird für jeden der n_n Knoten der verdeckten Schicht als $w_{n_n,0}$ in die Gewichtsmatrix $\boldsymbol{W}$ übernommen. Analog wird dies für die Ausgangsvektoren $h^{(r)}$ mit $h_0^{(r)} = 1$ angewendet. Die allgemeine Gleichung für eine beliebige verdeckte Schicht lautet

$$\mathbf{h}^{(r)} = \Phi^{(r)}(\mathbf{W}^{(r)T}\mathbf{h}^{(r-1)}), \quad r \in \{1, \ldots, n_h\}, \quad \mathbf{h}^{(0)} = \boldsymbol{x}. \tag{4.4}$$

Der Ausgangsvektor der vorangegangenen verdeckten Schicht ist beschrieben durch $\mathbf{h}^{(r-1)}$. Dieser setzt sich zusammen aus den Transformationen der einzelnen Knoten, welche mittels Skalarprodukt und Aktivierungsfunktion die Vektor-zu-Skalar-Transformation durchführen. Ein dreischichtiges Perzeptron (engl. *three layer perceptron*) hat $n_h = 1$ verdeckte Schichten, eine Eingangs- und eine Ausgangsschicht. Gleichung 4.2 definiert den Ausgangsvektor $h^{(1)}$ der verdeckten Schicht. Ab $n_h \geq 2$ spricht man in der Literatur von tiefen FNN [24], [25]. Die Aktivierungsfunktion $\Phi(\cdot)$ kann von Schicht zu Schicht, wahlweise auch für jeden einzelnen Knoten variiert werden. In der Praxis hat es sich durchgesetzt, alle verdeckten Neuronen mit derselben Aktivierungsfunktion auszustatten. Lediglich die Eingangsschicht (keine bzw. lineare Aktivierungsfunktion) und die Ausgangsschicht mit der Gewichtsmatrix $\mathbf{W}^{(n_h+1)}$

$$\hat{\boldsymbol{y}} = \Phi^{(n_h+1)}(\mathbf{W}^{(n_h+1)T}\mathbf{h}^{(n_h)}) \tag{4.5}$$

unterscheiden sich. Dabei wird die Aktivierungsfunktion für die letzte Schicht (Ausgangsschicht) des Modells dem Anwendungsfall angepasst. In der vorliegenden Arbeit wurden verschiedene Aktivierungsfunktionen untersucht (vgl. Kap. 4.3). Ein FNN kann als Funktion $f(\boldsymbol{x}; \mathbf{W})$ verstanden werden, welches die Eingänge $\boldsymbol{x}$ auf die Ausgänge $\hat{\boldsymbol{y}}$

abbildet.

$$\hat{\boldsymbol{y}} = f(\boldsymbol{x}; \mathbf{W}) \tag{4.6}$$

Die Verwendung zeitunabhängiger Datenpunkte führt zu einer Darstellung des Datenraums, die sich lediglich auf die Fahrzeugleistung im jeweiligen Betriebspunkt beschränkt. Historische Informationen stehen nicht zur Verfügung. Bereiche mit geringer Dichte an Datenpunkten können erkannt und nachträglich befüllt werden. Eine Methodik zur Bewertung der Datenraumabdeckung ist die Unterteilung in Unterdatenräume $\mathcal{D}_{\text{sub}}$ und die Bewertung der zugrunde liegenden Wahrscheinlichkeitsdichtefunktionen dieser einzelnen Datenräume. Bereiche im Datenraum mit geringer Datendichte können mit neuen, extra dafür aufgezeichneten Messdaten aufgefüllt werden. Eine weitere Möglichkeit bietet das Auffüllen mit synthetischen Daten.
Die Verwendung eines RNN ermöglicht es, zeitliche Zusammenhänge zu berücksichtigen. Dazu müssen die Datenpunkte während des *preprocessing* in ihrer zeitlichen Reihenfolge belassen werden. Die Beschreibung der Datenraumabdeckung erschwert sich. Für die Verwendung als RNN Eingangsdaten muss im *preprocessing* ein *sliding window* erstellt werden. Dadurch wird jedem Zeitschritt eine Historie an vergangenen Zeitpunktdaten angefügt. Dieser Vorverarbeitungsschritt erhöht die Datenmenge signifikant. Jeder zusätzliche Zeitschritt des *sliding window* multipliziert sich mit der Anzahl j der Signale des Eingangsvektors $\boldsymbol{x}$ (vgl. Gl. 3.1) und erzeugt so zusätzliche Daten.
In einem RNN, wie in Abbildung 4.2 dargestellt, hängt der Ausgangsvektor $\mathbf{h}^{(r)}$ einer verdeckten Schicht für einen Zeitschritt t zusätzlich zu der Beziehung aus Gl. 4.4 vom für den vorangegangenen Zeitschritt berechneten Ausgangsvektor $\mathbf{h}_{t-1}^{(r)}$ ab.

$$\mathbf{h}_t^{(r)} = \Phi^{(r)}(\mathbf{W}^{(r)T}\mathbf{h}_t^{(r-1)} + \mathbf{W}_{rek}^{(r)T}\mathbf{h}_{t-1}^{(r)}), \quad r \in \{1, \ldots, n_h\}, \quad \mathbf{h}^{(0)} = \boldsymbol{x}. \tag{4.7}$$

Für den Ausgangsvektor $\mathbf{h}_{t-1}^{(r)}$ der verdeckten Schicht zum Zeitpunkt $t = t-1$ existiert eine zusätzliche Gewichtsmatrix $W_{rek}^{(r)T}$, welche die rekurrenten Verbindungen mit Gewichten versieht. Diese sind der Kern des *vanishing* und *exploding gradient* Problems, da die Berechnungen zeitlicher Abhängigkeiten bei einer großen Anzahl an Zeitschritten den Gradienten stark beeinflussen können. Dies ist im Training rekurrenter Modelle begründet. Zum Einsatz kommt die sogenannte *Backpropagation Through Time* (BPTT). Dabei müssen die Fehler vom Netzausgang über die Schichten und durch die zeitliche Dimension im ausgerollten Netz (vgl. Abb. 4.2) zurückpropagiert werden. Das Vorgehen beinhaltet zwei Phasen. Zuerst wird das Netzwerk in der Zeitdimension entfaltet, um im nächsten Schritt die Fehler zwischen Modellberechnung $\hat{\boldsymbol{y}}$ und Zielwert $\hat{\boldsymbol{y}}$ durch die Schichten zurück zur Eingangsschicht zu propagieren. Dabei kommt der von Rumelhart, Hinton und Williams vorgestellte *backpropagation*-Algorithmus zum Einsatz (vgl. Kap. 4.2.2). Der Fakt, dass die Gewichte der rekurrenten Verbindungen aus $W_{rek}^{(r)T}$ in der Zeitdimension entfaltet unverändert bleiben, sich aber ständig wiederholen, begünstigt die Entstehung eines *vanishing* oder *exploding gradients*.

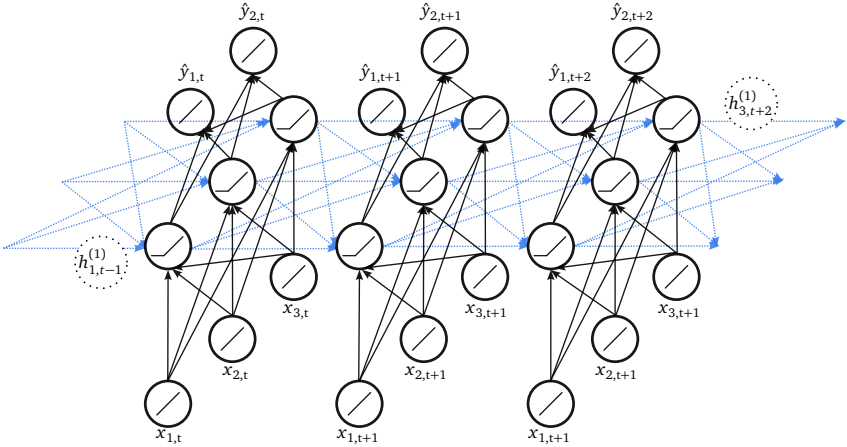


Abbildung 4.2: Ein RNN ausgerollt in der Zeitdimension nach [81]. Die hellblau-gepunkteten Linien stellen eine Rückführung innerhalb der verdeckten Schicht dar.

Ein verschwindend kleiner Gradient ist im Zuge des Trainings ein Problem, da die Anpassung der Netzwerkgewichte (das Lernen) proportional zu den Gradienten der jeweiligen Verknüpfung (sowie zur Lernrate) ist. Somit finden bei einem verschwindend geringen Gradienten keine merkliche Veränderung der Gewichtung dieser Verknüpfung und keine Anpassung (kein Lernen) der Netzwerkberechnung statt. Der Grund dafür ist, dass die betroffenen Gewichtungen keinen Beitrag durch ihre Anpassung (Veränderung) liefern, um den Fehler der Modellberechnung zu minimieren. Dem kann mit einer passenden Initialisierung der Gewichte vor dem Training entgegengewirkt werden (vgl. Kap. 4.1.4).
Der *backpropagation*-Algorithmus beschreibt, welche Änderungen der Gewichts- und Biaswerte für ein einziges Trainingsbeispiel nötig wären, um den Modellfehler für dieses Trainingsbeispiel (Trainingsdatenpunkt) zu verringern. Dies wird im Training nicht für jeden Datenpunkt des Trainingsdatensatzes durchgeführt, sondern ein Durchschnittswert über alle Änderungen ermittelt, um die Gewichte und Biaswerte anzupassen. Diese Methodik wird einmal pro Epoche bzw. einmal pro *mini-batch* angewendet. Der Vorteil eines RNN liegt in einer höheren Modellkomplexität, welche zu einer höheren Modellgüte führen kann. Dies ist bedingt durch den höheren Informationsgehalt der Zeitreihendaten.
Eine Möglichkeit zum Aggregieren der Daten besteht auch für FNN. Durch das Berechnen historischer Mittelwerte für gewisse Zeitbereiche der vorhanden Signale können im *feature engineering* zusätzliche Signale erzeugt werden (vgl. Kap. 3.2.3).

Ein Nachteil konventioneller RNN (engl. *vanilla RNN*) liegt im Erlernen von Langzeitabhängigkeiten (engl. *long term dependencies*). Dabei tritt das Phänomen der verschwindenden Gradienten (engl. *vanishing gradients*) auf, welches von Hochreiter, Bengio, Frasconi et al. in [82] definiert wurde. Kapitel 4.1.4 liefert Möglichkeiten, um diesem Phänomen und den explodierenden Gradienten (engl. *exploding gradients*) entgegen zu wirken. Für die Zielstellung dieser Arbeit sind einfache RNN den LSTM Netzwerken vorzuziehen, da sie eine ausreichende Leistungssteigerung in der Vorhersage erreichen und zwar bei gleichzeitig deutlich reduzierter Parameterzahl im Vergleich zu LSTM. Zudem wurde bereits in [103] untersucht, wie groß der Einfluss vorangegangener Zeitschritte auf die Berechnung der Motorleistung ist. Die Ergebnisse zeigen anhand der Autokorrelation der Zielsignale, dass große zeitliche Abhängigkeiten über mehr als als 10 Zeitschritte (in Sekunden) zu vernachlässigen sind.
Letztendlich ist aufgrund der hohen Modellkomplexität und der damit verbundenen größeren Modellgröße von LSTM-Netzwerken ein RNN vorzuziehen. Für die verfügbaren Ressourcen im vorliegenden Anwendungsfall bieten RNN einen Kompromiss aus Modellkomplexität und Größe. Gleichzeitig liefern sie ein deutlich größeres Potenzial zur Erfassung komplexer Abhängigkeiten als FNN.

4.1.4 Wahl der Aktivierungsfunktionen und Anpassung der Kernel-Initalisierungen

In der Grundidee der KNN stellt die Aktivierungsfunktion $\Phi(\cdot)$ die sprunghafte Aktivierung eines Neurons des menschlichen Gehirns dar, welches nach der Überschreitung eines Schwellwerts durch Reize der verknüpften Neuronen sprunghaft aktiviert wird und damit seinerseits „feuert“. Diese Reizübertragung am synaptischen Spalt gleicht einer Sprungfunktion und wird durch die Aktivierungsfunktion ersatzweise modelliert. Zum Aufbau von KNN nach diesem biologischen Vorbild wird die Sprungfunktion mathematisch durch stetig differenzierbare (glatte) Funktionen angenähert. Gängige Funktionen sind die Sigmoidfunktion (vgl. Gl. 4.8) und die hyperbolische Tangensfunktion (vgl. Gl. 4.9). Das im Training mittels *backpropagation* verwendete Gradientenabstiegsverfahren (vgl. Kap. 2.1.3) kann für diese Aktivierungsfunktionen an jeder Stelle eine Differentiation durchführen. So werden in jedem Knoten die Gradienten der Knotenfunktion gebildet. So kann der Modellfehler durch die Schichten des Modells zurückpropagiert werden, um die Anpassung der Modellparameter zu beeinflussen.
Für KNN haben sich vor allem drei Aktivierungsfunktionen bewährt (vgl. Abb.: 4.3). Die Sigmoidfunktion

$$\Phi(z) = \frac{1}{1 + e^{-z}} \tag{4.8}$$

und die hyperbolische Tangensfunktion

$$\Phi(z) = tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \tag{4.9}$$

nähern die geforderte Sprungfunktion an. Wobei die Sigmoidfunktion durch Transformation in den Tangens hyperbolicus überführt werden kann. Beide Funktionen können während des Trainings tiefer neuronaler Netze zu Problemen in Form von *vanishing* und *exploding gradients* führen [82]. Durch die Verwendung sogenannter ReLUs (zu deutsch: Einheit mit rektifizierender, teilweise linearer Aktivierungsfunktion) kann diese Problematik gelöst werden [83], [84]. In [84] haben die Autoren zudem nachgewiesen, dass diese Art der Aktivierung (vgl. Gl. 4.10) einen deutlichen Anstieg der Leistung tiefer künstlicher neuronaler Netze bewirkt [84]. Das Verhalten dieser teilweise linearen Aktivierungsfunktion bildet nach Erkenntnissen aus der Neurobiologie die Form der neuronalen Aktivierung im menschlichen Gehirn deutlich genauer ab als die Sigmoidfunktion und die hyperbolische Tangesfunktion [84].

$$\Phi(z) = max\{0, z\}. \tag{4.10}$$

Die ReLU ist derzeit die empfohlene und meist verbreitete Form der Aktivierung für tiefe neuronale Netze [25, S. 171]. ReLUs erzeugen Neuronen mit harter Nichtlinearität, welche an der Stelle Null nicht differenzierbar sind. Neuronen mit ReLU können den Wert $h_i^{(r)} = 0$ ausgeben und sind somit faktisch ausgeschaltet. Das führt in der Anwendung zu einer möglichen Ausdünnung der verwendeten Neuronen in einer verdeckten Schicht. Die Neuronen sind entweder ausgeschaltet oder sie liefern eine lineare Aktivierung. Dies kommt einer Regularisierung in Form von Dropout-Schichten gleich. Dabei wird der Nachteil, dass die Funktion an der Stelle $z = 0$ nicht stetig differenzierbar ist, als vergleichsweise gering gegenüber den nutzbaren Vorteilen bewertet [25, S. 188f]. Zwar führt dies bei den verwendeten gradientenbasierten Lernverfahren theoretisch zur Möglichkeit die Fehlerfunktion an einer Stelle nicht differenzieren zu können. In der Praxis wird dies softwareseitig durch die Verwendung des links- oder rechtsseitigen Gradienten (0 oder 1) der ReLU an der Stelle $z = 0$ gelöst [25, S. 188]. Damit wird kein Fehler ausgegeben. Die einfache Differentiation der linearen Abschnitte dieser Aktivierungsfunktion kann ihr Potential für alle anderen Werte ausspielen. Dieses teilweise lineare Verhalten ist ein weiterer Vorteil und ermöglicht einen besseren Fluss der Gradienten. Zudem bedingen ReLUs weniger komplexe mathematische Berechnungen, da keine Exponentialfunktionen verarbeitet werden müssen und ein Teil der Neuronen exakt Null ist. Im Vergleich dazu geben Sigmoid und Tangens hyperbolicus niemals den exakten Wert Null aus [84]. Dafür laufen sie für stark positive oder negative Werte in eine Sättigung. Ihre volle Sensitivität zeigt sich nur für Eingangswerte nahe $z = 0.5$ (Sigmoidfunktion) bzw. nahe $z = 0$ (hyperbolische Tangensfunktion). Die weiten Bereiche, in denen diese stetig differenzierbaren Funktionen in eine Sättigung laufen, erschweren das Training mit dem Gradientenabstiegsverfahren bzw. SGD (vgl. Kap. 2.1.3). Daher sind sie für den Einsatz in verdeckten Schichten gegenüber der Verwendung von ReLUs deutlich weniger geeignet [25].
Ziel der Verwendung dieser nichtlinearen Aktivierungsfunktionen (vgl. Abb.: 4.3) ist es, für nichtlineare Beziehungen und Zusammenhänge eine lineare Separierbar-

keit zu ermöglichen [26]. Ohne eine Aktivierungsfunktion, bzw. mit einer linearen Aktivierungsfunktion wäre ein KNN lediglich in der Lage, eine lineare Regression durchzuführen. Nichtlineare Zusammenhänge zwischen Ein- und Ausgangsdaten könnten nicht dargestellt werden.
Die Verwendung von ReLUs (vgl. Gl. 4.10) wurde zuerst von Hahnloser, Sarpeshkar, Mahowald et al. im Jahr 2000 in [83] vorgeschlagen. Sie hat große Vorteile in Hinsicht auf Geschwindigkeit des Trainings und Genauigkeit der kreierten Modelle. Dabei ist die höherer Geschwindigkeit direkt mit einer höheren Genauigkeit verwoben. Sie ermöglicht das Training von tieferen Modellen in kürzerer Zeit [26]. Im Jahr 2011 untersuchten Glorot, Bordes und Bengio, dass sich tiefe KNN mit ReLUs effizienter trainieren lassen als mit Sigmoid- oder der hyperbolischen Tangensfunktionen (vgl. Gl. 4.8 u. 4.9) [84]. Weiterer Vorteil gegenüber der Sigmoid- und hyperbolischen Tangensfunktion ist die stark reduzierte Anfälligkeit für *vanishing* und *exploding gradients* [82]. Dennoch können bei hoher Lernrate Neuronen mit ReLU absterben. Um dem entgegen zu wirken, ist eine kleinere Lernrate oder die Verwendung einer *leaky ReLU*, einer modifizierten ReLU Aktivierung, möglich [56]. Goodfellow, Bengio und Courville haben bestätigt, dass unter Einsatz von ReLU-Aktivierung ein universeller Funktionsapproximator erstellt werden kann [25]. Dies ist bereits von Nelles im Jahr 2001 für ein MLP mit den damals weit verbreiteten nichtlinearen Aktivierungsfunktonen (Sigmoidfunktion und Tanges hyperbolicus) beschrieben worden [24]. Dadurch wird diese Erkenntnis auf den aktuellen Stand der Wissenschaft übertragen. Der Einsatz

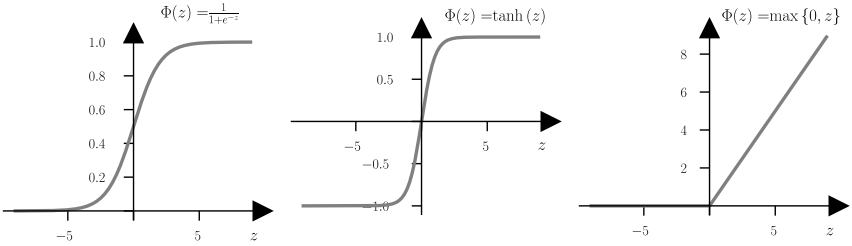


Abbildung 4.3: Gängige Aktivierungsfunktionen $\Phi(\cdot)$ von links nach rechts: Sigmoidfunktion, hyperbolische Tangensfunktion, ReLU.

einer linearen Aktivierungsfunktion führt zu einem KNN, das ausschließlich zur linearen Regression verwendet werden kann. Die Generalisierbarkeit und Modellgüte für KNN nimmt bei der Verwendung nichtlinearer Aktivierungsfunktionen (vgl. Abb. 4.3) signifikant zu [26].
Da die Initialisierung der Parameter eines KNN immer randomisiert stattfindet, kann es nach Glorot und Bengio zu ungünstigen Gewichtungen zum Start des Trainings kommen. Je nach Wahl der Aktivierungsfunktion müssen diese auf unterschiedliche

Tabelle 4.6: Übersicht gängiger Aktivierungsfunktionen mit entsprechender Kernel-Initialisierungen

Aktivierungsfunktion	Kernel-Initialisierung	Verteilung
Sigmoidfunktion	normierte *Xavier*-Init.	stetige Gleichvert. $\mathcal{U}$
Tangens hyperbolicus	normierte *Xavier*-Init.	stetige Gleichvert. $\mathcal{U}$
ReLU	normierte *He*-Init.	gaußsche Normalvert. $\mathcal{N}$

Wertebereiche angepasst werden. Gleiches gilt für die gesättigten Bereiche, in denen sich keine Änderungen in der Ausgabe der Aktivierungsfunktion ergeben (trotz Variation in der Eingabe). Um gesättigte Neuronen (Knoten) zu Beginn des Trainings zu vermeiden, werden in Abhängigkeit der Aktivierungsfunktion passende sogenannte Kernel-Initialisierungen verwendet (siehe Tabelle 4.6). Die normierte Initialisierung nach Glorot und Bengio, auch normierte *Xavier*-Initialisierung genannt, sieht einen initialen Wert von Null für die Biaswerte jeder Schicht vor. Die Gewichtsmatrix wird nach folgender Gleichung initialisiert

$$\mathbf{W}^{(r)} \sim \mathcal{U}\left[-\frac{\sqrt{6}}{\sqrt{n_{r-1}+n_r}}, \frac{\sqrt{6}}{\sqrt{n_{r-1}+n_r}}\right]. \tag{4.11}$$

Hierbei stellt $\mathcal{U}[-a,a]$ die uniforme Verteilung im Intervall $[-a,a]$ dar, aus welchem der Zufallswerte für die Initialisierung gezogen werden. Die Variable n_r gibt die Anzahl der Knoten in der zugehörigen Schicht an. Hierbei ist das Ziel, den Mittelwert Null und eine konstante Varianz für alle Gewichtsmatrizen jeder Schicht zu gewährleisten. Dadurch soll das Auftreten von *vanishing* und *exploding gradients* reduziert werden [85]. Es soll also gelten:

$$n_r Var\left[\mathbf{W}^{(r)}\right] = 1, \quad \forall r. \tag{4.12}$$

Dadurch sollen die initialisierten Wichtungen eine Sättigung des Tangens hyperbolicus vermeiden.
Für Schichten mit ReLUs gilt eine abgewandelte Form, die dem Umstand Rechnung trägt, dass die Aktivierungsfunktion nicht punktsymmetrisch zum Ursprung ist. Sie berücksichtigt das nichtlineare Verhalten [86]. Folgende Bedingung muss gelten:

$$\frac{1}{2} n_r Var\left[\mathbf{W}^{(r)}\right] = 1, \quad \forall r. \tag{4.13}$$

Dies führt zu einer gaußschen Normalverteilung $\mathcal{N}(x, \sigma_{He}^2)$ mit dem Mittelwert Null (vgl. Kap. 2.1) [86]. Die zugehörige Standardabweichung σ_{He} ist

$$\sigma_{He} = \sqrt{\frac{2}{n_{r-1}}}. \tag{4.14}$$

Erneut werden die Biaswerte mit Null initialisiert. Die ReLU-Aktivierung für Eingangswerte $z \leq 0$ wird Null (vgl. Gl. 4.10)). Eine beschnittene, gaußsche Normalverteilung wird für die Ziehung der zufällig initialisierten Gewichtsparameter verwendet.

$$\mathbf{W}^{(r)} \sim \mathcal{N}\left[0, \sqrt{\frac{2}{n_{r-1}}}\right] \tag{4.15}$$

Diese Verteilung hat nach [86] als untere Grenze Null und die obere Grenze $\sigma_{He} = \sqrt{\frac{2}{n_{r-1}}}$. Die hier aufgeführten wissenschaftlichen Betrachtungen der Aktivierungsfunktionen wurden zum Anlass genommen, für diese Arbeit alle verdecken künstlichen Neuronen mit einer ReLU und passender Initialisierung auszustatten.

4.2 Training der Modelle

Die Optimierung der Parameter eines KNN wird als Training bezeichnet. Ziel ist es, den Fehler der Zielfunktion zu verringern. Der Begriff „Training“ ist begründet in der iterativen Verwendung der Trainingsdaten. Für das Training der KNN werden immer wieder dieselben Datensätze verwendet, um die Zielfunktion zu minimieren. Ein Durchlauf aller Trainingsdaten wird als Epoche bezeichnet [25]. Die Parameterzahl datenbasierter Modelle ist abhängig von ihrer Architektur. Bei KNN beeinflussen die Anzahl n_h der verdeckten Schichten (Modelltiefe) sowie die Anzahl n_n der Neuronen pro verdeckter Schicht diese Zahl maßgeblich. Die Variablen n_h und n_n sind Hyperparameter. Sie definieren die Größe der Gewichtsmatrizen $\boldsymbol{W}$ und Biasvektoren $\boldsymbol{b}$, welche die im Training zu optimierenden Parameter enthalten. Zusätzliche Hyperparameter stellen bspw. die Wahl der Aktivierungsfunktion, die Initialisierung der Gewichte, die Zielfunktion, die Regularisierungsmaßnahmen, die Anzahl der Trainingsepochen, die Lernrate sowie die Art der Reduzierung der Lernrate dar (vgl. Tab. 4.5). Hyperparameter werden in Form einer Gitterstudie außerhalb des individuellen Modelltrainings optimiert (vgl. Kap. 4.1.2). Die Gitterstudien verändern die Hyperparameter in diskreten Schritten innerhalb vorgegebener Intervalle. Die Modellparameter werden im Training mittels Gradientenabstiegsverfahren optimiert, um die Modellausgänge $\hat{\boldsymbol{y}}$ mit einem möglichst geringen Fehler an die Zielwerte $\boldsymbol{y}$ anzugleichen. Dabei wird der *backpropagation*-Algorithmus von [80] angewendet, um den Modellfehler durch die Schichten des KNN zum Modelleingang zu propagieren [80]. Die Kombination der Gitterstudie mit dem Modelltraining wurde als automatisierter Workflow umgesetzt.

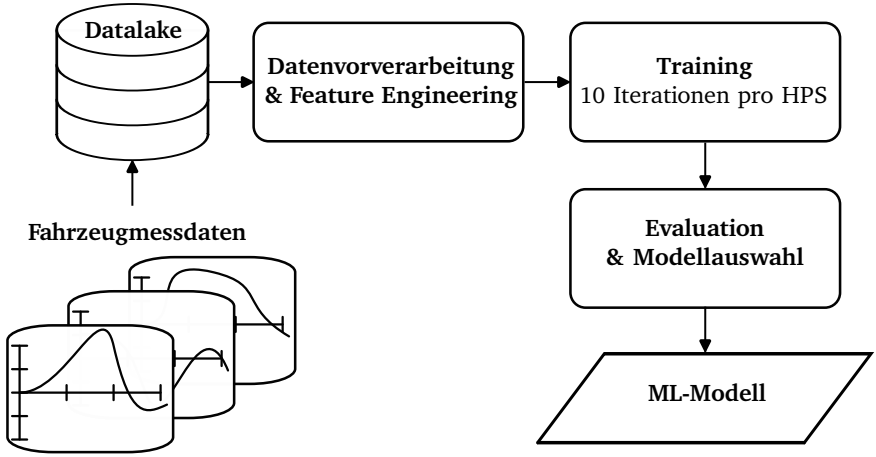


Abbildung 4.4: Schema der ML-Pipeline. Die verschiedenen HPS einer Gitterstudie werden jeweils für $n_g = 10$ Trainingsläufe verwendet, um die Einflüsse der randomisierten Initialisierung auszugleichen.

Eine mögliche Hyperparameterkombination wird als Hyperparameterset (HPS) bezeichnet. Jedes HPS wird $n_g = 10$ mal trainiert, um eine Varianz bei der Initialisierung der Gewichtsmatrizen $\boldsymbol{W}$ auszugleichen und das beste Modell für jedes HPS zu finden. Der gesamten Modellierungsprozess wurde automatisiert auf einer Cloudinstanz ausgeführt.
Durch den Einsatz des Gradientenabstiegsverfahrens werden die Modellparameter schrittweise optimiert. Zu Beginn des Trainings werden sämtliche Gewichtsmatrizen zufällig initialisiert (vgl. Kap. 4.1.4). Die Biasvektoren werden mit Null initialisiert. Die Eingangsvektoren $\boldsymbol{x}$ werden in das KNN eingespeist und die Netzberechnung $\hat{\boldsymbol{y}}$ mit den zugehörigen Zielwerten $\boldsymbol{y}$ (engl. *targets*) verglichen (vgl. Kap. 4.2.1). Ein kompletter Durchlauf aller Trainingsdaten wird als Epoche bezeichnet, sofern das Trainingsset finit ist [55]. Anhand der Ergebnisse einer Epoche werden im Anschluss die Modellparameter in $\boldsymbol{W}$ und $\boldsymbol{b}$ angepasst, um den aktuellen Fehler zu reduzieren. Die Anzahl der Epochen beeinflusst die Modellgüte. Sie ist ein wichtiger Hyperparameter. Die Anzahl der Epochen und die Wahl der weiteren Hyperparameter (siehe Kap. 4.2.3) beeinflussen die Modellgüte stark. Durch eine zu große Zahl an Epochen kann es im Training zu einer sogenannten Überanpassung (engl. *overfitting*) kommen (vgl. Kap. 3.1 bzw. Kap. 2.2.4). Das Modell repräsentiert in diesem Fall zwar die Trainingsdaten, verliert aber die Fähigkeit, auf unbekannten Daten zu generalisieren. Mechanismen wie das *early stopping* werden eingesetzt, um ein *overfitting* zu vermeiden. Die Überanpassung

wird anhand der unabhängigen Validierungsdaten bewertet. Findet für einen definierten Zeitraum keine Verbesserung auf den Validierungsdaten statt, wird das Training frühzeitig abgebrochen.

4.2.1 Bewertung der Modellgüte

Die Bewertung der Modellgüte ist zum einen während des Trainings für den Optimierungsalgorithmus (vgl. Kap. 4.2.2) ausschlaggebend, zum anderen wird sie in der Gitterstudie verwendet, um verschiedenen Architekturen und Modellgrößen zu vergleichen. Dabei soll die Tauglichkeit der Modelle für den vorgesehenen Einsatzbereich ermittelt werden. Verschiedene Kennzahlen werden als Gütemaße eingeführt, welche für jedes Modell berechnet werden. Die errechneten, skalaren Werte können anschließend zur automatisierten Bildung einer Rangfolge und Auswahl des besten Modells verwendet werden.
Das Ziel einer jeden Modellierung ist es, einen Prozess so genau wie möglich durch ein Modell abzubilden [24]. Die Modellgüte beschreibt, wie gut das Modell die gegebenen Zielwerte abbilden kann. Die erreichte Modellgüte muss mindestens so hoch sein wie für den Anwendungsfall gefordert. Dabei limitieren Speicherkapazität, Rechenzeit, Echtzeitfähigkeit und Komplexität der physikalischen Zusammenhänge in vielen Fällen die Modellgüte. Speicherkapazität und Rechenzeit beschränken im Speziellen die Größe und Komplexität des verwendeten Modells bzw. die Anzahl der Modellparameter. Da im Training eine Fehlerfunktion zwischen dem Modellausgang $\hat{\boldsymbol{y}}$ und dem abzubildenden Prozess minimiert werden muss, finden die Begriffe Kostenfunktion oder Verlustfunktion (engl. *loss function*) ebenfalls Verwendung in der Literatur [54], [87]. Mit Hilfe dieser Kostenfunktion werden die Modellparameter iterativ optimiert. Das angestrebte Minimum ist erreicht, wenn der Gradient der Zielfunktion Null ist [24]. Während des Trainings werden die Trainings- und Validierungsdaten $\mathbb{S}_{\text{train}}$ und $\mathbb{S}_{\text{vali}}$ zur Berechnung der Zielfunktion verwendet. Die trainierten Modelle werden anhand des unabhängigen Testdatensatzes $\mathbb{S}_{\text{test}}$ bewertet (vgl. Kap. 3.3.1).
Ein weit verbreitetes Fehlermaß ist die summierte quadratische Abweichung (engl. *Sum of Squared Errors* (SSE))

$$SSE = \sum_{i=1}^{n} (\boldsymbol{y}_i - \hat{\boldsymbol{y}}_i)^2, \tag{4.16}$$

wobei n die Anzahl der Datenpunkte in Form des Zielvektors $\boldsymbol{y}$ darstellt, welcher mit den Modellberechnungen $\hat{\boldsymbol{y}}$ für jeden Datenpunkt verglichen wird. Die Quadratsumme der Abweichungen der Messungen vom Mittelwert (engl. *Total Sum of Squares* (SST)) kann mit

$$SST = \sum_{i=1}^{n} (\boldsymbol{y}_i - \overline{\boldsymbol{y}}_i)^2 \tag{4.17}$$

berechnet werden. Die Größe $\overline{\hat{y}}$ bezeichnet den Mittelwert der Messdaten. Dem gegenüber steht der mittlere quadratische Fehler (engl. *Mean Squared Error* (MSE)),

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{y}_i - \hat{\boldsymbol{y}}_i)^2 \tag{4.18}$$

berechnet mit Hilfe der Anzahl aller zum Testen verwendeten Messpunkte n. Dadurch entsteht eine gemittelte Abweichung über alle berechneten Modellausgänge hinweg. Der MSE wird in dieser Arbeit als Kostenfunktion (Zielfunktion) für die Optimierungsalgorithmen *Adam* und *RMSprop* (vgl. Kap. 4.2.2) eingesetzt. Für die genannten Verfahren wird ein Regularisierungsterm Reg_λ angefügt (vgl. Gl. 4.22 und Kap. 4.2.4). Um ein Fehlermaß in der Einheit der betrachteten Ausgangsgröße zu erhalten, wird die Quadratwurzel des MSE als Wurzel der mittleren quadratischen Abweichung (RMSE) wie folgt berechnet:

$$RMSE = \sqrt{MSE}. \tag{4.19}$$

Diese Größe dient der intuitiven Vergleichbarkeit zwischen Messgröße und Modellausgang, da es die Einheit der Ziel- bzw. Ausgangsgröße besitzt. Das Gütemaß wird zur verständlichen Aufbereitung der Ergebnisse der Gitterstudie verwendet.
Zur Überprüfung, wie gut ein Modell die gegebenen Testdaten abbildet, existiert darüber hinaus das Bestimmtheitsmaß nach [76]

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(\boldsymbol{y}_i - \hat{\boldsymbol{y}}_i)^2}{\sum_{i=1}^{n}(\boldsymbol{y}_i - \overline{\boldsymbol{y}}_i)^2} = 1 - \frac{SSE}{SST}. \tag{4.20}$$

Dieses Gütekriterium hängt unmittelbar von den Residuen der Modellberechnungen ab und kann durch einen Strafterm erweitert werden, der eine große Zahl der Modellparameter n_{para} bestraft und somit einer Überanpassung entgegenwirkt [76]. Für das Bestimmtheitsmaß gilt: je näher R^2 der Zahl 1, desto besser ist das Modell. Das Bestimmtheitsmaß $R^2 = 1$ sagt aus, dass das überprüfte Modell $100 \cdot R^2$ % der Messdaten erklärt [45], [76].
Zur Ermittelung des RMSE für die Bewertung innerhalb der Gitterstudie wird das trainierte Modell auf mehrere Testmessungen angewendet. Die berechneten Modellausgänge werden mit den Zielgrößen aus den Testmessungen verglichen, um daraus den RMSE zu berechnen (vgl. Gl. 4.19). Dieser skalare Wert gibt Aufschluss über die Abweichung zum wahren Zielwert (engl. *ground truth*).
Zwischen den Trainingsschritten wird in regelmäßigen Abständen ein Validierungsschritt durchgeführt. Hierbei wird das aktuelle Modell auf den Validierungsdatensatz $\mathbb{S}_{\text{vali}}$ (vgl. Kap. 3.3.1) angewandt. Zeigt sich über mehrere Validierungsschritte keine Verbesserung oder eine Verschlechterung, ist dies ein Hinweis auf eine Überanpassung des Modells an die Trainingsdaten. Diese Ergebnisse aus den Validierungsschritten werden eingesetzt, um das Training frühzeitig abzubrechen (*early stopping*), da das Modell ansonsten zwar eine hohe Modellgüte auf den Trainingsdaten zeigt, aber mit

unbekannten Daten keine vergleichbar guten Ergebnisse erzielt. Eine gute Modellgüte bei der Verarbeitung unbekannter Daten ist jedoch für die Anwendung des Modells nach dem Training essentiell, um eine gute Generalisierung zu erzielen. Eine weitere Evaluation findet am Ende einer jeden Epoche (Bewertung des aktuellen Modellfehlers) sowie am Ende des Trainings mit dem Testdatensatz $\mathbb{S}_{\text{test}}$ statt.
Um auszuschließen, dass der Testdatensatz $\mathbb{S}_{\text{test}}$ nicht ausreichend repräsentativ ist, wird für die besten Modelle der Gitterstudie eine Kreuzvalidierung (engl. *cross validation*) durchgeführt werden. Dabei werden die gesamten verfügbaren Daten iterativ jeweils als Trainings- und als Testdaten verwendet [24].

4.2.2 Lernverfahren und Parameteroptimierung

Lernverfahren können in *batch-learning* und *online-learning* unterteilt werden [54], [55]. Erstgenanntes Verfahren verwendet den kompletten Trainingsdatensatz auf einmal. Innerhalb einer Epoche wird der Mittelwert der Abweichung für alle Datenpunkte verwendet, um die Optimierung der Gewichte vorzunehmen. Dieses Vorgehen kann schnell die Rechenkapazität handelsüblicher Computer übersteigen und wird daher meist auf skalierbaren Cloud-Instanzen angewendet.
Das *online-learning* hingegen berücksichtigt jeden Datenpunkt isoliert und führt eine Optimierung der Modellparameter aufgrund der resultierenden Abweichung für diesen Modelleingang durch. Ein inkrementelles Lernen kann durchgeführt werden. Während der Anwendung des Modells gesammelte, neue Trainingsdaten können direkt für weitere Optimierungen der Modellparameter (*online-learning*) verwendet werden. Die Modellparameter sind nicht fixiert und das Modellverhalten kann sich während der Anwendung ändern.
Ein Kombination aus beiden Verfahren ist das sogenannte *mini-batch*-Verfahren. Dieses unterteilt den Trainingsdatensatz in kleinere Datenbündel (engl. *batches*), für welche der Mittelwert der Abweichung im Training verwendet wird, um die Modellparameter nach jedem *mini-batch* anzupassen. Dieses Verfahren ist ressourcenschonend, da es nur den benötigten *mini-batch* in den Speicher lädt. Zudem ist es effizienter als das konventionelle *online-learning*, da nicht jeder Datenpunkt einzeln für ein Update der Modellparameter verwendet wird. Das *mini-batch*-Verfahren bietet eine effiziente und praktikable Lösung für ein Training von ML-Modellen [25], [54]. Da die Anwendung der KNN auf dem Steuergerät kein inkrementelles Lernen zulässt (vgl. Kap. 1.2), wird für das Training das *mini-batch*-Verfahren verwendet, welches auch breite Anwendung in der Literatur findet [88], [89, S. 9]. Die Größe der *mini-batches* n_{mb} ist ebenfalls ein Hyperparameter, der das Modelltraining beeinflusst (vgl. Tab. 4.5).
Die Anforderung, ein möglichst ressourcenschonendes ML-Modell für die Anwendung auf dem Steuergerät zu entwickeln, steht in direktem Gegensatz zur Anforderung, eine möglichst hohe Modellgüte zu realisieren. Daher müssen am Ende der Gitterstudie eine Pareto-Front erstellt werden und die geeigneten Modelle mittels Gewichtung der beiden Anforderungen sortiert werden. Im Training der Modelle wird ein Vorgehen

verfolgt, welches mit dem kleinstmöglichen Modell beginnt und dessen Modellgüte bewertet. Darauf aufbauend wird Schritt für Schritt die Modellkomplexität erhöht, bis die geforderte Modellgüte erreicht ist. Dabei wird die Modellgüte anhand von vorab definierten Testmessungen (vgl. Kap. 3.3.1) und den in Kap. 4.2.1 beschriebenen Gütekriterien bestimmt. Diese Praxis wird in [25] und [24] empfohlen. Die Modellkomplexität wird mit steigender Anzahl der verdeckten Schichten n_h und der verdeckten Neuronen n_n gesteigert. Alle derartigen Erweiterungen des FNN führen eine Zunahme der Parameterzahl herbei. Sie ermöglichen es, komplexere Zusammenhänge abzubilden. Gleichzeitig steig jedoch die Gefahr der Überanpassung durch ein Auswendiglernen der Trainingsdaten.
Als Optimierungsalgorithmus zur Anpassung der Netzwerkgewichte haben sich in der Literatur die Gradientenabstiegsverfahren durchgesetzt (vgl. Kap. 2.1.3. Hierbei vor allem Varianten des SGD wie der sogenannte *Adam*-Optimierer [90] und *RMSprop* [91], [92], welche in der Gitterstudie verglichen werden. Beide Verfahren verbessern den klassischen SGD in Bezug auf Konvergenzgeschwindigkeit und Vermeidung von Sattelpunkten der Kostenfunktion [92]. Dafür bedienen sie sich adaptiver Lernraten, welche in Abhängigkeit der vorangegangenen Iterationen reduziert wird. Das klassische SGD passt die Netzwerkgewichte $\mathbf{W}$ wie in Gleichung 4.21 an [92].

$$\mathbf{W}^{(r+1)} = \mathbf{W}^r + \epsilon_0 \nabla g^{(r)}(\mathbf{W}^r) \tag{4.21}$$

Dabei ist z die Zielfunktion oder Kostenfunktion (engl. *cost function*), die sich aus Gl. 4.18 und einem Regularisierungsterm Reg_λ (vgl. Kap. 4.2.4) zusammensetzt [92].

$$z = \frac{1}{n}\sum_{i=1}^{n}(y_l - \hat{y}_l)^2 + Reg_\lambda \tag{4.22}$$

Der Einfluss des Terms zur Regularisierung Reg_λ wird über einen Hyperparameter λ gesteuert. *RMSprop* erweitert die impulsbasierten (engl. *momentum*) Methoden des Gradientenabstiegs mit adaptiven Lernraten um ein Verfahren zur Vermeidung zu stark verringerter Lernraten [91], [92]. Der *Adam*-Optimierer kombiniert ebenfalls *momemtum*-Methoden und erweitert diese um Verfahren und Parameter zur Vermeidung zu stark fallender Lernraten [90]. Beide Optimierer sind in der aktuellen Anwendung für das Training von KNN stark vertreten [92] und werden in der Gitterstudie verglichen.

Backpropagation Um das Gradientenabstiegsverfahren auf alle Netzwerkparameter anwenden zu können und die Abweichung von Modellberechnung zu den Zielwerten zu berücksichtigen, wird der *backpropagation*-Algorithmus (vgl. Algorithmus 2) verwendet [80]. Die *backpropagation* wurde 1985 von Rumelhart, Hinton und Williams vorgestellt. Der Algorithmus berechnet die für die Ableitung der Zielfunktion benötigte Kettenregel in einer bestimmten Reihenfolge an Operationen, welche den Prozess sehr effizient

machen [25], [80]. Der *backpropagation*-Algorithmus liefert im Training die effiziente Berechnung der Gradienten unter Anwendung der Kettenregel. Diese Gradienten werden vom Algorithmus des gewählten Lernverfahrens benötigt, um eine Anpassung der Modellparameter durchzuführen. Für den allgemeinen Fall werden keine Vektoren,

Algorithmus 2 Durchführung der *backpropagation*

Eingabe: Modellberechnung $\hat{\boldsymbol{y}}$

1: Berechnung der Zielfunktion $z = g(\hat{\boldsymbol{y}})$ (vgl. Gl. 4.22) als Abweichung zu Zielvektor $\boldsymbol{y}$
2: Zielfunktion z ist Fehlerfunktion der Modellberechnung $\hat{\boldsymbol{y}}$ mit $\hat{\boldsymbol{y}} = [\hat{y}_1, \hat{y}_2, \dots \hat{y}_k]^T$
3: Zielfunktion muss minimiert werde, um Modellfehler zu minimieren
4: Gradient der Zielfunktion zeigt in Richtung des lokalen Maximums, der negative Gradient $-\frac{\partial z}{\partial \hat{y}}$ zeigt in Richtung des lokalen Minimums
5: Der Gradient von z gibt an, in welche Richtung $\hat{\boldsymbol{y}}$ verändert werden muss, um z zu minimieren. Dies kann für jeden Wert $\hat{y}_l$ des Vektors $\hat{\boldsymbol{y}}$ berechnet werden.
6: $\hat{\boldsymbol{y}}$ ist eine Funktion von $\hat{\boldsymbol{y}} = f(\boldsymbol{x})$ bzw. der Ausgabe $h^{(r)}$ der letzten Schicht des KNN.
7: Damit ergibt sich eine Abhängigkeit der Größe $z = g(f(x))$, für die ein Gradient $\frac{\partial z}{\partial x_i}$ gebildet werden muss.
8: In der Analysis kann die Ableitung einer Funktion z, die aus anderen Funktionen $y = f(x)$ erzeugt wird und deren Ableitungen bekannt sind, aus den Ableitungen dieser Funktionen gebildet werden [25, S. 201].
9: Dafür muss die Kettenregel der Analysis verwendet werden $\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}$ (für einen skalaren Wert x)
10: Generalisiert man dies für Vektoren so muss gelten $\boldsymbol{x} \in \mathbb{R}^m$ sowie $\hat{\boldsymbol{y}} \in \mathbb{R}^n$ und f bildet von $\mathbb{R}^m$ auf $\mathbb{R}^n$ ab und g bildet von $\mathbb{R}^n$ auf $\mathbb{R}$ ab.
11: Wenn gilt $\hat{\boldsymbol{y}} = f(\boldsymbol{x})$ und $z = g(\hat{\boldsymbol{y}})$, dann ist $\frac{\partial z}{\partial x_i} = \sum_l^k \frac{\partial z}{\partial \hat{y}_l}\frac{\partial \hat{y}_l}{\partial x_i}$
12: Dabei muss für die Ableitung von z nach einem Wert x_i die Ableitung eines jeden Eintrags $\hat{y}_l$ des Vektors $\hat{\boldsymbol{y}}$ nach x_i berechnet werden (mit $\frac{\partial \hat{y}_l}{\partial x_i}$)
13: Dann muss die Ableitung von z nach jedem Wert $\hat{y}_l$ berechnet werden. Die Addition all dieser Ergebnisse ergibt den Wert für die Ableitung von z nach einem Eintrag x_i des Eingangsvektors $\boldsymbol{x}$
14: Das kann in Vektorschreibweise ausgedrückt werden durch $\nabla_{\boldsymbol{x}} z = \left(\frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{x}}\right)^T \nabla_{\hat{\boldsymbol{y}}} z$
15: Hierbei ist $\frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{x}}$ die $n \times m$ Jacobi-Matrix von f.

Ausgabe: Der *backpropagation*-Algorithmus führt diese Operation (Multiplikation einer Jacobi-Matrix mit einem Gradienten) für alle Verknüpfungen in einem KNN aus.

sondern Tensoren betrachtet, um alle Architekturen und Netzgrößen abbilden zu können. Mathematisch gesehen, können diese Tensoren ebenfalls in Vektoren überführt und die in Algorithmus 2 beschriebe Logik angewendet werden. Hierbei werden beim

Training in jeder Epoche, pro *mini-batch*, die errechneten Fehlerwerte verwendet. Ausgehend vom berechneten Ergebnis $\hat{y}$ der Ausgabeschicht werden die Gewichts- und Biasvektoren (Modellparameter) angepasst. Dabei werden die einzelnen Schichten des Modells von hinten nach vorne durchlaufen. Die Parameteranpassung findet mittels den auf dem Gradientenabstieg basierenden Optimierungsverfahren stets in Richtung des aktuell kleinsten Modellfehlers statt. Der *backpropagation*-Algorithmus propagiert den in der Ausgabeschicht generierten Modellfehler Schicht für Schicht durch das Modell zurück [24], [25], [80]. Zum Zeitpunkt dieser Arbeit ist der *backpropagation*-Algorithmus *state-of-the-art* und die effizienteste Methode, um Modellparameter von KNN zu optimieren, da er die Anwendung der Kettenregel auf eine Vielzahl ineinander verschachtelter Funktionen effizient in Bezug auf die benötigte Rechenzeit durchführt [25].
Im entwickelten, automatisierten Arbeitsablauf werden kontinuierlich neue Daten aus dem Fahrzeug zu den vorhandenen Trainingsdaten hinzugefügt. Nach einem Datenupdate wird das gesamte KNN nochmals mit der neuen Datenbasis trainiert. Damit können im erneuten Training die Einflüsse der neuen Datengrundlage berücksichtigt werden [54]. Dafür bietet die Ausführung auf einer Cloud-Instanz bestmögliche Rechenleistung und Geschwindigkeit. Nach obiger Definition wird das *mini-batch*-Verfahren und kein *online-learning* angewendet.
Die allgemein verwendeten, iterativen Gradientenabstiegsverfahren haben den Nachteil, dass sie eine nicht-konvexe Zielfunktion nur in sehr kleine Wertebereich treiben, aber keine Garantie auf Konvergenz bieten. Im Vergleich dazu konvergieren konvexe Optimierungsalgorithmen für konvexe Zielfunktionen von jedem Startpunkt aus. Das heißt, in der Theorie genügt jede Parameterwahl als Ausgangspunkt für eine erfolgreiche Optimierung mit einer konvergierten Zielfunktion. Die Gradientenabstiegsverfahren sind sensitiv auf die Initialisierung der Modellparameter (vgl. Kap 4.1.4) [25]. Das exakte Lösen eines Optimierungsproblems ist der Komplexitätsklasse der nicht deterministischen polynomialen (NP-schweren) Probleme zugeordnet. Für diese ist meist eine Näherungslösung das Ergebnis einer Optimierung [88], [93]. Eine passende Initialisierung (vgl. Kap. 4.1.4) ist nötig, um die Chancen zu erhöhen, ein lokales Minimum zu finden. In der Gitterstudie werden mit jedem HPS 10 Modelle trainiert, um diese Unsicherheiten weiter zu reduzieren.
Die verwendete *Deep-Learning*-Programmbibliothek *Tensorflow* ist durch ihren Aufbau ausschließlich auf *backpropagation* ausgerichtet. Unter der Abstraktionsschicht *Keras* wird ein sogenannter Berechnungsgraph aufgebaut, der pro Optimierungsschritt rückwärts durchlaufen wird [94]. Die im *backpropagation*-Algorithmus verwendeten Tensoren fließen durch den aufgebauten Programmgraphen. Die Programmbibliothek (engl. *library*) bietet eine Vielzahl an Optimierungsalgorithmen. In dieser Arbeit wurden die beiden oben beschriebenen Optimierungsverfahren *Adam* und *RMSprop* getestet und verglichen [90], [91]. Im Zuge der Gitterstudien in dieser Arbeit zeigt sich, dass beide Optimierungsalgorithmen leistungsstark sind und ähnliche Ergebnisse erzielen. Die Modellgüte hängt deutlich stärker von der gewählten Netzarchitektur

und den Eingangsdaten ab als von den unterschiedlichen Trainingsalgorithmen. Dies haben Untersuchungen im Zuge dieser Arbeit ergeben [29]. Ergebnisse werden in Kapitel 4.4 dargestellt.

4.2.3 Entwicklung einer Methodik zur Identifikation der Hyperparameter

Während des Trainings eines ML-Modells müssen eine Vielzahl von Modellparametern optimiert werden. Übergeordnet existieren sogenannte Hyperparameter. Diese sind in [55] definiert als Variablen, welche vor dem eigentlichen Training gewählt und nicht direkt selbst im Training angepasst werden. Die Hyperparameter unterteilen sich in zwei Kategorien.
Zum einen existieren Hyperparameter, die direkt die Modellarchitektur und Funktionsweise des ML-Modells bestimmen, zum anderen existieren Hyperparameter, die das Training bzw. die Lernverfahren definieren. Erstere definieren bspw. die Netzarchitektur durch die Anzahl der Neuronen n_n pro verdeckter Schicht $h^{(r)}$, die Anzahl n_h dieser verdeckten Schichten, den Wert der deaktivierten Neuronen p_drpt in Prozent für eine Dropout-Schicht.
Die zweite Kategorie definiert die wichtigsten Vorgaben für das Training. Im Speziellen die Anzahl n_e der Epochen, die Größe der *mini-batches* n_{mb}, in welche die Gesamtmenge aller Trainingsdaten unterteilt wird, und die initiale Lernrate ϵ_0. Die Lernrate stellt einen der wichtigsten Hyperparameter im Zusammenhang mit dem Gradientenabstiegsverfahren dar und sollte mit großer Sorgfalt gewählt werden [55]. Die Lernrate beschreibt die Schrittweite in Richtung des steilsten Abstiegs (in negative Richtung des Gradienten). Eine adaptive Lernrate definiert eine Änderung der Schrittweite in Abhängigkeit von Epochenzahl, Wert bzw. Impuls der Zielfunktion (vgl. Kap. 4.2.2).
Zusammengefasst ergibt sich eine Vielzahl an Hyperparametern, für die eine optimale Kombination identifiziert werden muss, um das bestmögliche ML-Modell für den Anwendungsfall zu trainieren. Um die Modellgüte im Training zu maximieren, wird eine Gitterstudie (engl. *grid search*) durchgeführt, die dazu dient, die beste Hyperparameterkombination zu identifizieren. Dies geschieht automatisiert nach dem in Abbildung 4.4 dargestellten Arbeitsablauf. Dabei wird jede Kombination 10 mal ausgeführt, um die zufällig initialisierten Gewichtsvektoren in verschiedenen Kombinationen zu testen. Die zu untersuchenden Wertebereiche der Hyperparameter werden in diskrete Schritte unterteilt (siehe Tabelle 4.5). Die Ergebnisse der Gitterstudien zur Hyperparameter-Identifikation werden in Kapitel 6.2 diskutiert. Eine weitere entscheidende Variable definiert die Netzwerkarchitektur. In den beiden *Fallstudien I & II* werden hierfür FNN und RNN verglichen.

4.2.4 Regularisierung

Wie in Abschnitt 4.2.2 bereits erwähnt, besteht die zum Training von KNN verwendete Kostenfunktion aus einer Gütefunktion und einem zusätzlichen Term zur Regularisie-

Tabelle 4.7: Hyperparameter werden unterteilt nach ihrem Einfluss auf das Modell bzw. Architektur und auf das Training bzw. die Optimierung

Modellparameter	**Trainingsparameter / Optimierung**
Anzahl n_h verdeckter Schichten	initiale Lernrate ϵ_0 (adaptiv)
Anzahl n_n Neuronen pro Schicht	Batch Size n_{mb}
Drop Out p_{drpt} und andere Regularization Techniken	Epochen n_e
-	*Early stopping*
-	Trainingsalgorithmus (*Adam* & *RMSprop*)
-	Zielfunktion
-	Initialisierung der Gewichte

rung (vgl. Gl. 4.22). Ziel der Regularisierung ist es, die Generalisierungsfähigkeit des Modells zu erhöhen und *overfitting* zu vermeiden. Dies kann durch Beschneiden der verfügbaren Verknüpfungen, die Anwendung von Straftermen oder Parameterbegrenzungen erfolgen. Sie gilt als einer der wichtigsten Parameter im *Machine Learning*, direkt in Konkurrenz zur Optimierung selbst [25], [59]. Regularisierung wird im ML entweder durch stochastische Methoden (wie durch Einbinden von *dropout*-Schichten) oder durch die Anwendung von Straftermen implementiert. Die Strafterme basieren auf der Berechnung der Vektornorm l_u. Diese kann für einen Vektor $\mathbf{v} = (v_1, v_2, \ldots, v_m)^T$ mit $\mathbf{v} \in \mathbb{R}^m$ durch

$$||\mathbf{v}||_u = (|v_1| + |v_2| + \cdots + |v_m|)^{\frac{1}{u}} \tag{4.23}$$

berechnet werden. Die meist verwendeten sind die l_1-Norm

$$||\mathbf{v}||_1 = \sum_{i=1}^{n} |v_i|, \tag{4.24}$$

welche auch als Betragssummennorm des Vektors bezeichnet wird [35]. Und die l_2-Norm

$$||\mathbf{v}||_2 = \sqrt{\sum_{i=1}^{n} v_i^2}, \tag{4.25}$$

welche die euklidischen Länge des Vektors darstellt. Sie wird auch als euklidische Norm bezeichnet.
Folgende gängige Formen der Regularisierung ergeben sich:

- *Dropout*-Schichten: Deaktivierung von künstlichen Neuronen in einer sog. *dropout*-Schicht. Dabei wird die *dropout*-Rate p_{drpt} als Wahrscheinlichkeit in Prozent

definiert. Im Training werden Neuronen zufällig ausgewählt und deaktiviert, um das Modell zu zwingen, mehrere und verschiedenen Verknüpfungen zu verwenden. Ohne dieses Vorgehen besteht die Gefahr, dass einige künstliche Neuronen und deren Verknüpfungen im Modell ungenutzt bleiben [95].

- l_1-Regularisierung: Anwendung der l_1-Norm als Strafterm. Dieser stellt die Summe der Beträge der Parameter. Dabei können Parameter genau Null werden. Die Anzahl der deaktivierten Parameter kann über den Hyperparameter λ eingestellt werden. Tendenziell wird ein großer Anteil der Parameter zu Null. Das Vorgehen führt zu Modellen mit einem geringen Anteil an verwendeten Verknüpfungen [59, S. 52]. Dadurch ist es leichter zu interpretieren.
  Die Verwendung empfiehlt sich, wenn nur wenige Modelleingänge als signifikant betrachtet werden [59].
- l_2-Regularisierung: Dabei wird die euklidische Länge des Parametervektors berechnet und als Strafterm zur Gütefunktion addiert [59, S. 7]. Es drängt die Parameter in Richtung Null. Dieses Vorgehen strebt gegen ein Modell aus Konstanten [24]. Der Einfluss der Regularisierung kann über den Hyperparameter λ gesteuert werden.
- Parameterbeschränkungen: Bspw. führten Gouk, Frank, Pfahringer et al. die Erstellung Lipschitz-stetiger KNN ein [43].
- *Early stopping*: Frühzeitiger Abbruch des Trainings zur Vermeidung von *overfitting* aufgrund nachlassender Modellgüte in den Validierungsschritten [24].

Bei kleinen Datensätzen führt die Anwendung von Regularisierung meist zu einer Verbesserung der Ergebnisse [59].

4.3 Einfluss der Trainingsdaten auf die Modellgüte

Um die Modellgüte der ML-Modelle zu steigern, können verschiedene Faktoren angepasst werden. Die vorstehend beschriebenen Hyperparameter dienen der exakten Definition der Architektur der KNN. Dadurch lassen sich unter anderem Tiefe und Komplexität des Modells stark beeinflussen (vgl. Kap. 4.2). In diesem Kapitel werden weitere Einflussgrößen und ihre direkte Wirkung auf die Modellgüte untersucht. Die verfügbaren Trainingsdaten stellen einen limitierenden Faktor für die zu erreichende Modellgüte dar. Nur mit einer Datengrundlage, die den definierten Anwendungsbereich in ausreichender Menge und passender Verteilung repräsentiert, kann eine hohe Modellgüte für den vorgegebenen Anwendungsfall erreicht werden [24], [25]. Die vorhandenen Messdaten aus den Erprobungsfahrzeugen decken die gängigen Anwendungsfälle ab und können in ihrer Verteilung angepasst werden. Generell gilt für datenbasierte Methoden wie das ML, dass eine große Zahl an Datenpunkten benötigt wird, die alle Betriebspunkte und Anwendungsfälle des zu modellierenden Prozesses abbilden. Neben der Menge ist vor allem die Verteilung der Datenpunkte entscheidend.

Weitere Einflussgrößen auf die Modellgüte sind die Anzahl j der Eingangswerte des Eingangsvektors $\boldsymbol{x}$ sowie die Anzahl k der Zielgrößen im Zielvektor $\boldsymbol{y}$. Für die Anzahl der Zielgrößen wurden in dieser Arbeit pro *Fallstudie* zwei Optionen untersucht. Einerseits das Training von Modellen mit einem Modellausgang, welches ein Modell pro Zielgröße erfordert. Damit werden die Ausgangsgrößen isoliert betrachtet. Andererseits die Zusammenfassung aller Zielgrößen y_l in ein Modell, um von den Abhängigkeiten der Zielgrößen y_l untereinander zusätzliche Informationen für das Training der Modelle zu erhalten. Somit wird in dieser Arbeit zwischen Modellen mit $k = 1$ Modellausgängen und Modellen mit $k = 3$ Modellausgängen unterscheiden.
Die minimal benötigte Anzahl j der Modelleingänge x_i ist durch die zwei relevanten ADAS-Signale und das Signal der Fahrzeugmasse m_{Fzg} auf $j = 3$ festgelegt. Die maximal mögliche Anzahl an Modelleingängen ergibt sich durch das *feature engineering*. Die zusätzlich erzeugten Signale stehen ebenfalls als Eingangsgrößen x_i zur Auswahl. Somit wird die Anzahl j der Eingangswerte x_i für die vorliegenden Untersuchungen in einem Bereich von $3 \leq j \leq 10$ variiert. Die Obergrenze ist auf $j = 10$ festgesetzt, um eine sinnvolle Modellgröße für die verschiedenen Gitterstudien und Fallstudien nicht zu überschreiten. Dies begrenzt die Modellgröße und die Trainingszeiten. In begleitenden Untersuchungen in [29] wurde ermittelt, dass ein weiterer Zuwachs an zusätzlich im *feature engineering* erzeugten Signalen keinen signifikanten Mehrwert an Informationen mit sich bringt.
Aus den beschriebenen Vorüberlegungen ergibt sich die Untersuchung der in Tabelle 4.8 angegebenen Modellvariationen. Die Auswahl der Eingangsignale des ML-Modells

Tabelle 4.8: Modellvariationen betrachtet in den zwei Fallstudien

Modellvariation	Eingangsparameter	Ausgangsparameter je KNN	Anzahl Varianten
Drei Ausgänge kombiniert in einem KNN	$3 \leq j \leq 10$	$k = 3$	8
Ausgänge isoliert in drei KNN	$3 \leq j \leq 10$	$k = 1$	24

ist ebenfalls Gegenstand der Untersuchungen in diesem Kapitel. Der Informationsgehalt der unterschiedlichen Eingangssignale kann die Modellgüte steigern. Gleichzeitig vergrößert sich auch die Anzahl an Netzparametern und die Modellkomplexität.

4.3.1 Anzahl der Zielgrößen pro Modell

Zur Entwicklung eines KNN zur Berechnung des Wärmestroms $\dot{Q}_{\text{in}}$ in den Kühlkreislauf werden die einzelnen Komponenten (Wärmeströme)

$$\dot{Q}_{\text{in}} = \dot{Q}_{\text{Reib}} + \dot{Q}_{\text{Zyl}} + \dot{Q}_{\text{Oel}} \tag{4.26}$$

als Zielgrößen verwendet (vgl. Kap. 2.3). Da $\dot{Q}_{\text{in}}$ nicht direkt gemessen werden kann, werden Messgrößen wie die Kühlmitteltemperatur am Motorausgang $T_{\text{mot,aus}}$, die Motordrehzahl N_{mot} (vgl. Gl. A.1), das Motordrehmoment M_{mot} (vgl. Gl. A.3) und ein motorspezifisches Kennfeld verwendet, um $\dot{Q}_{\text{in}}$ zu ermitteln. Die modellbasierte Beschreibung dieser Wärmeströme durch die PE-Modelle wurde in [16] entwickelt und detailliert beschrieben. Die Verknüpfung der PE-Modelle zur Berechnung von $\dot{Q}_{\text{in}}$ wird in Kapitel 4.4.1 dargelegt. Das Training erzeugt KNN, welche die Wechselwirkungen und Abhängigkeiten des Fahrzeugantriebsstrangs abbilden. Durch eine kontinuierliche Erweiterung der Datenbasis anhand von Fahrzeugmessungen im Laufe des Entwicklungszyklus, können Veränderungen an Hard- und Software aufgezeichnet werden und in das Training einfließen. Das aufwändige, manuelle Kalibrieren von Kennfeldern und Kennlinien ist für diesen Anwendungsfall nicht mehr notwendig. Weitere Vorteile der Nutzung datenbasierter Modelle und der Datensammlung in der Cloud ergeben sich aus Kapitel 2.3.

4.3.2 Eingangssignale für Training und Anwendung

Für die Eingangsgrößen des Modells muss zwischen dem Training und der Anwendung unterschieden werden. Im Training werden fein aufgelöste Messdaten (Abtastrate 10 Hz) aus dem Fahrzeug verwendet. In der Anwendung werden grob aufgelöste Vorausschaudaten $\boldsymbol{x}_{\text{HRC}}$ aus dem HRC eingespielt (vgl. Kap. 2.2.3 und 3.1). Das trainierte ML-Modell soll den Wärmestrom $\dot{Q}_{\text{in}}$ in die Kühlflüssigkeit errechnen (vgl. Kap. 1.2). Dieser resultiert aus den Anforderungen der Fahrtstrecke an den Fahrzeugantriebsstrang. Die bisher verwendeten PE-Modelle berechnen die dafür benötigte Leistung des Antriebsstrangs, um die prädizierte Fahrtstrecke in prädizierter Geschwindigkeit v_{lim} zurückzulegen (ADAS-Daten). Die errechnete Leistung kann in den erwarteten Wärmestrom $\dot{Q}_{\text{in}}$ in die Kühlflüssigkeit übersetzt werden. Um $\dot{Q}_{\text{in}}$ mittels ML-Modellen zu berechnen, wird in Kapitel 4.4 ein geeignetes Vorgehen zur Substitution der PE-Modelle erarbeitet und definiert.
Die Auflösung der Daten aus dem HRC ist im Vergleich zu Fahrzeugmessdaten sehr grob (vgl. Kap, 3.1). Vor allem die Geschwindigkeit das Fahrzeugs kann nur in groben Klassen, den Geschwindigkeitslimits, angegeben werden. Im Vergleich dazu liefern die Fahrzeugmessdaten mehrmals pro Sekunde die aktuelle Fahrzeuggeschwindigkeit. Dazu wurde in [16] eine Verfeinerung durchgeführt, die es erlaubt, anhand der Straßenklasse, der Kurvenradien und des aktuellen Fahrverhaltens eine genauere

Schätzung der erwarteten Fahrzeuggeschwindigkeit anzugeben. Zusätzlich wird in der vorliegenden Arbeit die aktuelle Fahrzeugmasse m_{Fzg} verwendet, welche im Messfahrzeug als Signal vorliegt. Für den Vorausschauvektor $\boldsymbol{x}_{\mathrm{HRC}}$ aus dem HRC kann die aktuelle Masse m_{Fzg} als konstant für die nächsten 200 Einträge des Vorausschauvektors, geschätzt werden. Da der Vorausschauvektor $\boldsymbol{x}_{\mathrm{HRC}}$ im Steuergerätetakt von 100 Hz neu berechnet wird und für jeden Zyklus ebenfalls die aktuelle Fahrzeugmasse m_{Fzg} vorliegt, ist eine ausreichende Genauigkeit für diese Größe gewährleistet. Selbst nach einer abrupten Änderung bspw. nach einer Standphase durch Tanken oder Anhängen einer Anhängelast wird die neue Fahrzeugmasse nach wenigen Rechenschritten aktualisiert sein. Diese kurze Verzögerung gründet auf der Art und Weise der Berechnung der Fahrzeugmasse. Diese basiert auf Beschleunigungs- und Widerstandswerten des Fahrzeugs während der Fahrt [16].
Der Vorausschauvektor $\boldsymbol{x}_{\mathrm{HRC}}$ liefert Signale, welche deutlich gröber aufgelöst sind als die Trainingsdaten. Im Speziellen bedeutet das, dass Signale wie die erlaubten Geschwindigkeitslimits v_{lim} nur in den bekannten Abstufungen (30 km/h, 50 km/h, 70 km/h, ...) vorhanden sind [103].

4.4 PE-Modelle und deren Substitution

In den *Fallstudien I & II* (vgl. Kap. 1.2.3) werden zwei unterschiedliche KNN-Architekturen (vgl. Kap. 4.1.2) untersucht, welche die derzeit im prädiktiven TM verwendeten PE-Modelle ersetzen sollen. Die PE-Modelle des prädiktiven TM wurden bisher in *Matlab Simulink* modelliert und mit Hilfe einer standardisierten Tool-Kette in Steuergerätecode übersetzt. Die ML-Modelle werden hingegen mit der ML-Pipeline in *Python*-Code erstellt. Daher liegt keine gemeinsame Plattform zum automatisierten Vergleich der gegebenen PE-Modelle mit einer großen Anzahl an ML-Modellen vor. Um die Modellgüte hardwareunabhängig und automatisiert vergleichen zu können, wurden die in *Matlab Simulink* vorliegenden PE-Modelle im Zuge dieser Arbeit exakt in der Programmiersprache *Python* nachmodelliert [29]. Dafür wurden alle relevanten Fahrwiderstandsgleichungen, Kennfelder, Kennlinien sowie die physikalischen und mathematischen Zusammenhänge aus dem *Matlab Simulink*-Modell übernommen. Die in *Python* nachmodellierten PE-Modelle wurden anschließend, anhand von aufgezeichneten Messdaten, mit den PE-Modellen in *Matlab Simulink* aus [16] verglichen. So wurde sichergestellt, dass beide Implementierungen dieselben Ergebnisse ausgeben. Bei Verwendung derselben Eingangsdaten und Datentypen (*Matlab* Single-Datentyp *float32*) konnte dieselbe Genauigkeit erreicht werden. Dadurch, dass sowohl die PE-Modelle als auch die ML-Modelle in *Python*-Code vorliegen ist es möglich, einen direkten Vergleich der Modellgüte für die Ergebnisse der *Fallstudien* durchzuführen. Dieser Bewertungsschritt kann in die ML-Pipeline integriert sowie hardwareunabhängig in der Cloud ausgeführt werden und ist entkoppelt von Erprobungsträgern und Steuergeräten.

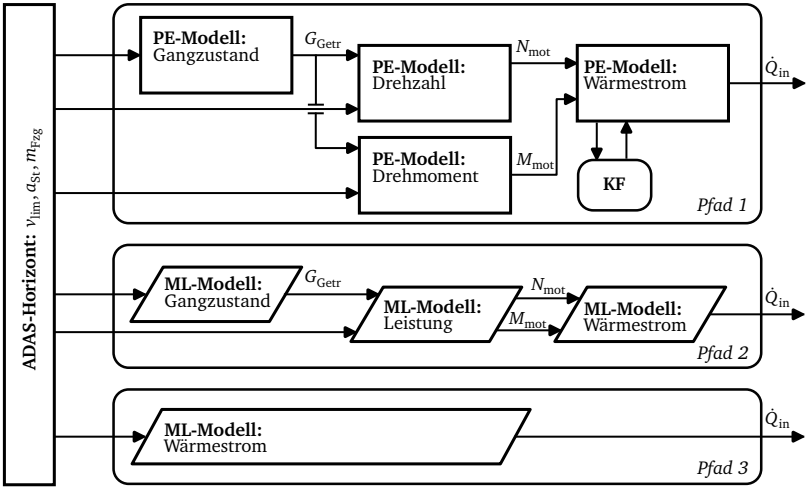


Abbildung 4.5: *Pfad 1*: Schematische Darstellung der Verknüpfung der PE-Modelle und Kennfelder (KF) zur Berechnung des Wärmestroms in den KKL.
Pfad 2: Substitution der einzelnen PE-Modelle durch mehrere ML-Modelle.
Pfad 3: Substitution aller PE-Modelle durch ein ML-Modell.

4.4.1 Verknüpfung der PE-Modelle

Die Regelung der Kühlmitteltemperatur im HT-KKL basiert maßgeblich auf der Kühlmitteltemperatur am Motorausgang vor dem Thermostat $T_{mot,aus}$. Die PE-Modelle des prädiktiven TM errechnen den voraussichtlichen Wärmestrom in den Motorkühlkreislauf $\dot{Q}_{in}$ aus den ADAS-Vorausschaudaten (vgl. *Pfad 1* in Abb. 4.5). Anhand des errechneten Wärmestroms $\dot{Q}_{in}$ wird nachfolgend die zur Regelung nötige Temperatur $T_{mot,aus}$ ermittelt. Die Temperatur $T_{mot,aus}$ ist direkt abhängig vom Betriebspunkt des Verbrennungsmotors und der darin stattfindenden Verbrennungsprozesse. Die Regelung der Kühlmitteltemperatur am Motorausgang zielt darauf ab, den Verbrennungsmotor stets in seinem optimalen Betriebspunkt zu halten. Ein optimaler Betriebspunkt ermöglicht den geringsten Kraftstoffverbrauch und Schadstoffausstoß sowie die Reduzierung von Reibung und Verschleiß.
Die PE-Modelle des prädiktiven TM zur Berechnung des Wärmestroms $\dot{Q}_{in}$ stellen durch ihre Verknüpfung ein grobes Ersatzmodell des Fahrzeugantriebsstrangs dar. Auf diese Weise errechnen einzelne PE-Modelle beispielsweise den Gangzustand des Getriebes G_{Getr}, das Motordrehmoment M_{mot} und die Motordrehzahl N_{mot}. Der Getriebegang

G_{Getr} wird zur Berechnung der Motorleistung in Form von Drehmoment M_{mot} und Drehzahl N_{mot} verwendet (vgl. Abb. 4.5). Der Wärmestrom in die Kühlflüssigkeit $\dot{Q}_{\text{in}}$ wird im Anschluss daran anhand eines $3D$-Motorkennfelds ermittelt. Eingangsgrößen sind die Motordrehzahl N_{mot} und der effektive Mitteldruck p_{me}, welcher sich direkt aus dem Motordrehmoment M_{mot} berechnen lässt [14], [16]. Das Kennfeld liefert für diese Eingangswertekombinationen die gesamt entwickelte Motorabwärme $\dot{Q}_{\text{mot}}$. Das $3D$-Motorkennfeld ist ein Wärmebilanzkennfeld und stellt den empirisch auf dem Motorenprüfstand ermittelten Wärmestrom des Motors $\dot{Q}_{\text{mot}}$ dar [16]. Aus der Größe $\dot{Q}_{\text{mot}}$ werden die einzelnen Wärmeströme in den KKL ermittelt, welche nach Gleichung 4.26 den gesamten Wärmestrom in den KKL $\dot{Q}_{\text{in}}$ ergeben [16]. Eine detaillierte Beschreibung und Herleitungen der PE-Modelle kann [16] und der einschlägigen Fachliteratur entnommen werden.
Da im Falle der prädiktiven Anwendung der PE-Modelle nur grobe Informationen zur Fahrstrecke, jedoch keine Gangzustände vorliegen, müssen diese separat berechnet werden, um anschließend als Modelleingang zur Verfügung zu stehen. Der Gangzustand wird im prädiktiven TM derzeit mit Hilfe einer Kennlinie abgeschätzt [16]. Die Stützstellen dieser Kennlinie müssen für jeden Antriebsstrang und die verwendete Kombination aus Verbrennungsmotor und Getriebe empirisch ermittelt werden. Die erreichbare Genauigkeit ist vor allem durch die groben Geschwindigkeitsklassen aus dem ADAS-Horizont (vgl. Kap. 2.2.3) limitiert [29].

4.4.2 Methodik zur Substitution der PE-Modelle durch KNN

In Abbildung 4.5 werden zwei Möglichkeiten zur Substitution der bisher verwendeten PE-Modelle durch ML-Modelle dargestellt. Beide *Pfade* werden nachfolgend beschrieben. Der in dieser Arbeit verwendete *Pfad* zur Berechnung des Wärmestroms $\dot{Q}_{\text{in}}$ in den KKL wird ausgewählt.

Pfad 2 - ML-Modelle zur Berechnung der Motorleistung In begleitenden Untersuchungen zu dieser Arbeit wurden Modelle zur Berechnung der Motorleistung in Form von Motordrehmoment M_{mot} und Motordrehzahl N_{mot} durchgeführt [29], [103]. Dieses Vorgehen entspricht *Pfad 2* in Abbildung 4.5 und zielt darauf ab, die vorgegebene Struktur aus dem prädiktiven TM zu erhalten. Lediglich die einzelnen PE-Modelle und Kennfelder werden durch ML-Modelle ersetzt. Untersuchungen in [29] haben gezeigt, dass weder die PE-Modelle noch die ML-Modelle aus den verfügbaren ADAS-Horizontdaten eine hohe Genauigkeiten in der Schätzung des Gangzustands errechnen. Wie beschrieben, ist die Modellierung mittels ML-Modellen durch die grobe Informationslage in ihrer Genauigkeit begrenzt. Dennoch konnten KNN zu Gangklassifikation erstellt werden, welche die kennfeldbasierten Vorhersagen in ihrer Genauigkeit (engl. *accuracy*) und Präzision (engl. *precision*) übertrafen. Zudem bieten sie den Vorteil der automatisierten Anpassbarkeit. Im weiteren Verlauf wurden ML-Modelle untersucht,

welche entsprechend zu *Pfad 2*, aus dem Gangzustand G_{Getr} und den Eingangsdaten aus dem ADAS-Horizont die Berechnung von M_{mot} und N_{mot} durchführen [29], [103]. Dies bietet den Vorteil, dass hardwareabhängige Modell-Konstanten bei einer Veränderung der Hardware im Antriebsstrang nicht manuell angepasst werden müssen, sondern durch das Aufzeichnen aktueller Messdaten mit in das Training einfließen. Ein Beispiel hierfür ist eine Änderungen der Getriebehardware während des Entwicklungszyklus des Fahrzeugs. Um diese Änderung im PE-Modell widerzuspiegeln, muss unter anderem die Getriebeübersetzung (im Modell eine Konstante vgl. Anhang A) manuell angepasst werden.
Die Berechnung von Drehmoment und Drehzahl durch KNN liefert Ergebnisse, die daraufhin von einem weiteren KNN verarbeitet werden, um damit die Wärmeströme zu berechnen (vgl. *Pfad 2* in Abb. 4.5). Die Auswahl der Zielgrößen wird, wie in Tabelle 3.2 gezeigt, durchgeführt. Die Untersuchung von *Pfad 2* in [29], [103] ergab keine signifikante Steigerung in der Genauigkeit im Vergleich zu den PE-Modellen, da die Unsicherheiten in den ADAS-Horizontdaten und die Fehlerfortpflanzung zwischen den einzelnen Modellen ähnliche Auswirkungen hat wie bei der Verknüpfung der physikalisch motivierten Systeme.

Pfad 3 - Ein ML-Modell zur direkten Berechnung des Wärmestroms Im Kontrast zur Modellierung des Gangzustands, der Motordrehzahl und des Drehmoments durch mehrere ML-Modelle (vgl. *Pfad 2* in Abb. 4.5) werden in *Pfad 3* alle PE-Modelle durch ein ML-Modell ersetzt. Es zeigt sich, dass bei direkter Berechnung des Wärmestroms $\dot{Q}_{\text{in}}$ deutlich bessere Ergebnisse zu erzielen sind als durch Kombination von mindestens drei Untermodellen [29], [103]. Die Verwendung eines ML-Modells zur Berechnung der Komponenten des Wärmestroms hat deutliche Verbesserungen in der Genauigkeit erzeugt (vgl. Kap. 6.2). Die Implementierung nur eines ML-Modells, wie in *Pfad 3* dargestellt, bringt mehrere Vorteile mit sich:

- Die Berechnung von $\dot{Q}_{\text{in}}$ ist ohne Zwischenschritte möglich.
- Die Anzahl der benötigten Modelle und Komplexität des Systems wird reduziert.
- Die in der Zielstellung (vgl. Kap. 1.2) geforderte Automatisierung der Modellerstellung ist gewährleistet und lässt sich in die ML-Pipeline integrieren.

Aus diesem Grund wird in dieser Arbeit die Substitution aller PE-Modelle durch ein ML-Modell, wie in *Pfad 3* (vgl. Abb. 4.5) dargestellt, vorgenommen. Dadurch werden die beschriebenen Vorteile genutzt, um eine möglichst hohe Modellgüte mit einem weniger komplexen Gesamtsystem zu erzielen. Die Fehlerfortpflanzung zwischen den einzelnen Modellen (wie in *Pfad 2* möglich) entfällt, da nur ein Modell verwendet wird.
In Kapitel 6 werden die Ergebnisse der *Fallstudien I & II* hinsichtlich ihrer Modellgüte mit den PE-Modellen verglichen. Die Substitution der PE-Modelle ermöglicht eine automatisierte Modellbildung in der ML-Pipeline. Zudem können die Modelle mittels neuer Messdaten an neue Hard- und Softwarestände angepasst werden.

4.4.3 Beschreibung der Gitterstudien

In den Gitterstudien werden jeweils über 20.000 unterschiedliche HPSs erstellt und die daraus resultierenden ML-Modelle evaluiert. Die in Kapitel 4.2.1 beschriebenen Verfahren zur Bewertung der Modellgüte dienen dabei dazu, das beste Modell automatisiert auszuwählen und als Ergebnis der ML-Pipeline auszugeben. Die Anzahl an HPSs für die Gitterstudien ergibt sich aus der Kombination der Tabellen 4.7 und 4.8. Wie im Kapitel 4.3 beschrieben, wird jedes HPS der Gitterstudie 10 mal ausgeführt. Die Modellgüte hängt neben den Hyperparametern Neuronenzahl n_n, Anzahl verdeckter Schichten n_h entscheidend von der Netzarchitektur ab. Ein Vergleich zwischen FNN und komplexeren RNN mit Ergebnisrückführung ist einer der Schwerpunkte dieser Arbeit. Aus den gewonnene Erkenntnissen in diesem Kapitel bilden sich weitere Entwicklungsschritte wie z.B. die Aktualisierung der Modelle auf dem Fahrzeug aufgrund einer neuen Datenbasis im Cloud-Backend. Jedes Modell wird dafür mit demselben Testdatensatz $\mathbb{S}_{\text{test}}$ überprüft, so kann eine Steigerung der Modellgüte automatisiert gewährleistet werden. Die Plausibilitätsanalyse der Modellberechnungen (vgl. Kap. 5) liefert in der Anwendung eine weitere Absicherung der Modellausgänge.

5 Entwicklung der rückverknüpften Plausibilitätsanalyse für ML-Modelle

Um die Modellberechnungen der erzeugten ML-Modelle auf ihre Plausibilität hin überprüfen zu können, wird in diesem Kapitel die rückverknüpfte Plausibilitätsanalyse (RPA) entwickelt. Sie schafft die Möglichkeit, den Einsatz von datenbasierten Modellen mit einem effiziente Algorithmus zur Plausibilisierung der Modellausgänge zu kombinieren. Dazu wurde ein Verfahren entwickelt, um zulässige Ausgangsdatenräume $\mathcal{D}_{\text{aus,sub}}$ zurück auf zugehörige Eingangsdatenräume $\mathcal{D}_{\text{bw,sub}}$ zu verknüpfen. Dadurch entstehen die sogenannten BRSs. Diese werden mittels OCSVMs erfasst. Damit kann eine einfache und rechenzeitgünstige Klassifizierung neuer Datenpunkte durchgeführt. Die verwendeten mathematischen Beziehungen und die Entwicklung der gesamten Methodik der RPA werden in diesem Kapitel beschrieben. Der *black-box* Charakter von ML-Modellen schränkt deren Anwendung im Fahrzeug derzeit in vielen Systemen ein (vgl. Kap. 4.1.1). Im Vergleich dazu sind bei klassischen *white-box* Modellen wie Kennfeldern, Kennlinien oder PE-Modellen die zugrundeliegenden physikalischen Zusammenhänge bekannt und ersichtlich. Kennfelder und Kennlinien liefern Werte basierend auf Interpolationen innerhalb der Systemgrenzen. Durch fehlerhafte Annahmen, Algorithmen oder Ermittlung der Stützstellen können auch diese mit Fehlern behaftet sein. Eine Überwachung der Ergebnisse erfolgt in der Regel über die Definition eines zulässigen Wertebereichs für die Modellberechnungen. Da die Beziehung zwischen Modelleingang und -ausgang für ein *black-box* Modell nicht generell als stetig betrachtet werden kann, ist die Definition eines zulässigen Wertebereichs alleine für ML-Modelle im PKW-TM nicht ausreichend. Daher wird in dieser Arbeit eine Möglichkeit zur Plausibilisierung der ML-Modelle während der Inferenz entwickelt. Die daraus resultierende Problemstellung wird in Kapitel 5.1 dargelegt.
Um Modellberechnungen von ML-Modellen auf ihre Plausibilität zu bewerten, werden Methoden zur Plausibilisierung entwickelt. Diese werden mit Vorverarbeitungsschritten zu Datenaufbereitung kombiniert. Die entwickelte RPA ermöglicht es, die Modellberechnungen während der Inferenz zu bewerten. Dabei verknüpft die RPA die in Kapitel 3 vorgestellten Methoden zur Eingrenzung und Unterteilung des Trainingsdatenraums mit den in diesem Kapitel eingeführten Methoden zur Bewertung der Modellberechnungen. Grundlegende Untersuchungen dazu wurden in [104] publiziert. Die RPA basiert auf Algorithmen zur Erzeugung von validen Ausgangsdatenräumen $\mathcal{D}_{\text{aus}}$. Diese werden mit zugehörigen Eingangsdatenräumen $\mathcal{D}_{\text{ein}}$ verknüpft und ergeben die BRSs $\mathcal{D}_{\text{bw,sub}}$. Die Datenraumbegrenzung wird mittels OCSVMs automatisiert ermittelt. Für

das *hyperparameter tuning* der OCSVM wird in Kapitel 5.2.2 eine Gitterstudie durchgeführt.
Um die Modellberechnungen plausibilisieren zu können, wird der gegebene Trainingsdatenraum $\mathcal{D}_{\text{ges}}$ in Unterdatenräume $\mathcal{D}_{\text{sub}}$ zerlegt. In Kapitel 5.2 werden die verschiedenen Verfahren zur Definition und Einteilung der Unterdatenräume $\mathcal{D}_{\text{sub}}$ diskutiert.
Jeder dieser Unterdatenräume wird mittels einer OCSVM mit einer Hyperebene umhüllt. Diese als EG (vgl. Kap. 3.3.2) bezeichnete Datenraumbergrenzungen der einzelnen Unterdatenräume wird verwendet, um Datenpunkte als *innerhalb* oder *außerhalb* der EG zu klassifizieren. So kann überprüft werden, ob die Modellberechnung im erwarteten Unterdatenraum $\mathcal{D}_{\text{aus,sub}}$ liegt. Um eine Beziehung zwischen Ausgangsdatenraum $\mathcal{D}_{\text{aus,sub}}$ und BRS $\mathcal{D}_{\text{bw,sub}}$ voraussetzten zu können, muss das ML-Modell die Modelleingänge mittels einer stetigen Funktion auf die Modellausgänge projizieren. Ein stetiges Verhalten der ML-Modelle kann nicht pauschal angenommen werden. Die Absicherung bzw. Plausibilisierung basiert auf dem Nachweis der Lipschitz-Stetigkeit für FNN. Dazu wird in Kapitel 5.3 deren Lipschitz-Stetigkeit untersucht. Entsprechendes kann in eingeschränktem Maße für RNN nachgewiesen werden. Beide Fälle können somit in den *Fallstudien* berücksichtigt werden.
In Kapitel 5.4 wird der Prozess der RPA zusammengesetzt und automatisiert. Mit den entwickelten Methoden können die Modellberechnungen als plausible oder implausibel klassifiziert werden. Soweit die Modellberechnung als implausibel klassifiziert wird, wird eine Ersatzwertbildung durchgeführt, welche ebenfalls in diesem Kapitel eingeführt wird. Die Methodik der RPA kann in vier Schritte unterteilt werden:

- Datenvorverarbeitung und *feature engineering* zur Erstellung des Trainingsdatenraums $\mathcal{D}_{\text{ges}}$ (vgl. Kap. 3)
- Unterteilung in Unterdatenräume $\mathcal{D}_{\text{sub}}$
- Berechnung der EG (via OCSVM) und weiterer kritischer Eigenschaften
- Plausibilitätsbewertung der Modellberechnungen

In Kapitel 5.4 wird die entwickelte Methodik auf die Ergebnisse der beiden Fallstudien angewendet.

5.1 Problemstellung in Bezug auf datenbasierte Modelle

Derzeit basieren die Regelungsalgorithmen im TM auf Kennfeldern und Kennlinien bzw. auf physikalisch motivierten Regelungsmodellen für die Umsetzung modellbasierter Regelungsverfahren [15], [27]. Sie können als mathematische Funktion dargestellt werden. Somit ist ihr Definitionsbereich eindeutig beschreibbar. Dadurch ist eine Überprüfung der Modellausgänge möglich. Die im Folgenden entwickelte RPA zeigt eine Möglichkeit auf, wie der Einsatz von ML-Modellen abgesichert werden kann. Sie stellt einen Forschungsansatz dar, um eine Plausibilisierung der Modellberechnungen durchzuführen.

5.1.1 Übersicht über Absicherungsverfahren

Es existieren Untersuchungen zur Unsicherheitsabschätzung (engl. *uncertainty estimation*) von ML-Modellen. Diese Methoden sind rechenzeitintensiv und daher nicht für die Anwendung während der Laufzeit (Inferenz) eines ML-Modells auf dem Steuergerät geeignet [96]. Parallel dazu ermöglichen die *out-of-distribution*-Methoden eine Fehlererkennung, die zum Beispiel eine *outlier*-Erkennung beinhaltet. Diese erkennt Modelleingänge, die außerhalb des im Training verwendeten Eingangsdatenraums $\mathcal{D}_{\text{ein}}$ liegen, oder Modellberechnungen, die außerhalb des im Training verfügbaren Ausgangsdatenraums $\mathcal{D}_{\text{aus}}$ liegen. Dafür kommen Methoden wie SVM, OCSVM oder *convex-hull*-Algorithmen zum Einsatz.
Ein weiterer Ansatz ist die Verwendung sogenannter selektiver Klassifikation (engl. *selective classification*) zur Erkennung von Modellfehlern [96]. Sie sind nach [96] der *in-distribution*-Fehlerdetektion zuzuordnen. Die in diesem Kapitel entwickelte RPA ermöglicht sowohl eine *out-of-distribution*-Erkennung als auch die selektive Klassifikation von *in-distribution*-Fehlern.
Die Anwendung im prädiktiven TM bietet zudem eine abgesicherte Ausführung der ML-Modelle. Sollten die Modelle in der Anwendung implausible Werte liefern, wird lediglich die Prädiktion des erwarteten Wärmestroms $\dot{Q}_{\text{in}}$ beeinträchtigt. Ist keine Ersatzwertbildung für implausible Werte möglich (vgl. Kap. 5.4.3), kann die generell verfügbare Rückfallebene des prädiktiven TM verwendet werden. Diese Rückfallebene berücksichtigt beispielsweise die Möglichkeit, dass während des Fahrzeugbetriebs keine ADAS-Vorausschaudaten verfügbar sind. In diesem Fall muss das konventionelle, kennfeldbasierte TM des Fahrzeugs ohne Prädiktion funktionieren [16].

5.1.2 Auswahl und Weiterentwicklung aktueller Methoden

Um eine Methodik zur Plausibilisierung von ML-Modellausgängen zu entwickeln, werden die in Kapitel 3 beschriebenen Methoden zur automatisierten Datenraumbeschreibung mittels OCSVM verwendet. Das in Kapitel 3 beschriebene Verfahren wird um weitere Arbeitsschritte ergänzt:

- die optimale Einteilung der Unterdatenräume $\mathcal{D}_{\text{sub}}$,
- die Berechnung kritischer Parameter für jeden $\mathcal{D}_{\text{sub}}$,
- die Überprüfung der Lipschitz-Stetigkeit für die verwendeten FNN.

Der Einsatz von OCSVMs ermöglicht eine effiziente Klassifizierung der Modellberechnungen $\hat{\boldsymbol{y}}$ zur Einteilung in die verfügbaren Unterdatenräume. Dabei haben OCSVM im Vergleich zu anderen Klassifizierungsmethoden einen geringen Bedarf an Speicherplatz und Rechenleistung. Die komplexe Erstellung der Hyperebene erfolgt im Training und kann in der Anwendung mit Hilfe des *kernel tricks* durch einfache Vektoroperationen effizient durchgeführt werden [62], [64].
Mit OCSVM kann eine beliebig dimensionale Punktewolke im Raum beschrieben

werden, unabhängig von deren Form. Während konkave Datenräume oft nur schwer manuell durch eine mathematische Formel beschrieben werden können, leisten dies die OCSVMs unabhängig von der Anordnung der Punktewolke. Zudem können auch Löcher innerhalb der Punktewolke erkannt und ausgespart werden. Dazu sind die Hyperparameter für das Training der OCSVM sorgfältig zu wählen (vgl. Kap. 5.2.2). Das Ziel ist es, die EG um die Datenpunkte möglichst eng zu definieren. Die Form der EG stellt immer einen Kompromiss zwischen einer möglichst genauen Kontur der Punktewolke und dem Ignorieren von Ausreißern in der Verteilung dar.
Zur Unterteilung des Trainingsdatenraums $\mathcal{D}_{\text{ges}}$ in Unterdatenräume $\mathcal{D}_{\text{sub}}$ werden zwei Methoden verglichen:

- Einteilung der Wertebereiche der Zielgrößen $\boldsymbol{y}$ in gleichmäßige Intervalle,
- Unterteilung der Trainingsdaten anhand ihrer statistischen Verteilung in Quantile.

In Kapitel 5.2.1 werden daraus sogenannte BRSs erzeugt. Diese verknüpfen Unterdatenräume $\mathcal{D}_{\text{aus,sub}}$ des Ausgangsdatenraums $\mathcal{D}_{\text{aus}}$ mit Unterdatenräumen $\mathcal{D}_{\text{bw,sub}}$ des Eingangsdatenraums $\mathcal{D}_{\text{ein}}$. Letztere Unterdatenräume $\mathcal{D}_{\text{bw,sub}}$ werden als BRSs bezeichnet. Modellberechnungen $\hat{\boldsymbol{y}}$ können direkt einem $\mathcal{D}_{\text{bw,sub}}$ zugeordnet werden.
Um diese Zuordnung zu plausibilisieren, wird in Kapitel 5.3 die Lipschitz-Stetigkeit der FNN überprüft. Ist diese gegeben, kann von einer Änderung der Eingangsgröße einer Funktion $f(\boldsymbol{x})$ auf eine entsprechende Änderungen der durch $f(\boldsymbol{x}) = \hat{\boldsymbol{y}}$ berechneten Ausgangsgröße ausgegangen werden. Somit stehen alle benötigten Methoden zur Überprüfung und Plausibilisierung der Modellausgänge von datenbasierten Modellen (im Speziellen FNN) zur Verfügung.

5.2 Unterteilung der Datenbasis

5.2.1 Verfahren zur Einteilung des Datenraums in Unterräume

Die entwickelte RPA überprüft, ob der vom Modell berechnete Ausgang $\hat{\boldsymbol{y}}$ innerhalb des durch die Trainingsdaten definierten Ausgangsdatenraums $\mathcal{D}_{\text{aus}}$ liegt. Dieser wird von allen im Training verfügbaren Zielgrößen $\boldsymbol{y}$ aufgespannt. Seine Dimension richtet sich nach der Anzahl k an Ausgangssignalen des Modells

$$\mathcal{D}_{\text{aus}} \in \mathbb{R}^k. \tag{5.1}$$

Falls die Modellberechnung nicht innerhalb des Ausgangsdatenraums liegt und somit

$$\hat{\boldsymbol{y}} \notin \mathcal{D}_{\text{aus}} \tag{5.2}$$

gilt, kann die Modellberechnung direkt als implausibel definiert werden. Dieses Vorgehen ist konservativ motiviert und schließt mögliche korrekte Extrapolationen für

unbekannte Eingangswertekombinationen aus. Sollte der berechnete Wert innerhalb von $\mathcal{D}_{\text{aus}}$ liegen, so ist folgende Bedingung erfüllt:

$$\hat{\boldsymbol{y}} \in \mathcal{D}_{\text{aus}}. \tag{5.3}$$

Dies reicht noch nicht, um eine Berechnung als plausibel zu bewerten. Der Grund dafür ist, dass fehlerhafte Modellberechnungen im Ausgangsdatenraum der Trainingsdaten liegen können. Dieser enthält nahezu alle Ausgangswerte(-kombinationen), die im Training verwendet wurden. Für eine Plausibilisierung ist es essentiell, die Modelleingänge $\boldsymbol{x}$ zu berücksichtigen, auf welche das Modell zur Bestimmung von $\hat{\boldsymbol{y}}$ angewendet wurde. Um die Lage der Modellberechnungen $\hat{\boldsymbol{y}}$ im Raum enger mit den zugehörigen Eingangsdaten $\boldsymbol{x}$ zu verknüpfen, werden BRSs erstellt. Darunter versteht man Datensätze, die ausgehend von der Modellberechnung $\hat{\boldsymbol{y}}$ rückwärts durch das Modell als zugehörigen Eingangsdaten $\boldsymbol{x}$ verknüpft werden können. Mit diesem Verfahren kann jedem Zielvektor $\boldsymbol{y}$ des Trainingsdatensatzes ein zugehöriger (rückwärts erreichbarer) Eingangsvektor $\boldsymbol{x}$ zugeordnet werden. Da eine 1 : 1-Beziehung für jeden Datenpunkt genau eine Verknüpfung liefert, wäre eine Überprüfung jeweils aller Trainingsdatenpunkte notwendig. Falls die Kombination aus $\boldsymbol{y}$ und $\boldsymbol{x}$ im Training nicht vorhanden war, müsste die Berechnung als implausibel markiert werden. Dieses Vorgehen erlaubt keine Interpolation und Generalisierung des Netzes. Zudem ist es unrealistisch und nicht das Ziel eines ML-Modells, den gesamten Trainingsdatensatz zur Plausibilisierung der Modellausgänge heranzuziehen.
Aus diesem Grund wird im vorliegenden Kapitel ein Verfahren entwickelt, um den Ausgangsdatenraum $\mathcal{D}_{\text{aus}}$ in Unterdatenräume $\mathcal{D}_{\text{aus,sub}}$ zu unterteilen. Die damit verknüpften Punktewolken der Eingangsvektoren $\boldsymbol{x}$ werden als BRSs $\mathcal{D}_{\text{bw,sub}}$ bezeichnet [61]. Es gilt

$$\mathcal{D}_{\text{bw,sub}} \subseteq \mathcal{D}_{\text{ein}}; \tag{5.4}$$

$$\mathcal{D}_{\text{aus,sub}} \subseteq \mathcal{D}_{\text{aus}}. \tag{5.5}$$

Dadurch wird eine Kategorisierung eingeführt, die es ermöglicht, umhüllende Hyperebenen für jeden Unterdatenraum zu erstellen. Dafür werden OCSVMs verwendet. Diese liefern nach dem Training eine EG, welche die Datenpunkte des Unterdatenraums repräsentiert. Diese kann unter geringem Speicherbedarf und mit geringer Rechenkomplexität zur Klassifikation von Datenpunkten angewendet werden. Um Modellausgänge $\hat{\boldsymbol{y}}$ als plausibel oder implausibel zu klassifizieren, werden BRSs verwendet. Jedem Unterdatenraum $\mathcal{D}_{\text{aus,sub}}$ ist ein $\mathcal{D}_{\text{bw,sub}}$ zugeordnet. Dabei ist ein BRS $\mathcal{D}^{c}_{\text{bw,sub}}$ eines Unterdatenraums $\mathcal{D}^{c}_{\text{aus,sub}}$ durch alle Eingangsvektoren $\boldsymbol{x}$ des Eingangsdatenraums $\mathcal{D}_{\text{ein}}$ definiert, welche mit den Zielvektoren $\boldsymbol{y}$ aus $\mathcal{D}_{\text{aus,sub}}$ als ein Datenpunkt (vgl. Kap. 3.1) aufgezeichnet wurden. Bspw. die Fahrzeuggeschwindigkeit v_{Fzg} als Eingangssignal x_i und der Wärmestrom in den Zylinderkopf des Motors $\dot{Q}_{\text{Zyl}}$ als Zielwert y_l zum selben Zeitpunkt der Messung aufgezeichnet. Dabei ist der Ausgangsdatenraum $\mathcal{D}_{\text{aus}}$ unterteilt

in N Unterdatenräume $\mathcal{D}_{\text{aus,sub}}$, für die gilt,

$$c = [1, N] \quad \forall c \in \mathrm{N}. \tag{5.6}$$

Die Variable c beschreibt einen der erstellten N Unterdatenräume. Sie beeinflusst direkt die Anzahl $n_{\text{ein,c}}$ an rückwärts erreichbaren Eingangsvektoren $\boldsymbol{x}$ pro Unterdatenraum $\mathcal{D}^{c}_{\text{aus,sub}}$. Mit wachsendem Wert für N verringert sich die Anzahl an Datenpunkten pro Unterdatenraum. Die erstellten BRSs $\mathcal{D}_{\text{bw,sub}}$ stellen eine Punktemenge in einem j-dimensionalen Datenraum dar. Dabei ist j die Anzahl der Eingangssignale des Modells (vgl. Kap. 3). Um eine Berechnung des ML-Modells zu plausibilisieren, muss der Modellausgang $\hat{y}_l$ einem der N Unterdatenräume $\mathcal{D}_{\text{aus,sub}}$ zugeordnet werden können. Ist das der Fall, muss überprüft werden, ob der verwendete Eingangsvektor $\boldsymbol{x}$ im verknüpften BRS $\mathcal{D}_{\text{bw,sub}}$ liegt. Um dies mit geringer Rechenkomplexität und kleinem Speicherbedarf online durchführen zu können, wird für jedes BRS eine OCSVM trainiert. Diese liefert eine EG für den Unterdatenraum und kann Datenpunkte als *innerhalb* oder *außerhalb* liegend klassifizieren. Für die Unterteilung des Ausgangsdatenraums $\mathcal{D}_{\text{aus}}$ muss ein Kompromiss zwischen feiner Auflösung der unterschiedlichen Betriebspunkte und der Anzahl an Datenpunkten pro Unterdatenraum $\mathcal{D}^{c}_{\text{aus,sub}}$ gefunden werden. Dies wird in Kapitel 5.2.2 genauer beschrieben.
Im Folgenden werden zwei Verfahren zur Unterteilung der Datenräume erläutert. Durch die **Verwendung äquidistanter Intervalle** wird der Wertebereich des Ausgangsdatenraums $\mathcal{D}_{\text{aus}}$ in N Intervalle (engl. *bins*) unterteilt. Ausschlaggebend hierfür sind die Minima und Maxima aller Zielgrößen $\boldsymbol{y}$. Hierbei müssen zwei Fälle unterscheiden werden.

1. Das Modell berechnet einen Ausgangswert. Der Ausgangsvektor ist ein Skalar. Es gilt $k = 1$ und $\boldsymbol{y} \in \mathbb{R}$.
2. Das Modell berechnet mehrere Ausgangswerte. Der Ausgangsvektor hat k Einträge, es gilt $\boldsymbol{y} \in \mathbb{R}^k$.

Für den **ersten Fall** eines eindimensionalen Modellausgangs wird die Länge l_c eines jeden Intervalls I^c in Gleichung 5.7 definiert:

$$l_c = \frac{y_{l,max} - y_{l,min}}{N} \tag{5.7}$$

Jedes Intervall I^c umfasst eine Menge an Zielgrößen $\boldsymbol{y}$ verknüpft mit dem zugehörigen BRS $\mathcal{D}_{\text{bw,sub}}$. Es gilt

$$I^c = \left\{ y_l \in \mathbb{R} \mid y_{l,min} + l_c \cdot (c-1) \leq y_l \leq y_{l,min} + l_c \cdot c \right\}. \tag{5.8}$$

Gleichung 5.8 beschreibt die Intervalle der einzelnen *bins*.
Der **zweite Fall** berücksichtigt ein Modell mit mehreren Ausgängen y_l. Hierfür müssen die *bins* für jede Zielgröße y_l des Modells separat erzeugt werden. Die Intervalle werden ebenfalls nach den Gleichungen 5.7 und 5.8 berechnet.

Eine isolierte Betrachtung der einzelnen Modellausgänge wird in dieser Arbeit bewusst verwendet, um eine falsche Kombination der Wertebereich zu vermeiden. Da ML-Modelle Ausgangswerte unterschiedlichster physikalischer Zusammenhänge und Einheiten in einem Modell berechnen können, ist für einen Eingangsvektor $\boldsymbol{x}$ eine breite Streuung der Ausgangswerte $\hat{y}_l$ im Ausgangsdatenraum $\mathcal{D}_{\text{aus}}$ möglich. Das Verfahren zur Ermittlung von BRSs würde bei einer kombinierten Betrachtung und Unterteilung des Ausgangsdatenraums $\mathcal{D}_{\text{aus}}$ Zielgrößen y_l in Unterdatenräume $\mathcal{D}_{\text{aus,sub}}$ zusammenfassen, deren zugehörige Eingangswertekombinationen $\boldsymbol{x}$ im Eingangsdatenraum $\mathcal{D}_{\text{ein}}$ weit auseinander liegen bzw. nicht durch einen Modelleingang zu erzeugen sind. Die Minima und Maxima der einzelnen Zielgrößen y_l spannen einen Hyperquader in R^k auf. Eine gekoppelte Unterteilung aller Wertebereiche in N *bins* erzeugt äquidistante Hyperquader. In diesen sind die Werte der Zielvektoren $\boldsymbol{y}$ ihrem skalaren (und normierten) Wert nach gruppiert. Dabei wird nicht berücksichtigt, dass die zugrundeliegenden Eingangswertekombinationen $\boldsymbol{x}$ für die verschiedenen Zielvektoren $\boldsymbol{y}$ signifikante Unterschiede aufweisen können. Eine gekoppelte Unterteilung würde zu OCSVMs führen, deren EG durch den gesamten, zulässigen Eingangsdatenraum $\mathcal{D}_{\text{ein}}$ reichen, um verschiedene Eingangswertekombination gemeinsam abdecken zu können. Dies führt zu vielen Überschneidungen der unterschiedlichen EG und damit zu zusätzlichen Unsicherheiten in der Plausibilisierung.
Daher wird wie für den eindimensionalen **ersten Fall** verfahren: der Wertebereich einer jeden Zielgröße y_l wird in unabhängige *bins* unterteilt. Allen Zielwerten eines *bins* $\mathcal{D}^c_{\text{aus,sub}}$ werden die zugehörigen Eingangsvektoren $\boldsymbol{x}$ zugeordnet. Diese ergeben ein BRS $\mathcal{D}^c_{\text{bw,sub}}$.

Die **Verwendung empirischer Quantile** stellt ein weiteres Verfahren zur Aufteilung in Unterdatenräume $\mathcal{D}_{\text{aus,sub}}$ dar. Dabei wird der Wertebereich des Ausgangsdatenraums in N Quantile unterteilt. Die Größe $c = [1, N]$ beschreibt die Anzahl der erzeugten Unterdatenräume bzw. Quantile. Jeder der N Unterdatenräume $\mathcal{D}^c_{\text{aus,sub}}$ ist durch ein Quantil beschrieben und wird näherungsweise mit derselben Anzahl an Datenpunkten (Zielvektoren) $\boldsymbol{y}$ aufgefüllt. Die Größe p^c gibt den Anteil an Datenpunkten in Prozent an, die in das c-Quantil fallen:

$$p^c = \frac{1}{N} c \cdot 100\,\%; \tag{5.9}$$

$$\mathcal{D}^c_{\text{aus,sub}} = Q_{p^c} - Q_{p^{c-1}} \quad \forall c \in [1, N]. \tag{5.10}$$

In Gleichung 5.10 wird die Berechnung der zugehörigen Datenpunkte eines Unterdatenraums $\mathcal{D}_{\text{aus,sub}}$ anhand der aus N resultierenden Quantile für den sortierten Wertebereich eines Modellausgangs y_l definiert. Dies hat zur Folge, dass jeder Unterdatenraum $\mathcal{D}^c_{\text{aus,sub}}$ annähernd dieselbe Anzahl an Datenpunkten enthält [35]. Im Unterschied dazu kann die Einteilung in äquidistante Intervalle (*bins*) eine ungleiche Anzahl an Datenpunkten pro Intervall ergeben. Dies kann die Verteilung der

Datenpunkte im Unterdatenraum maßgeblich beeinflussen. Die aus den c-Quantilen erzeugten Unterdatenräume $\mathcal{D}^{c}_{\text{aus,sub}}$ haben eine andere Zusammensetzung als bei der Verwendung von einer gleichen Anzahl an *bins*.
Die Handhabung von mehrdimensionalen Ausgangsdatenräumen für $k > 1$ wird für dieses Verfahren analog zur beschriebenen Methodik für äquidistante Intervallen durchgeführt.
Die Auswirkungen der beiden Methoden zur Erstellung der BRSs auf die RPA werden in Kapitel 5.4 bewertet. Sowohl der Einfluss des Unterteilungsverfahrens als auch die Anzahl c der Klassen werden hierfür untersucht.

5.2.2 Systematische Optimierung der Datenraumbegrenzung

Im Folgenden wird die Methodik zur Erstellung geeigneter Begrenzungen der einzelnen Unterdatenräume $\mathcal{D}_{\text{bw,sub}}$ beschrieben. Dafür werden OCSVMs mit den Datensätzen der einzelnen $\mathcal{D}_{\text{bw,sub}}$ trainiert.
Um Modellberechnungen mit der RPA plausibilisieren zu können, müssen die Modellausgänge $\hat{\boldsymbol{y}}$ einem der erzeugten Unterdatenräume zugeordnet werden (vgl. Kap. 5.2.1) können. Diese Klassifizierung wird mittels der Wertebereiche der Unterdatenräume $\mathcal{D}^{c}_{\text{aus,sub}}$ durchgeführt.
Daran schließt sich die Überprüfung des rückwärts verknüpften Eingangsvektors $\boldsymbol{x}$ an. Der Vektor wird als *innerhalb* oder *außerhalb* des zugehörigen BRS $\mathcal{D}_{\text{bw,sub}}$ klassifiziert. Dafür werden OCSVMs verwendet. Sie ermöglichen eine automatisierte und umfassende Ummantelung der Trainingsdaten mit einer Hyperebene (EG). Gleichzeitig sollen leere Bereiche des Datenraums möglichst genau ausgeschnitten werden. Um eine OCSVM optimal an die Kontur der Datenpunkte im $\mathbb{R}^i$ anzupassen, ist eine Parameteridentifikation der Hyperparameter γ und ν nötig. Diese Parameteridentifikation wird in Form einer Gitterstudie durchgeführt.
Um eine Optimierung durchzuführen, bedarf es einer Zielgröße. Aus der Literatur sind verschiedene Methoden bekannt. Für diese Arbeit wird die von Xiao, Wang, Zhang et al. vorgestellte Methode zur Optimierung der Straffheit (engl. *tightness*) der EG einer OCSVM mit einem Gauß-Kernel verwendet [66]. Hierbei wird der Hyperparameter γ optimiert, welcher das Verhalten des Kernels der OCSVM steuert. Nach [66] ist die Variation direkt monoton mit der Straffheit der EG verknüpft. Je kleiner γ, desto straffer und je größer, desto weiter ist die EG einer damit trainierten OCSVM. Zur Beurteilung der *tightness* werden eine untere Grenze und eine obere Grenze des Hyperparameters γ festgelegt. Nach jeder Bestimmung eines Wertes für γ wird anhand von Testdatenpunkten ermittelt, wie gut die erstellte EG diese beschreibt. Folgendes Vorgehen wird dabei durchgeführt:

1. zu weit: $\gamma \downarrow$
2. zu straff: $\gamma \uparrow$
3. zu straff und zu weit: $\gamma \uparrow$

4. weder zu weit noch zu straff: Optimierung beendet

In Abbildung 6.4 sind zwei unterschiedliche EG zur Beschreibung des selben Datensatzes in $\mathbb{R}^3$ dargestellt. Es zeigt sich, dass für $\gamma = 175$ eine deutlich straffere EG erzeugt wird, als bei der Verwendung von $\gamma = 10$. Ein niedriger Wert des Hyperparameters γ erzeugt eine EG, welche die vorliegende Punktewolke des Datensatzes zwar einschließt aber keine detaillierte Beschreibung der Form ermöglicht. Für $\gamma = 175$ zeigt sich hingegen eine EG, welche der Form der Punktewolke folgt. Zudem werden Leerstellen und nicht-konvexe Formen durch die EG beschrieben. Ein weiter Parameter der Gitterstudie ist ν, welcher den Anteil an *außerhalb* der EG liegenden Datenpunkten steuert. Dieser wird in einem Intervall von $\nu = [0.001, 0.01]$ in der Gitterstudie untersucht. Der Parameter ν bestimmt die Anzahl an ignorierten Messpunkten $\boldsymbol{x}_{\text{ign}}$ *außerhalb* der EG. Gleichzeitig gibt ν den Anteil an zu verwendenden Stützvektoren an [97]. In Abbildung 3.2 ist der Einfluss des Hyperparameters ν dargestellt. Für $\nu = 0.01$ verbleiben deutlich sichtbar mehr Datenpunkte $\boldsymbol{x}_{\text{ign}}$ *außerhalb* der EG als für $\nu = 0.001$. Durch Verwendung der Methodik zur Optimierung der *tightness* der EG kann eine gute und schnelle Optimierung der Hyperebene (EG) durchgeführt werden.
Die Rechenkomplexität der Optimierung von γ kann mit $O(n^2)$ angegeben werden [66]. Das Training einer OCSVM hat die Rechenkomplexität $O(n^3)$ [64]. Damit kann für die Gitterstudie mit $n_{EG,G}$ Hyperparametersätzen und q Iterationen zur Optimierung von γ von einer Rechenkomplexität von $O(n_{EG,G} q n^3)$ ausgegangen werden. Da das Training nicht auf dem Fahrzeugsteuergerät, sondern im Cloud-Backend stattfindet, sind keine Einschränkungen in Bezug auf die Rechenkomplexität gegeben. Die Ausführung der trainierten OCSVM zur Klassifizierung von Datenpunkten als *innerhalb* oder *außerhalb* der erstellten Hyperebene (EG) kann mit $O(n)$ angenommen werden [64]

5.2.3 Kritische Eigenschaften der Unterdatenräume

Für das RPA werden BRSs $\mathcal{D}_{\text{bw,sub}}$ mit zugehörigen EG erstellt. Dazu kommen die in Kapitel 5.2.1 beschriebenen Verfahren zum Einsatz. Können die Modellberechnungen im Ausgangsdatenraum einem der Unterdatenräume $\mathcal{D}^c_{\text{aus,sub}}$ (äquidistante Intervalle oder Quantile) zugeordnet werden, so kann mittels der EG überprüft werden, ob der zugehörige Eingangsvektor im rückverknüpften BRS $\mathcal{D}^c_{\text{bw,sub}}$ liegt. Ist dies gegeben, kann grundlegend von einer plausiblen Modellberechnung ausgegangen werden. Um weitere Verfeinerungen der Plausibilisierung durchführen zu können, werden für jedes $\mathcal{D}^c_{\text{bw,sub}}$ kritische Eigenschaften ermittelt. Diese umfassen statistische Größen und Kennzahlen zur Beschreibung der inkludierten Datenmenge. Folgende Werte werden für jedes $\mathcal{D}^c_{\text{bw,sub}}$ errechnet:

- Anzahl der Datenpunkte $\boldsymbol{x}$ innerhalb der EG eines jeden BRS $\mathcal{D}^c_{\text{bw,sub}}$,
- Anteil p_e der fälschlicherweise im Training der EG ignorierten Datenpunkte $\boldsymbol{x}_{\text{ign}}$ *außerhalb* der trainierten EG eines jeden $\mathcal{D}^c_{\text{bw,sub}}$,

- Anteil p_t der zu $\mathcal{D}^c_{\text{bw,sub}}$ gehörenden Datenpunkte $\boldsymbol{x}$ gegenüber allen von der EG eingefassten Datenpunkte im Eingangsdatenraum $\mathcal{D}_{\text{ein}}$,
- Geometrischer Schwerpunkt C^c_{m}, errechnet für alle Datenpunkte $\boldsymbol{x}$, zugehörig zu $\mathcal{D}^c_{\text{bw,sub}}$ innerhalb der EG (vgl. Gl. 5.11),
- Anteil der Datenpunkte aus anderen BRSs, welche von der EG eingeschlossen sind.

Diese kritischen Eigenschaften dienen der Klassifizierung von Datenpunkten $\boldsymbol{x}$, welche innerhalb mehrerer EG liegen. Solche Fälle treten vor allem in Gebieten hoher Dichte auf. Hier liegen viele Datenpunkte eng bei einander und die EG mehrerer BRSs überschneiden sich. Um während der Inferenz des ML-Modells und der Anwendung der RPA neue Eingangsvektoren $\boldsymbol{x}_{\text{HRC}}$ dem passenden BRS $\mathcal{D}^c_{\text{bw,sub}}$ zuzuordnen, wird der geometrische Schwerpunkt nach Gleichung 5.11 verwendet. Der euklidische Abstand zu den in Frage kommenden, geometrischen Schwerpunkten wird als Kriterium verwendet, um den Datenpunkt $\boldsymbol{x}_{\text{HRC}}$ einem BRS zuzuordnen.
Durch die Verwendung des geometrischen Schwerpunkts wird die Verteilung der Datenpunkte eines jeden BRS $\mathcal{D}^c_{\text{bw,sub}}$ berücksichtigt. Dieses Vorgehen beeinflusst die Klassifizierung neuer Eingangsvektoren $\boldsymbol{x}_{\text{HRC}}$, indem eine Zuordnung zu Randgebieten eines BRS vermieden wird.
Ein Eingangsvektor $\boldsymbol{x}_{\text{HRC}}$ kann aufgrund seiner Lage im Raum innerhalb verschiedener, sich überschneidender EG liegen. Um diesen Datenpunkt einem BRS zuzuordnen, wird der euklidische Abstand zu den geometrischen Schwerpunkten aller relevanten BRSs verwendet. Der Eingangsvektor $\boldsymbol{x}_{\text{HRC}}$ wird dem BRS mit dem kürzesten euklidischen Abstand zugeordnet. Dafür werden im *preprocessing* die geometrischen Schwerpunkte nach Gleichung 5.11 für jedes BRS berechnet. In Gleichung 5.11 werden alle $n_{bw,c}$ Datenpunkte des jeweiligen $\mathcal{D}^c_{\text{bw,sub}}$ verwendet.

$$C^c_{\text{m}} = \frac{1}{N} \sum_{s=1}^{n_{bw,c}} \boldsymbol{x}_s \quad \forall \boldsymbol{x} \subseteq \mathcal{D}^c_{\text{bw,sub}} \tag{5.11}$$

Während der Inferenz wird für jeden Eingangsvektor $\boldsymbol{x}_{\text{HRC}}$ die euklidische Distanz d^c_{min} zum nächstgelegenen geometrischen Schwerpunkt ermittelt werden. Der euklidische Abstand d^c_{min} bezieht sich auf $\boldsymbol{x}_{\text{HRC}}$ zu C^c_{m} in $\mathbb{R}^j$. Die Größe d^c_{min} dient als Ähnlichkeitsmaß für die Zuordnung der Datenpunkte zu einem BRS.

5.3 Lipschitz-Stetigkeit und Modellrobustheit

Die entwickelte RPA zur Plausibilisierung der ML-Modellberechnungen beruht auf dem Nachweis der Lipschitz-Stetigkeit der Modelle sowie der Definition valider Ausgangsdatenräume $\mathcal{D}_{\text{aus,sub}}$ und zugehöriger BRSs $\mathcal{D}_{\text{bw,sub}}$ (vgl. Kap. 5.2).
Um den Einsatz der entwickelten KNN im PKW-TM als Teil der Antriebsstrangsoftware im Fahrzeug einzusetzen, muss ein robustes Modellverhalten sichergestellt werden.

Neben den gängigen Sicherheitsüberprüfungen der zulässigen Wertebereiche und der Laufzeit soll das Modell zuverlässig die richtigen Werte berechnen.
Das Ziel ist es ein robustest Modellverhalten zu gewährleisten. Diese kann in anderen Bereichen des *Machine Learning* zum Beispiel durch leicht veränderte Testdaten überprüft werden. Dabei wird die Modellrobustheit von ML-Modellen zur Klassifizierung von Bildern beispielsweise anhand modifizierter Bilder überprüft. Diese Bilder sind für das menschliche Auge nicht erkennbar verändert, führen aber bei vielen ML-Modellen zu Fehlklassifikation. Diese Schwäche kann für sogenannte feindliche Angriffe (engl. *adversarial attacks*) auf sicherheitskritische Softwarekomponenten genutzt werden [98]. Ein Modell wird in diesem Zusammenhang als robust bezeichnet, wenn es Bilder mit einer nicht erkennbaren Störungen in dieselbe Klasse einordnet wie das unveränderte Original.
Dieser Ansatz lässt sich nicht ohne weiteres auf die Verwendung von Regressionsmodellen anwenden. In der Literatur werden andere Verfahren vorgeschlagen. Eines davon ist die Verwendung der Lipschitz-Konstante L als Maß der Modellrobustheit. Die Lipschitz-Konstante stellt die Anfälligkeit des Modellausgangs auf Störungen des Modelleingangs dar [98].
Die Modellrobustheit basiert auf der Lipschitz-Stetigkeit des Modells. Diese ist für FNN bewiesen. Betrachtet man das FNN als Funktion $f : \mathbb{R}^j \to \mathbb{R}^k$, so ist eine globale Lipschitz-Stetigkeit für f auf $\mathcal{D}_{\text{ges}} \subseteq \mathbb{R}^j$ gegeben, sofern eine Lipschitz-Konstante $L \geq 0$ in der Form existiert, dass gilt

$$\| f(\boldsymbol{x}_1) - f(\boldsymbol{x}_2) \| \leq L \, \| \boldsymbol{x}_1 - \boldsymbol{x}_2 \| \quad \forall \boldsymbol{x} \in \mathcal{D}_{\text{ges}}. \tag{5.12}$$

Die Konstante L begrenzt den maximalen Gradienten der Funktion f. Auf diese Weise beeinflusst L, wie Veränderungen eines Funktionswerts $f(\boldsymbol{x}) = \hat{\boldsymbol{y}}$ im Ausgangsdatenraum $\mathcal{D}_{\text{aus}}$ von Veränderungen des Eingangswertes $\boldsymbol{x}$ im Eingangsdatenraum $\mathcal{D}_{\text{ein}}$ abhängen. Die Sensitivität der Funktion f auf Veränderungen der Eingangswerte $\boldsymbol{x}$ wird durch die Lipschitz-Konstante L beschrieben [35], [98], [99]. Allerdings ist die Berechnung der exakten Lipschitz-Konstante L ein NP-schweres Problem [100]. Es existieren verschiedene Wege, um die Konstante L näherungsweise zu bestimmen.

5.3.1 Lipschitz-Stetigkeit von FNN

Die Lipschitz-Konstante kann für KNN und die meisten gängigen Schichten ermittelt werden [99], [100]. Aus den Ausführungen von Ruan, Huang und Kwiatkowska, Fazlyab, Robey, Hassani et al. und Szegedy, Zaremba, Sutskever et al. geht hervor, dass FNN mit ReLUs Lipschitz-stetig im Sinne von Gleichung 5.12 sind [77], [101].
Diese Eigenschaft der FNN wird verwendet, um die Verknüpfung der BRS $\mathcal{D}^c_{\text{bw,sub}}$ mit den zugehörigen Unterdatenräumen $\mathcal{D}^c_{\text{aus,sub}}$ im Ausgangsdatenraum $\mathcal{D}_{\text{aus}}$ zu validieren. Damit kann die RPA der Modellberechnungen (der *Fallstudie I*) mittels der eingeführten Methodik durchgeführt werden.

Für die in *Fallstudie I* verwendeten FNN und ReLUs gilt die Lipschitz-Stetigkeit [77], [101]. Es kann zudem eine exakte Beschränkung der Lipschitz-Konstante L im Training der KNN vorgenommen werden, um diese noch weiter einzuschränken. Dies führt in der Literatur meist zur besseren Generalisierung der Modelle auf neuen Daten. Diese Methodik führt im Training zu einem erheblichen Mehraufwand und erhöhter Rechenkomplexität [77], [99], [101]. Für die Validierung der RPA wurde auf Erweiterungen des Modelltrainings verzichtet.

5.3.2 Lipschitz-Stetigkeit von RNN und weiteren ML-Modellen

Die nachweisliche Lipschitz-Stetigkeit für FNN mit ReLUs hat gezeigt, dass die entwickelte RPA auf diese Modelle angewendet werden kann. Ein Nachweis der Lipschitz-Stetigkeit von RNN kann aus der Literatur nur im Zusammenhang mit einer beschränkten Lipschitz-Konstante im Training entnommen werden [78]. Wie vorstehend für FNN beschrieben, trägt dieses Verfahren zur Robustheit der RNN bei. Es verlangt im Training allerdings einen hohen Zusatzaufwand aufgrund der zusätzlichen, vorwärts gekoppelten Anwendung der Kettenregel zu Bestimmung der Lipschitz-Konstante während der Optimierung der Modellparameter [100]. Auf einer skalierbaren Cloudinstanz kann dieser Zusatzaufwand problemlos in Kauf genommen werden. Für die vorliegende Arbeit liegt der Fokus auf der Entwicklung möglichst kleiner und gleichzeitig genauer KNN. Daher wird für die Bewertung der entwickelten RPA die *Fallstudie I* betrachtet. Die darin verwendeten FNN sind Lipschitz-stetig und können auf plausible Modellausgänge hin untersucht werden.

5.4 Methodik zur Plausibilisierung der Modellausgänge

Um die berechneten Modellausgänge $\hat{y}$ eines ML-Modells auf ihre Plausibilität hin zu überprüfen, wurden in diesem Kapitel verschiedene Methoden und Vorverarbeitungsschritte eingeführt. Diese werden zur RPA kombiniert. Die RPA kann als Arbeitsablauf in die Inferenz der Modelle eingebunden werden. Nach jeder Ausführung des KNN können die Ausgabewerte $\hat{y}$ mittels RPA überprüft werden. Implausible Werte werden mit einem Ersatzwert substituiert (vgl. Kap. 5.4.3).
Die dafür entwickelten Verfahren lassen sich in vier Schritte unterteilen:

1. Datenvorverarbeitung und *feature engineering* zur Erstellung der Trainingsdaten,
2. Definition der Ein- und Ausgangsdatenräume inklusive Aufteilung in Unterdatenräume und BRSs,
3. Training der EG und Berechnung der kritischen Eigenschaften,
4. Plausibilitätsbewertung der Modellberechnungen.

Die Schritte 1. - 3. werden im *preprocessing* (vgl. Kap. 3) durchgeführt und stellen die nötigen EGn sowie die kritischen Eigenschaften der BRSs für den 4. Schritt und damit

für die RPA zur Verfügung. Dieser wird im Folgenden genauer beschrieben.

5.4.1 Plausibilitätsanalyse der Modellberechnung

Sobald das *perprocessing* abgeschlossen und die KNN trainiert sind, können die Modellberechnungen mittels RPA überprüft werden. Nach jedem Inferenzschritt wird überprüft, ob der verwendete Eingangsvektor $\boldsymbol{x}$ innerhalb des zulässigen Eingangsdatenraums $\mathcal{D}_{\text{ein}}$ liegt. Analog dazu wird überprüft, ob sich der berechnete Modellausgang $\hat{\boldsymbol{y}}$ innerhalb des zulässigen Ausgangsdatenraums $\mathcal{D}_{\text{aus}}$ befindet. Die zulässigen Ein- und Ausgangsdatenräume basieren auf den verfügbaren Trainingsdaten (vgl. Kap. 3). Die Zugehörigkeit zu einem (Unter-)Datenraum bzw. BRS wird anhand der dafür trainierten OCSVMs überprüft. Dazu wird sie auf den Datenpunkt angewendet und klassifiziert diesen als *innerhalb* oder *außerhalb* der EG. Wird der Eingangs- oder Ausgangsvektor des Modells in diesem Schritt schon als *außerhalb* des zulässigen Datenraums definiert, wird unmittelbar die Ersatzwertbildung durchgeführt (vgl. Kap. 5.4.3). Die Modellberechnung wird als implausibel markiert. Liegt die Modellberechnung $\hat{\boldsymbol{y}}$ innerhalb des Ausgangsdatenraum $\mathcal{D}_{\text{aus}}$, kann jede Komponente $\hat{y}_l$ des Ausgangsvektors einem der vorab definierten Unterdatenräume $\mathcal{D}^c_{\text{aus,sub}}$ zugeordnet werden. Wenn gilt,

$$\hat{\boldsymbol{y}} \in \mathcal{D}^c_{\text{aus,sub}}, \tag{5.13}$$

muss der zugehörige Eingangsvektor $\boldsymbol{x}$ innerhalb des rückwärts verknüpften BRS $\mathcal{D}^c_{\text{bw,sub}}$ liegen. In diesem Fall muss zusätzlich gelten:

$$\boldsymbol{x} \in \mathcal{D}^c_{\text{bw,sub}}. \tag{5.14}$$

Falls der Fall eintritt, dass $\boldsymbol{x}$ mehreren $\mathcal{D}^c_{\text{bw,sub}}$ zugeordnet werden kann, werden die kritischen Eigenschaften der einzelnen BRSs zur automatischen Zuordnung verwendet. Zuerst wird der euklidische Abstand d^c zu jedem geometrischen Schwerpunkt der möglichen BRSs nach berechnet.
Der geringste Abstand d^c_{min} wird ermittelt und der Eingangsvektor $\boldsymbol{x}$ wird daraufhin dem nächst gelegenen BRS zugeordnet. Dafür wird der d^c auf zwei Stellen hinter dem Komma gerundet. Sollten mehrere BRSs mit identischen Abstände vorliegen, wird der Anteil p_e an falsch klassifizierten Datenpunkten in $\mathcal{D}^c_{\text{bw,sub}}$ berücksichtigt. Der Eingangsvektor $\boldsymbol{x}$ wird dem BRS mit dem geringsten Wert p_e zugeordnet.
Ist $\boldsymbol{x}$ dem mit $\mathcal{D}^c_{\text{aus,sub}}$ verknüpften $\mathcal{D}^c_{\text{bw,sub}}$ zugeordnet, kann die Berechnung $\hat{\boldsymbol{y}}$ als plausibel (in Bezug auf die gegebene Trainingsdatenbasis) bewertet werden.
Die Gründe für implausible Modellberechnungen sind vielfältig. Eine Über- oder Unteranpassung des Modells kann dafür verantwortlich sein. Ebenso lässt sich auf Grund des *black-box* Charakters von KNN nicht ausschließen, dass eine im Training nicht verwendet Kombination aus Ein - und Ausgängen während der Inferenz zu implausiblen Ergebnissen führt [24], [25], [100].

5.4.2 Einfluss der Unterteilungsmethodik auf die RPA

Die Anzahl der Unterdatenräume beeinflusst die Genauigkeit sowie die Wahrscheinlichkeit sich überschneidender Hyperebenen. Somit muss eine kritische Betrachtung und anschließende Abwägung durchgeführt werden, wie eine Unterteilung sinnvoll aussehen kann.
Der Zielwert der Optimierung der Anzahl N von Unterdatenräumen ist durch die Anzahl an möglichen Unterdatenräumen pro Punkt gekennzeichnet. Iterativ wird die Anzahl N variiert (gesteuert über eine Gitterstudie). Für jede Iteration kann die durchschnittliche Mehrfachabdeckung pro Punkt $\overline{o}_{\mathrm{EG}}$ berechnet werden. Diese sagt aus, innerhalb wie vieler EG ein einziger Datenpunkt im Durchschnitt liegt. Die Ergebnisse der als implausibel klassifizierten Testdaten bieten in Kombination einen weiteren Zielwert für die Auswertung der Gitterstudie. Dieser wird höher gewichtet. Der Einfluss der Unterteilungsmethodik wurde einer begleitenden Untersuchung zu dieser Arbeit in [104] bewertet. Es hat sich gezeigt, dass die Unterteilung in äquidistante Intervalle bessere Ergebnisse liefert als die Einteilung in Quantile. Dies zeigt sich vor allem in den Extrembereichen, wie Minima oder Maxima. Diese verfügen meist über eine deutlich kleinere Anzahl an Trainingspunkten als gemeinhin oft auftretende Betriebspunkte. Daher führt eine Unterteilung in Quantile sehr wahrscheinlich zur Zusammenfassung weit auseinander liegender Datenpunkte in einem Unterdatenraum $\mathcal{D}^{c}_{\mathrm{aus,sub}}$. Somit beinhaltet das rückverknüpfte BRS Datenpunkte großer Varianz.

5.4.3 Ersatzwertbildung

Da die ML-Modelle in einem zyklischen Rechentakt auf dem Steuergerät laufen, müssen diese zu jedem vollendeten Rechentakt einen validen Ausgang bereit stellen. Weitere Teile der Software sind auf die Bereitstellung der Werte in Echtzeit angewiesen [15]. Derzeit werden das Signal qualifizierende Markierungen (engl. *flags*) in Form eines codierten Bits im *header* mitgesendet (engl. *signal qualifier*), welche Aufschluss über die Gültigkeit eines Signals zulassen.
Als implausibel definierte Modellausgänge $\hat{\boldsymbol{y}}_{\mathrm{imp}}$ müssen für die Weiterverarbeitung nutzbar gemacht werden. Dies kann durch eine Ersatzwertbildung geschehen. Alternativ kann mittels eines *Signal Qualifiers* die Information übermittelt werden, dass gerade kein valider Wert vorliegt. Da die entwickelten ML-Modelle im Umfeld des prädiktiven TM eingesetzt werden sollen, welches auf grob aufgelösten Horizontdaten $\boldsymbol{x}_{\mathrm{HRC}}$ aus dem HRC basiert, können die verknüpften Softwarekomponenten mit beiden Fällen umgehen. Wird für nachstehende Berechnungen ein Eingabewert benötigt, soll die Ersatzwertbildung stets den Vorzug erhalten. Damit sollen Rechenzyklen ohne Datenupdate minimiert werden. Zur Bildung des Ersatzwerts $\hat{\boldsymbol{y}}_{\mathrm{ers}}$ werden die zwei nachfolgend vorgestellten Verfahren bewertet:

1. Verwendung des letzten, als plausibel bewerteten Ausgangs,
2. Mittelwert mehrerer, vorangegangener, plausibler Ausgangswerte $\hat{\boldsymbol{y}}_{\mathrm{plaus}}$.

Die Wahl des Verfahrens muss dem Anwendungsfall angepasst werden. In der vorliegenden Arbeit hat sich das zweite Verfahren durchgesetzt, da es den Trend der vergangenen Modellberechnungen mit einbezieht. Das Verfahren zur Ersatzwertbildung unterscheidet zwischen zwei Fällen. Untersucht wird, ob sich der letzte plausible Werte innerhalb des 5%-Quantils des Wertebereichs befindet. Ist dies der Fall, wird ein Ersatzwert mit dem Minimalwert des Wertebereichs von $\hat{y}$ angenommen. Dieses Vorgehen basiert auf empirischen Erkenntnissen, welche in der Praxis auf eine unzureichende Abbildung des Modells von Standphasen hindeuten. Befindet sich der letzte plausible Wert oberhalb des 5%-Quantils des Wertebereichs, wird ein Mittelwert als Ersatzwert verwendet. Dieser errechnet sich aus den letzten $n_{\text{ers}} = 5$ plausiblen Modellberechnungen $\hat{y}_{\text{plaus}}$ zum aktuellen Zeitschritt t_{ers}. Liegen für den Zeitschritt der Ersatzwertbildung noch weniger als n_{ers} plausible Modellberechnungen $\hat{y}_{\text{plaus}}$ vor, wird der letzte plausible Wert verwendet. Dies ist in Gleichung 5.15 beschrieben:

$$\hat{y}_{\text{ers}}(t_{\text{ers}}) = \begin{cases} \frac{1}{n_{\text{ers}}} \sum_{k=1}^{n_{\text{ers}}} \hat{y}_{\text{plaus}}(t_{\text{ers}} - k), & \text{wenn } t_{\text{ers}} \geq n_{\text{ers}}, \\ \hat{y}_{\text{plaus}}(t_{\text{ers}} - 1), & \text{wenn } t_{\text{ers}} < n_{\text{ers}}, \\ \text{implausibel}, & \text{ansonsten.} \end{cases} \tag{5.15}$$

Liegt noch kein plausibler Wert vor muss der *Signal Qualifier* diese Information übermitteln und die angeschlossenen Softwarekomponenten verarbeiten diese Information. Dieses Vorgehen stellt sicher, dass keine Ausreißer oder Trendwenden durch die Substitution implausibler Werte in der Zeitreihe entstehen.

Die grobe Eingangsdatenbasis aus dem HRC liefert für eine vertiefte Ersatzwertbildung keinen ergiebigen Ausgangspunkt. Die Anwendung der RPA wird in Kapitel 6 diskutiert.

6 Bewertung der Ergebnisse und Validierung im Gesamtsystem

In dieser Arbeit wurde eine Methodik entwickelt, die ein automatisiertes Training von ML-Modellen für die Anwendung im PKW-TM ermöglicht. Mit der neu entwickelten RPA können die Modellberechnungen mit geringem Ressourceneinsatz plausibilisiert werden.
Die Trainingsdaten werden als Fahrzeugmessdaten kontinuierlich vom Fahrzeug in das Cloud-Backend übermittelt. Dazu wird die serienmäßig verfügbare Kommunikationsschnittstelle (TCU) des Fahrzeugs verwendet. Zur Vorverarbeitung der Datengrundlage und zum Training von ML-Modellen wurde eine ML-Pipeline entwickelt und auf dem Cloud-Backend implementiert. In Kombination mit der RPA liegt eine vollständige Methodik zur automatisierten Modellierung und Plausibilisierung von ML-Modellen vor. Ein Großteil der entwickelten Methodik wird derzeit bei der *Mercedes-Benz Group AG* in der Entwicklung von Regelungsmodellen für das PKW-TM angewendet. Dabei kommt das in diese Arbeit verwendete Cloud-Backend zum Einsatz.

Die Bewertung beantwortet die anfangs formulierten Forschungsfragen (vgl. Kap. 1.2). Dafür werden in Kapitel 6.2 die *Fallstudien I & II* ausgewertet. Anhand der Gütekriterien (vgl. Kap. 4.2.1) werden das jeweils beste FNN und RNN automatisiert ausgewählt und als Ergebnis der ML-Pipeline bereitgestellt. Nachdem die Bewertung auf dem Testdatensatz $\mathbb{S}_{\text{test}}$ stattgefunden hat, folgt die dem Anwendungsfall im Gesamtsystem entsprechende Inferenz der Modelle mit ADAS-Horizontdaten. Darüber hinaus wird die RPA auf die Modellausgänge angewendet und der Einfluss auf die Modellgüte untersucht. Damit kann die Eignung der Modelle in prototypischer Anwendung bewertet werden. Kapitel 6.3 schließt die Auswertung mit einer Diskussion und Einordnung der gewonnen Erkenntnisse.

6.1 Anwendung und Bewertung der entwickelten Methodik

Zur Bewertung in der Anwendung werden die in dieser Arbeit entwickelten Methoden kombiniert und liefern so einen vollautomatisierten Prozess in Form einer ML-Pipeline. Diese beinhaltet den Datenaustausch zwischen Fahrzeug und Cloud-Backend über die TCU. Das Modul ermöglicht den Datentransfer über das Mobilfunknetz. Für die prototypische Evaluation der entwickelten Methodik kommt das in der Serienentwicklung eingesetzte Cloud-Backend der *Mercedes-Benz Group AG* zur Anwendung.

Die Messdaten werden vom Versuchsfahrzeug (vgl. Kap. 2.4) über die TCU an das Cloud-Backend gesendet. Die Messdaten werden in einem *Datalake* gesammelt und von der ML-Pipeline zu Trainingsdaten weiterverarbeitet. Das ressourcenintensive *preprocessing* und Training werden als Bestandteil der ML-Pipeline auf dem skalierbaren Cloud-Backend ausgeführt.
Die Methodik konnte bereits in den bestehenden Entwicklungsablauf integriert werden, um ein automatisiertes Modelltraining mit einer kontinuierlich wachsenden Datenbasis durchzuführen. Nach dem erfolgreichen Training können die Modellparameter über die TCU auf das Fahrzeugsteuergerät übertragen werden. Dadurch kann der manuelle Kalibrierungsaufwand der Modellparameter deutlich reduziert werden. Zudem ist zu Beginn der Modellierung keine Definition der physikalischen Modellarchitektur mehr nötig. Sowohl die Definition der Modellarchitektur als auch die Kalibrierung der Modellparameter wird durch die Verwendung der entwickelten ML-Pipeline durchgeführt. Dafür wird in der ML-Pipeline eine Gitterstudie ausgeführt (vgl. Kap. 4.4.3). Dadurch ist es erstmals möglich, während des Entwicklungszyklus des Fahrzeugs vorhandene Modellparameter durch einen automatisiert im Cloud-Backend optimierten Parametersatz zu erneuern.
Das Verfahren stellt sicher, dass für die aktuell verfügbaren Trainingsdaten das bestgeeignete Modell als Ergebnis der ML-Pipeline ausgegeben wird. Dazu werden die Modelle anhand skalarer Bewertungskriterien (vgl. Kap. 4.2.1) evaluiert. Eine objektive Bewertung und Verbesserung der vorhandenen Modelle ist damit zu jeder Zeit sichergestellt.
Um die Modellberechnungen nach der Inferenz zu plausibilisieren, wird die in Kapitel 5 entwickelte RPA verwendet.

6.1.1 Machine-Learning-Pipeline

Die ML-Pipeline bildet das Rückgrat der gesamten Methodik. Sie wurde zu Beginn der Arbeit entwickelt. In diesem Zug wurde der detaillierte Aufbau der Kommunikation zwischen Fahrzeug und Cloud-Backend zur Aktualisierung der verwendeten Regelungsmodelle zum Patent angemeldet und offengelegt [102]. Die Ausführung der ML-Pipeline findet automatisiert auf der *Microsoft-Azure*-Plattform statt. Dieses Cloud-Backend verfügt über eine *Compute*-Instanz, welche ein Steuerskript ausführt. Über dieses Steuerskript können die verschiedenen Schritte der Pipeline konfiguriert und aufgerufen werden. Für jeden Schritt kann die optimale Hardware Ressource definiert werden. Für die Datenvorverarbeitung der Messungen aus dem Fahrzeug wird ein leistungsstarkes Cluster aus CPUs allokiert (CPU-Cluster). Das Modelltraining wird auf einem dafür optimierten Cluster aus GPUs parallelisiert (GPU-Cluster). Dies ermöglicht eine hohe Flexibilität, beste Skalierbarkeit und verteiltes Training, um die optimalen Ressourcen für das ML-Projekt bereit zu stellen. Mit Hilfe dieser Konfiguration wurden die groß angelegten Gitterstudien für die beiden *Fallstudien I & II* durchgeführt. Diese

ist auf einem lokalen Rechner nicht ohne weiteres möglich und stellt einen weiteren Vorteil der entwickelten Methodik dar.
Im direkten Vergleich zur aktuell in der Entwicklung verwendeten, lokal ausgeführten Prozesskette konnte ein deutlicher Leistungsschub und eine Zeitreduktion für die gesamte Prozesskette realisiert werden. Ein Modelltraining dauert je nach Datengrundlage und verwendeter Ressourcen wenige Minuten und findet automatisiert statt. Die manuelle Modellentwicklung kann dahingegen mehrere Arbeitsstunden der Entwickler_innen in Anspruch nehmen. Die ML-Pipeline beinhaltet zudem weitere Schritte wie die Auswahl des besten Modells und das Aktualisieren der Modelle auf dem Fahrzeug. Dadurch wird eine vollständig automatisierte Prozesskette für die Entwicklung von datenbasierten Modellen realisiert.
Die erzeugten ML-Modelle können aus dem proprietären *TensorFlow*-Format **.savedModel* oder **.tflite* mittels *Matlab Simulink* in C-Code überführt werden. Dabei entstehen *Source* und *Header-Files* in der Größe von weniger als 60 kB. Je nach Netzarchitektur und Anzahl n_{h} der verdeckten Schichten schwanken diese Werte wie in Tabelle 6.1 ersichtlich. Die Größe bewegt sich in einem Bereich, der mit den aktuell verwendeten PE-Modellen vergleichbar ist. Die im Gesamten durch das ML-Modell substituierten PE-Modelle belegen ca. 200 kB auf dem Speicher des Steuergeräts. Ein Vergleich der Modellparameter und Modellgröße ist in Kapitel 6.2 dargestellt.
Da die gesamte Modellierung auf dem Cloud-Backend durchgeführt wird, ist die

Tabelle 6.1: Ergebnisse der Gitterstudien für *Fallstudie I & II* (FNN & RNN). Anwendung der Modelle auf den Testdatensatz $\mathbb{S}_{\mathrm{test}}$ (vgl. Kap. 3.1.2 & 3.3.1).

Parameter		**FNN**	**RNN**
Eingangsgrößen*	j	7	7
Zielgrößen*	k	3	3
Verdeckte Schichten	n_{h}	16	8
Neuronen pro Schicht	n_{n}	128	32
Batch Size	n_{bs}	512	128
Lernrate	LR	0.0001	0.0001
Dropout Wahrscheinlichkeit	p_{drp}	75 %	25 %
Historie in Zeitschritten	t_{RNN}	-	10
Parameteranzahl	n_{param}	216.067	2.563
Abweichung Energieeintrag	Q_{in}	8 %	−5 %
Gütekriterium	$RMSE$	2767 W	2474 W

* vgl. Tab. 6.2

Methodik gut geeignet, um auf veränderliche Hard- oder Softwarestände des Entwicklungsfahrzeugs zu reagieren. Neu aufgezeichnete Trainingsdaten können verwendet

werden, um neue Modelle anderer Architektur und Größe zu erzeugen. Gleichzeitig können neue *Constraints* einfließen, um die Modellbildung oder Bewertung zu beeinflussen. Ebenfalls kann die nötige Rechenleistung an die Aufgabe und Rechenkomplexität angepasst werden.
Die entwickelte ML-Pipeline ist dadurch sehr gut geeignet, um eine automatisierte, skalierbare und objektive Modellierung für das prädiktive TM durchzuführen. Die ML-Pipeline kann von Entwickler_innen als Werkzeug im Entwicklungszyklus eines Fahrzeugs verwendet werden. Die damit erzeugten ML-Modelle adaptieren zu jedem Zeitpunkt die Hardware- und Softwarestände der Entwicklungsfahrzeuge, basierend auf den an das Cloud-Backend übertragenen Messdaten. Das entwickelte ML-Modell kann bei Bedarf fixiert werden, um in einen Softwarestand integriert zu werden. Damit steht einem Einsatz der Methodik in der Entwicklung nichts im Wege.

6.1.2 Bewertung der rückverknüpften Plausibilitätsanalyse

Die in Kapitel 5 entwickelte RPA ermöglicht eine Validierung der Modellausgänge, basierend auf der dem Training zugrunde liegenden Datenbasis $\mathcal{D}_{\text{ges}}$. Durch die Verwendung von OCSVMs können nicht-konvexe mehrdimensionale Datenräume mit einer ausreichend straffen EG eingehüllt und dadurch beschrieben werden. Dies ist essentiell für die Zuordnung der verwendeten Eingangswerte $\boldsymbol{x}$ zu einem der BRSs $\mathcal{D}_{\text{bw,sub}}$ des Eingangsdatenraums $\mathcal{D}_{\text{ein}}$. Die Zuordnung der Modellberechnungen $\hat{\boldsymbol{y}}$ zu validen Ausgangsdatenräumen $\mathcal{D}_{\text{aus,sub}}$ und die anschließende Rückverknüpfung zu einem plausiblen BRS $\mathcal{D}_{\text{bw,sub}}$ ermöglichen so die Plausibilisierung der Modellberechnungen. Die EG kann im Verlauf des Fahrzeugentwicklungszyklus kontinuierlich an veränderliche Datenräume und Datenmengen angepasst werden. Dadurch ist es möglich eine wachsende Menge an Trainingsdaten über diesen Zeitraum zu berücksichtigen. Durch die Ausführung der ML-Pipeline auf dem Cloud-Backend kann das ressourcenintensive Training der OCSVM problemlos durchgeführt werden. Die so erstellten EG können daraufhin mit sehr geringer Speicher- und Rechenkomplexität auf der Zielhardware ausgeführt werden. Somit ist eine Klassifizierung neuer Datenpunkte innerhalb des Steuergerätetaktes möglich.
Der Einsatz der RPA ermöglicht zudem die Erstellung sinnvoller Ersatzwerte. Diese können nachfolgenden Softwarefunktionen bereitgestellt werden, um den Rechenzyklus weiter durchführen zu können. Die Ersatzwertbildung liefert aufgrund der Berücksichtigung vorangegangener und als plausibel eingestufter Modellberechnungen Ersatzwerte in einem kleinen und zu den Modelleingängen passenden Bereich.
Die Erstellung der EG für die BRSs adaptiert veränderte Datenräume und skaliert automatisch mit den Trainingsdaten. Die RPA ist einer klassischen Überwachung des zulässigen Wertebereichs einer Funktion überlegen. Diese klassische Überwachung der berechneten Ausgangsgrößen von Softwarefunktionen auf dem Steuergerät betrachtet vorab definierte, zulässige Wertebereiche. Die RPA ist dahingegen zusätzlich in der Lage Modellberechnungen innerhalb dieser Wertebereich zu überprüfen und

als implausibel zu klassifizieren, auch wenn diese Werte innerhalb des definierten Wertebereichs liegen. Die RPA fügt der Überprüfung von Modellberechnungen eine zusätzliche Methodik zur Plausibilisierung datenbasierter Modelle hinzu.

6.2 Auswertung der Fallstudien

Die Ergebnisse der beiden Fallstudien werden anhand von zwei Szenarien bewertet. Beide Szenarien verwenden unterschiedliche Eingangsdaten aus dem Testdatensatz $\mathbb{S}_{\text{test}}$:

1. Modellinferenz mit Messgrößen der Fahrzeugsignale als Eingangsvektoren $\boldsymbol{x}$.
2. Modellinferenz mit ADAS-Horizontdaten als Eingangsvektoren $\boldsymbol{x}_{\text{HRC}}$.

Im ersten Szenario werden die ML-Modelle mit den aus den Messungen ermittelten Zielwerten $\boldsymbol{y}$ verglichen. So wird überprüft, ob mit der entwickelten ML-Pipeline eine vergleichbare Modellgüte erreicht werden kann. Die Zielgrößen ($\dot{Q}_{\text{Reib}}$, $\dot{Q}_{\text{Zyl}}$ und $\dot{Q}_{\text{Oel}}$) werden durch die PE-Modelle während der Messfahrt im Fahrzeugsteuergerät berechnet. Als Teil des prädiktiven TM stellen die PE-Modelle steuergerätinterne Größen bereit [16]. Durch eine zusätzliche softwareseitige Anpassung wurden diese internen Größen im Zuge dieser Arbeit ebenfalls auf den CAN-Bus gelegt und so als Messgröße verfügbar gemacht. Ein Vergleich der Berechnungen der ML-Modelle in Szenario 1 mit den Zielgrößen des Testdatensatzes $\mathbb{S}_{\text{test}}$ entspricht somit einem direkten Vergleich mit den PE-Modellen für den beschriebenen Anwendungsfall.
Im zweiten Szenario werden sowohl die PE-Modelle als auch die ML-Modelle auf ADAS-Horizontdaten in Form des Eingangsvektors $\boldsymbol{x}_{\text{HRC}}$ angewendet. Dadurch kann die Leistung der Modelle für den Anwendungsfall im prädiktiven TM evaluiert werden. Hierbei ist das Ziel für einen Eingangsvektor $\boldsymbol{x}_{\text{HRC}}$ aus dem MPP eine möglichst genaue Berechnung der zu erwartenden Wärmeströme $\dot{Q}_{\text{Reib}}$, $\dot{Q}_{\text{Zyl}}$ und $\dot{Q}_{\text{Oel}}$ in den KKL zu liefern. Der Zielvektor $\boldsymbol{y}$ setzt sich aus diesen drei, anhand der Messdaten ermittelten Wärmeströmen zusammen (vgl. Gl. 4.26).
Der Testdatensatz $\mathbb{S}_{\text{test}}$ (vgl. Kap. 3.3.1) beinhaltet für Szenario 2, zusätzlich zu den für das Training aufgezeichneten Fahrzeugmessgrößen, den MPP für jeden Zeitpunkt der Messung in Form eines 200 Einträge umfassenden Arrays. Diese positionsbasierten Arrays werden, wie in Kapitel 3.2.2 beschrieben, aufbereitet und in eine zeitbasierte Datenreihe überführt. In Abbildung 6.7 ist der grobe Verlauf der prädizierten Geschwindigkeit im Vergleich zu den tatsächlich gemessenen, fein aufgelösten Geschwindigkeitswerten dargestellt.
Für die *Fallstudien I & II* wurden Gitterstudien mit $n_{\text{Gitter}} \sim 20.000$ HPSs untersucht (vgl. Kap. 4.2.3). Die Ergebnisse sind in Tabelle 6.1 aufgeführt. Für *Fallstudie I* wird zusätzlich die Anwendung der RPA auf die Modellberechnungen dargestellt.

6.2.1 Szenario 1 - Bewertung der Modellgüte anhand des Testdatensatzes

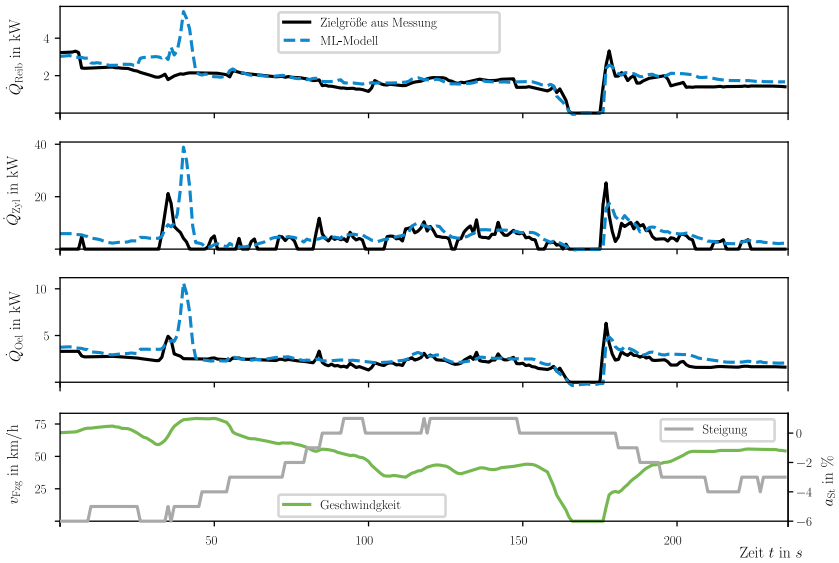


Abbildung 6.1: Vergleich der drei berechneten Ausgangswerte $\hat{y}_l$ des erstellten FNN mit den Zielgrößen y_l für die Inferenz mit Testdaten.
$RMSE_{\mathrm{Reib}} = 342\,\mathrm{W}$; $RMSE_{\mathrm{Zyl}} = 4496\,\mathrm{W}$; $RMSE_{\mathrm{Oel}} = 828\,\mathrm{W}$.

Für beide Fallstudien haben sich KNN mit $k = 3$ Ausgangsgrößen durchgesetzt. Modelle mit einer Ausgangsgröße ($k = 1$) konnten zusammen nicht die Modellgüte eines Modells mit $k = 3$ Ausgangsgrößen erreichen. Ein Grund hierfür liegt in der gekoppelten Berechnung der Ausgangsgrößen. Die Kombination aller Ausgangssignale in einem Modell liefert im Training zusätzliche Informationen über die Zusammenhänge der einzelnen Zielwerte. Durch ein kombiniertes Modell mit drei Ausgangsgrößen reduziert sich die Anzahl der benötigten Modelle und Parameter im Vergleich zu mehreren Modellen. Abbildung 6.1 zeigt die Modellberechnungen des als Ergebnis der ML-Pipeline ausgegebenen FNN aus *Fallstudie I*. Im Vergleich mit den Zielwerten $\boldsymbol{y}$ kann der Verlauf der Wärmeströme in der entsprechenden Größenordnung abgebildet werden. Es zeigen sich Abweichungen für die Berechnung von Maxima und für Bereiche hoher Dynamik. Speziell der Wärmestrom in den Wassermantel des Zylinderkopfs $\dot{Q}_{\mathrm{Zyl}}$, der aus den Verbrennungsprozessen im Zylinder resultiert, zeigt eine

deutliche Abweichung zum Zielwert in den Maxima und für dynamisch wechselnde Verläufe. Insbesondere das Maximum in der Nähe von $t \approx 40\,\mathrm{s}$ wird vom FNN für alle drei Zielgrößen überschätzt und versetzt dargestellt. Betrachtet man das zugehörige Fahrprofil in Abbildung 6.1 zeigt sich die Ursache in einem Geschwindigkeitsanstieg. Die Zielgrößen aus der Messung weisen dabei ein Maximum ungefähr an der Stelle des steilsten Geschwindigkeitsanstiegs auf, wohingegen die Berechnungen des FNN das Maximum erst beim Erreichen des Geschwindigkeitsplateaus ausgeben. Ein Grund dafür könnte in der übermäßigen Gewichtung des oszillierenden Steigungsverlaufs zu diesem Zeitpunkt durch das FNN liegen.
In Abbildung 6.2 werden die drei Ausgangssignale des als Ergebnis der ML-Pipeline für *Fallstudie II* ausgegebenen RNN dargestellt. Die Berechnung der Maxima durch das

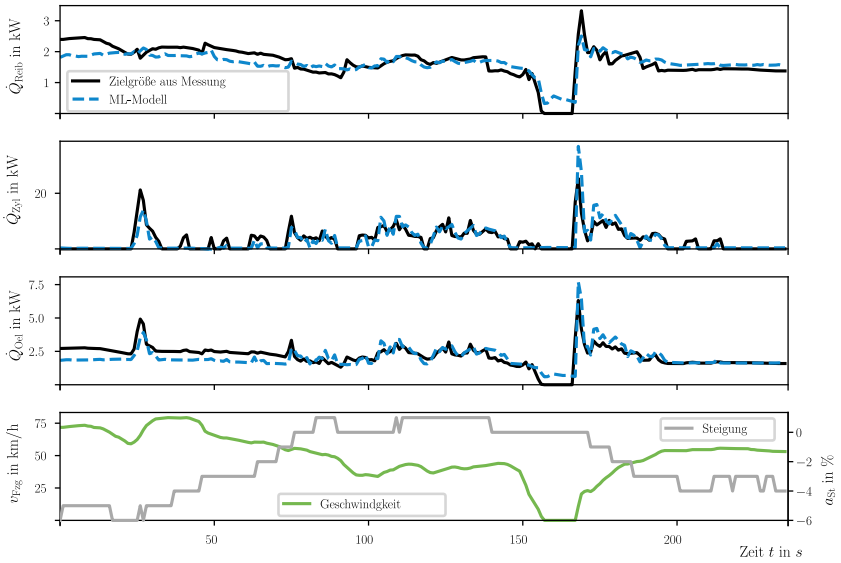


Abbildung 6.2: Vergleich der drei berechneten Ausgangswerte $\hat{y}_l$ des erstellten RNN mit den Zielgrößen y_l für die Inferenz mit Testdaten. $RMSE_{\mathrm{Reib}} = 336\,\mathrm{W}$; $RMSE_{\mathrm{Zyl}} = 3022\,\mathrm{W}$; $RMSE_{\mathrm{Oel}} = 566\,\mathrm{W}$.

RNN liefert deutlich genauere Übereinstimmungen mit den Zielwerten. Zudem ist kein zeitlicher Versatz wie in Abbildung 6.1 zu verzeichnen. Lokal können zudem deutlich bessere *RMSE*-Werte erzielt werden. Dies zeigt sich z.B. für $t \approx 40\,\mathrm{s}$. Dynamische Verläufe werden - verglichen mit den Berechnungen des FNN - deutlich genauer wie-

Tabelle 6.2: Eingangsgrößen als Ergebnisse der Gitterstudien für *Fallstudie I & II* (FNN & RNN).

		FNN	RNN
Eingangsgrößen			
Vorgegebene Eingangsgrößen:			
Fahrzeuggeschwindigkeit	v_{Fzg}	x	x
Streckensteigung	a_{St}	x	x
Fahrzeugmasse	m_{Fzg}	x	x
Zusätzliche variable Eingangsgrößen aus Feature Engineering:			
Mittelwert von v_{Fzg} über die letzten 10 Zeitschritte	$\overline{v}_{\mathrm{Fzg,10}}$	x	
Mittelwert von v_{Fzg} über die letzten 20 Zeitschritte	$\overline{v}_{\mathrm{Fzg,20}}$		x
Mittelwert von a_{St} über die letzten 10 Zeitschritte	$\overline{a}_{\mathrm{St,10}}$	x	
Mittelwert von a_{St} über die letzten 20 Zeitschritte	$\overline{a}_{\mathrm{St,20}}$		x
Gemittelte Ableitung von v_{Fzg} über die letzten 10 Zeitschritten	$\dot{v}_{\mathrm{Fzg,10}}$	x	
Gemittelte Ableitung von v_{Fzg} über die letzten 10 Zeitschritten	$\dot{v}_{\mathrm{Fzg,10}}$		x
Gemittelte Ableitung von a_{St} über die letzten 20 Zeitschritten	$\dot{a}_{\mathrm{St,20}}$	x	
Gemittelte Ableitung von a_{St} über die letzten 20 Zeitschritten	$\dot{a}_{\mathrm{St,20}}$		x
Zielgrößen			
Reibleistung	$\dot{Q}_{\mathrm{Reib}}$	x	x
Wärmestrom Zylinderkopf	$\dot{Q}_{\mathrm{Zyl}}$	x	x
Wärmestrom Getriebeöl	$\dot{Q}_{\mathrm{Oel}}$	x	x

dergegeben. Die größten Abweichungen sind für konstante Phasen mit den Zielwerten $\dot{Q}_{\mathrm{Reib}}$ und $\dot{Q}_{\mathrm{Oel}}$ nahe Null zu erkennen.
Es zeigt sich, dass RNN für Einsatz im PKW-TM eine bessere Modellgüte liefern. FNN liefern geringfügig schlechtere $RMSE$-Werte (vgl. Tab. 6.1), jedoch zum Preis eines deutlich höheren Speicherbedarfs. Die Parameterzahl ist um ein Vielfaches größer (vgl. Tab. 6.1). In *Fallstudie I* werden durch die Gitterstudie für das FNN andere zusätzliche Eingangssignale definiert als für das RNN in *Fallstudie II*. Wie in Tabelle 6.2 dargestellt, sind die zusätzlichen Eingangssignale des RNN jeweils über eine größere Anzahl an Zeitschritten (20 Zeitschritte) in der Historie der Messdaten gemittelt als die der FNN (10 Zeitschritte). Ein Grund dafür ist möglicherweise, dass das in *Fallstudie II* ermittelte RNN bereits eine Historie von $t_{\mathrm{RNN}} = 10$ Zeitschritten berücksichtigt. Zusätzliche Informationen über die Historie werden damit durch weiter zurückliegende Zeitschritte mit berücksichtigt. Im Vergleich dazu verarbeitet das FNN aus *Fallstudie I* die gemittelten Werte über 10 Zeitschritte als einzige Information über die Historie.
Betrachtet man weitere Gitterpunkte der Gitterstudie für *Fallstudie I*, so finden sich

FNN mit einer Parameteranzahl in der Größenordnung des RNN. Es zeigt sich, dass diese FNN zwar einen größeren $RMSE$ erzielen und daher nicht als bestes Ergebnis der Gitterstudie ausgegeben werden. In der Praxis können sie aber durchaus eingesetzt werden. Dies gilt vor allem in Anbetracht der praktischen Anwendung der Modelle auf grob aufgelöste ADAS-Horizontdaten in Form des Eingangsvektors $\boldsymbol{x}_{\mathrm{HRC}}$ und wird in Kapitel 6.2.2 bewertet.

Anwendung der RPA

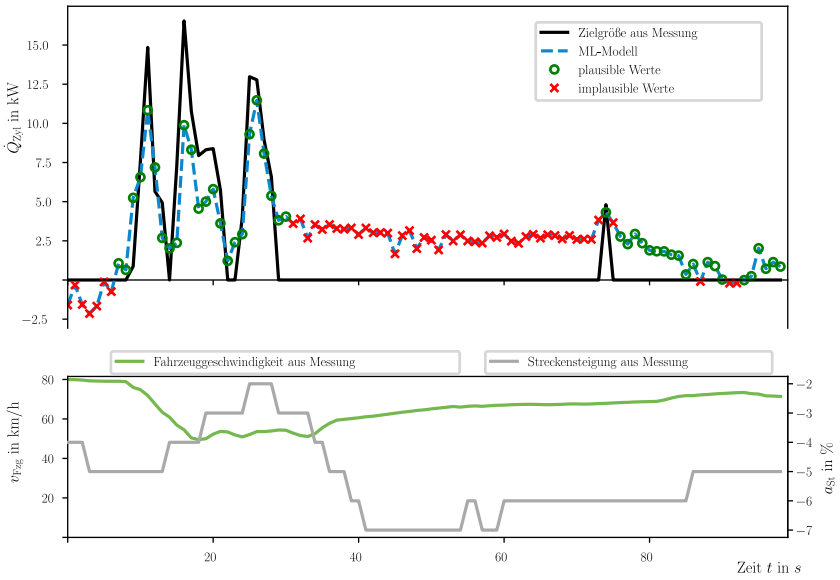


Abbildung 6.3: Die RPA identifiziert implausible Berechnungen des RNN. Für 10 Quantile mit zugehörigen BRSs. Jedes der 10 BRSs wird durch eine OCSVM mit den Parametern $\gamma = 99$ und $\nu = 0.05$ beschrieben.

Zur Einteilung der KNN Berechnungen in plausible und implausible Werte wird die RPA eingesetzt. Da FNN mit ReLUs Lipschitz-stetig sind (vgl. Kap. 5.3), können die berechneten Modellausgänge mit den verwendeten Eingangswerten $\boldsymbol{x}$ verknüpft werden. Die entstehenden BRSs werden zur Plausibilisierung der Modellberechnungen verwendet. In Abbildung 6.3 zeigt sich, dass offensichtlich falsch berechnete Werte (bspw. negative Wärmeströme $\dot{Q}_{\mathrm{Zyl}}$) von der RPA als implausibel (rot) markiert wer-

den. Die überschätzten Maxima werden genauso erkannt wie abweichende Werte für lange Konstantphasen. Die Anwendung der RPA liefert mit einmaligem Training der OCSVMs für die einzelnen BRSs eine einfache Methode, um die Modellberechnungen in plausible und implausible Werte zu unterteilen. Beste Ergebnisse in der Erkennung implausibler Werte wurden mit der Unterteilung des Ausgangsdatenraums in 20 äquidistante Intervalle erzielt. Für jedes Intervall wurde eine OCSVM für das zugehörige BRS trainiert. Verwendete Hyperparameter hierfür waren $\gamma = 99$ und $\nu = 0.05$. Wie in Kapitel 5.2 beschrieben steuert der Parameter γ die Straffheit der EG. In Abbildung 6.4 sind zwei solche Hyperebene (EG) zur Beschreibung eines dreidimensionalen Eingangsdatenraums $\mathcal{D}_{\text{ein}} \in \mathbb{R}^3$ dargestellt. Ein größerer Wert für γ erzeugt

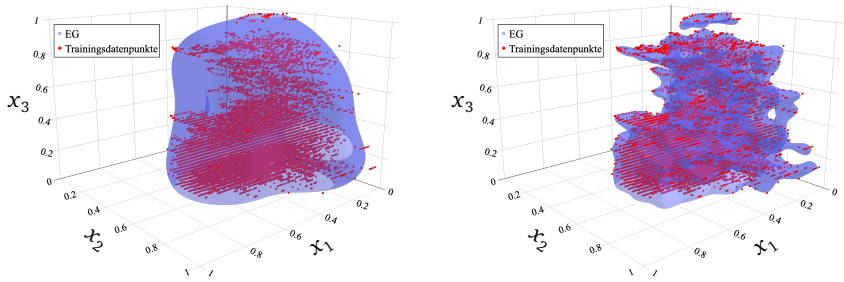


Abbildung 6.4: Die EG beschreibt als Hyperebene (blau) das dritte von 20 äquidistanten Intervallen des Trainingsdatensatzes $\mathbb{S}_{\text{train}}$ (rot).
links: $\gamma = 10$, $\nu = 0.001$; rechts: $\gamma = 175$, $\nu = 0.001$.

eine deutlich straffere Hyperebene. Das bedeutet, die einhüllende EG folgt der Form der Punktewolke enger als für kleinere Werte des Parameters γ. Der Parameter ν beschreibt den Anteil der Stützvektoren am Datensatz. Zudem beschreibt ν den Anteil an Datenpunkten, der nach dem Training der OCSVM zulässig außerhalb der Hyperebene liegen darf (vgl. Abb. 3.2). Die Anwendung der RPA auf die RNN Berechnungen liefert die in Abbildung 6.5 dargestellte Einteilung plausibler und implausibler Berechnungen. Die vom ML-Modell stark überschätzen Werte für den Zielwert $\dot{Q}_{\text{Zyl}} = 0$ werden als implausibel erkannt. Die RPA erkennt durchgängig Werte unterhalb 0 W und markiert überschätzte Berechnungen in Konstantphasen für FNN und RNN.

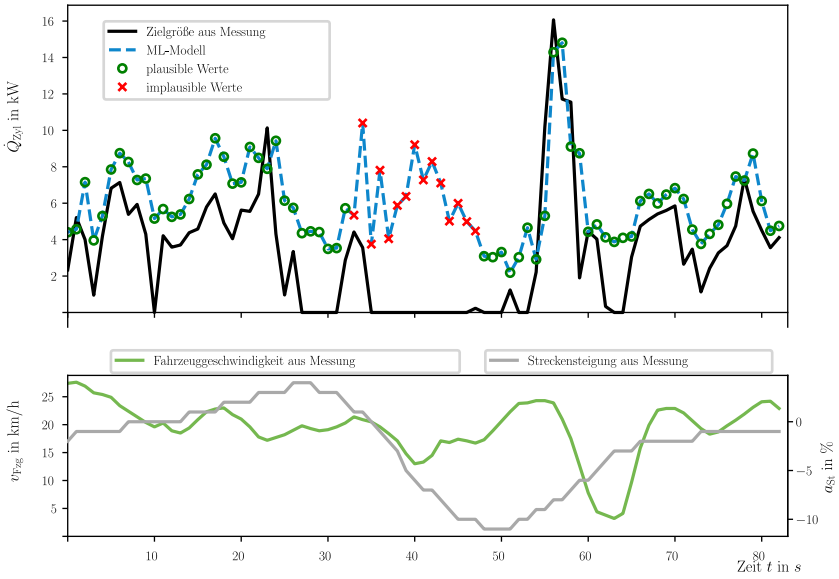


Abbildung 6.5: Die RPA identifiziert implausible Berechnungen des RNN. Für Quartile mit zugehörigen BRSs. Jedes der 4 BRSs wird durch eine OCSVM mit den Parametern $\gamma = 150$ und $\nu = 0.05$ beschreiben.

6.2.2 Szenario 2 - Vergleich von ML- und PE-Modellen bei Inferenz mit ADAS-Daten

Die als Ergebnisse aus den *Fallstudien I & II* hervorgegangenen ML-Modelle (FNN & RNN) werden zum Vergleich mit den PE-Modellen auf prädizierte Streckendaten aus dem ADAS-Horizont angewendet. Dazu wurde eine Messkampagne durchgeführt, in der speziell für jeden Messpunkt die 200 Einträge des zugehörigen Vorausschauvektors $\boldsymbol{x}_{\text{HRC}}$ aufgezeichnet wurden. Die Modellgüte der KNN in der Anwendung auf ADAS-Horizontdaten wird im Folgenden mit den derzeit im prädiktiven TM verwendeten PE-Modellen verglichen. Die PE-Modelle liefern Berechnungen für einen Horizont von 100 s, da in den Softwarefunktionen des prädiktiven TM nur ein Array mit 200 Einträgen für die zeitbasierten Berechnungen der PE-Modelle vorgehalten wird. Die Transformation der positionsbasierten ADAS-Vorausschaudaten in zeitbasierte Arrays findet mit einer Abtastrate von 2 Hz statt. Dies hat zur Folge, dass die zeitbasierte Prädiktion der TM-Modelle maximal 100 s in die Zukunft reicht. Für jeden Zeitpunkt des

100 s umfassenden Vorausschauhorizonts wird eine Modellberechnung durchgeführt. Die gesamte Länge des Vorausschauvektors geht in die Berechnung des $RMSE$ ein. Es zeigt sich, dass der $RMSE$ der ML-Modellberechnungen über die Länge des Vorausschauhorizonts geringer ausfällt als der $RMSE$ der PE-Modelle. Die höhere Modellgüte ermöglicht eine verbesserte Vorausschau. Dies gelingt bei gleichzeitig reduziertem Modellierungsaufwand durch die ML-Pipeline. Insbesondere wichtig ist die erwartete, über die Zeit eingetragene Energie (integrierter Wärmestrom) in den KKL für den MPP. Durch eine geringere Abweichung in Form eines geringeren $RMSE$ kann dieser genauer berechnet werden. Eine höhere Modellgüte trägt dazu bei, dass die Regelgrößen wie die Temperatur der Kühlflüssigkeit am Motorausgang $T_{\mathrm{mot,aus}}$ genauer abgeschätzt werden können. In Abbildung 6.6 sind die berechneten Komponenten des Wärmestroms $\dot{Q}_{\mathrm{in}}$ in den KKL dargestellt. In jeder Abbildung sind drei Signale abgebildet. Verglichen werden die Berechnung des RNN, die Berechnung des PE-Modells und die während der Messfahrt ermittelten Zielwerte $\boldsymbol{y}$ für jeden prädizierten Zeitpunkt des ADAS-Horizonts.

Die Zielwerte $\boldsymbol{y}$ weisen einen dynamischen Verlauf auf, da sie im Zuge der Messkampagne während Realfahrten gemessen wurden (vgl. Kap. 3.1.2). Die Eingangswerte $\boldsymbol{x}_{\mathrm{HRC}}$ des ADAS-Horizonts zeigen hingegen Verläufe mit langen Konstantphasen und sprunghaften Änderungen. Die ADAS-Horizontinformationen stellen einen konstanten Wert pro Abschnitt (Eventlänge im Horizont) zur Verfügung. Beispielsweise wird die zulässige Geschwindigkeit v_{lim} für einen Streckenabschnitt bereitgestellt (vgl. Abb. 6.6). Starke Ausschläge in der Geschwindigkeit oder der Steigung resultieren in KNN-Berechnungen für Extrempunkte weit oberhalb der Zielwerte. Dies liegt darin begründet, dass die Eingänge aus dem Geschwindigkeitslimit v_{lim}, Steigung a_{HRC} und Fahrzeugmasse m_{Fzg} bestehen. Sowohl v_{lim} als auch a_{HRC} liegen im ADAS-Horizont $\boldsymbol{x}_{\mathrm{HRC}}$ häufig als lange Konstantphasen vor. Die Fahrzeugmasse m_{Fzg} ist meist nur langsamen Veränderungen unterworfen und wird in einem Rechenzyklus für den ADAS-Horizont als konstant angenommen. Zusätzliche Eingangswerte sind im *feature engineering*, basierend auf diesen drei Signalen Werten entwickelt worden. In Tabelle 6.2 sind die Eingangssignale aufgelistet, welche im Zuge der Gitterstudien für FNN und RNN ausgewählt wurden. Mittelwerte und Ableitungen sind direkt abhängig von den Basiswerten Fahrzeuggeschwindigkeit v_{Fzg} und Steigung a_{St}. Liegt bei einem der Basiswerte ein konstanter Verlauf vor, so sind auch die Mittelwerte und Ableitungen dieses Wertes konstant bzw. null. Damit bleiben die Fahrzeugmasse m_{Fzg} und drei weitere Eingangssignale konstant. Das steigert den Einfluss des anderen Basiswertes überdurchschnittlich und bestimmt die Ergebnisse der Modellberechnungen maßgeblich.

Im Vergleich zu FNN können RNN diese Informationslage besser ausgleichen, da sie zusätzlich Informationen aus der Rückführung vorangegangener Zeitschritte berücksichtigen. Dadurch wird eine bessere Modellgüte in den Konstantphasen erreicht. Vergleicht man die dargestellten Modellberechnungen zeigt sich, dass KNN für alle drei Komponenten des Wärmestroms $\dot{Q}_{\mathrm{in}} = \dot{Q}_{\mathrm{Reib}} + \dot{Q}_{\mathrm{Zyl}} + \dot{Q}_{\mathrm{Oel}}$ bessere $RMSE$-Werte

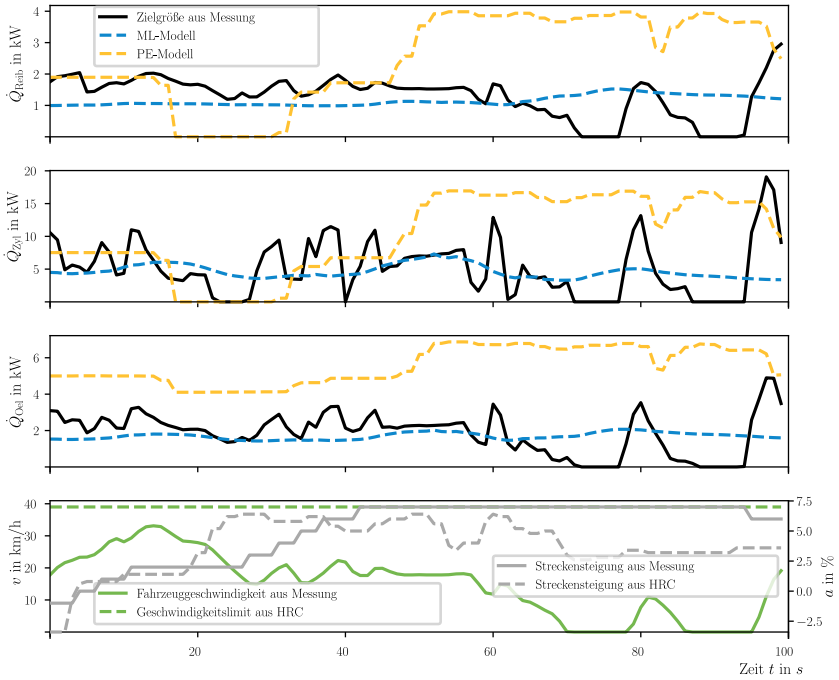


Abbildung 6.6: RNN: Vergleich der ML-Modell-Inferenz und der Berechnung der PE-Modelle mit ADAS-Horizontdaten einer Testmessung. Berechnet wurde der $RMSE$ für die ersten 100 s der ADAS-Horizontdaten.
$RMSE$-Werte des ML-Modells:
$RMSE_{\text{ML,Reib}} = 777\,\text{W}$; $RMSE_{\text{ML,Zyl}} = 4219\,\text{W}$; $RMSE_{\text{ML,Oel}} = 1194\,\text{W}$.
$RMSE$-Werte des PE-Modells:
$RMSE_{\text{PE,Reib}} = 2156\,\text{W}$; $RMSE_{\text{PE,Zyl}} = 9046\,\text{W}$; $RMSE_{\text{PE,Oel}} = 4135\,\text{W}$.

liefern. Die PE-Modelle liefern vor allem für die Komponente $\dot{Q}_{\text{Zyl}}$ gute Ergebnisse, können aber in der Gesamtheit keine höheren $RMSE$-Werte erreichen. Abbildung 6.7 zeigt hierfür konkret die Berechnung von $\dot{Q}_{\text{Zyl}}$ anhand von Vorausschaudaten aus dem ADAS-Horizont. Dabei fällt auf, dass die PE-Modelle vor allem im anfänglichen Horizontbereich eine gute Berechnungsgenauigkeit liefern, wohingegen die KNN für weiter in der Zukunft liegende Ereignisse deutlich besser generalisieren. Dies ist dadurch begründet, dass die PE-Modelle einen Startwert zum Rechnen verwenden, der dem

aktuellen, exakten Messwert zum Zeitpunkt $t_{0,\mathrm{HRC}}$ (erster Eintrag des Vorausschauvektors) entspricht. Diesen Messwert verwenden die KNN nicht, sondern rechnet vom ersten Zeitpunkt an mit den prädizierten ADAS-Horizontdaten. Dies erklärt, warum die PE-Modelle für die ersten Einträge des Vorausschauhorizonts eine gute Näherung der Messdaten zeigen. Betrachtet man die Energie, die durch den Wärmestrom

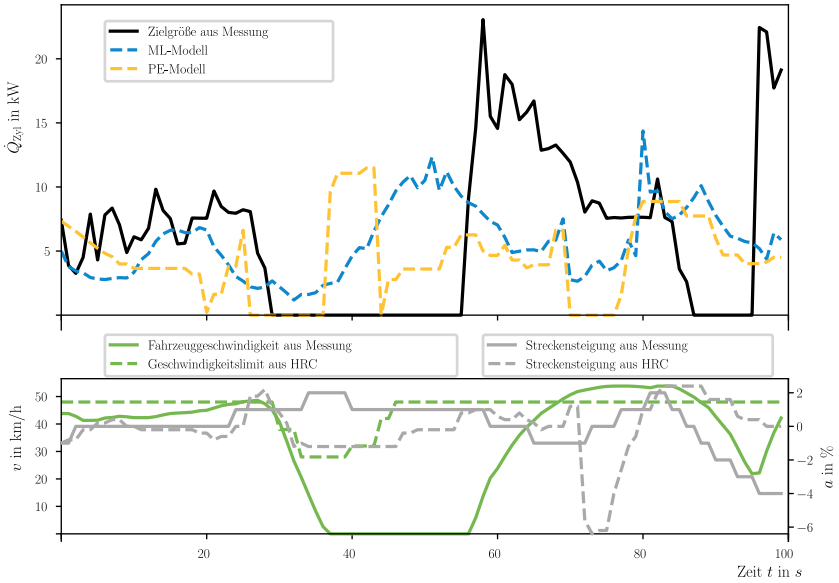


Abbildung 6.7: FNN: Vergleich der Modellberechnungen von PE-Modell und FNN für Inferenz mit ADAS-Horizontdaten. Der aus den Messungen ermittelte Zielwert zeigt den tatsächlichen Verlauf des Wärmestroms $\dot{Q}_{\mathrm{Zyl}}$. Im unteren Diagramm ist das zur Inferenz verwendete Fahrprofil in Form der prädizierten Geschwindigkeitslimits und Streckensteigung aus dem HRC dargestellt. Zum Vergleich sind die tatsächlich während der Messfahrt auf der prädizierten Strecke gemessenen Signale ebenfalls dargestellt.

$\dot{Q}_{\mathrm{in}} = \dot{Q}_{\mathrm{Reib}} + \dot{Q}_{\mathrm{Zyl}} + \dot{Q}_{\mathrm{Oel}}$ über die Zeit der Messung in den KKL eingetragen wird, zeigt sich, dass die ML-Modelle für die Anwendung auf ADAS-Daten eine geringere Abweichung als die PE-Modelle erzeugen. Für Abbildung 6.7 ist dieser Energieeintrag

(integrierter Wärmestrom) für $\dot{Q}_{\mathrm{Zyl}}$ in Tabelle 6.3 dargestellt. Es gilt

$$Q_{\mathrm{Zyl}} = \int_{t=0s}^{t=100s} \dot{Q}_{\mathrm{Zyl}} dt. \tag{6.1}$$

Berücksichtigt man die vorstehend beschriebene grobe Signalauflösung und Unsicherheiten bzgl. der tatsächlichen Fahrt im Vergleich zum MPP, ist die voraussichtlich in den KKL eingetragene Energie Q_{in} eine weitere wichtige Größe zur Beurteilung der Modellgüte. In Tabelle 6.1 ist daher für beide Fallstudien die Abweichung von der, aus den Zielgrößen ermittelten, eingetragenen Energie Q_{in} in den KKL dargestellt. Der *RMSE* ist das ausschlaggebende Gütekriterium in der Gitterstudie, da hiermit der detaillierte Verlauf der Modellberechnungen über die Zeitschritte der Messung hinweg berücksichtigt wird.

Tabelle 6.3: Eingetragene Energie (integrierter Wärmestrom) über ADAS-Horizont für $\dot{Q}_{\mathrm{Zyl}}$ aus Abbildung 6.7

	Zielgröße aus Messung	**PE-Modell**	**ML-Modell**
Integrierter Wärmestrom Q_{Zyl}	611 kJ	437 kJ	565 kJ
Abweichung von Zielgröße	–	28 %	7 %

Anwendung der RPA In Abbildung 6.7 ist dargestellt, mit welcher Unsicherheit die Modelle konfrontiert werden. Der ADAS-Horizont liefert jeweils eine Prognose für die wahrscheinlichste Fahrstrecke. Betrachtet wird hier der zeitbasierte Vorausschauvektor für die kommenden 100 s, da dieser in den Regelalgorithmen des prädiktiven TM verwendet wird. Der Vorausschauvektor kann keine unvorhergesehenen Stillstände, beispielsweise an Verkehrszeichen oder an einem Stauende bereitstellen. Dadurch kann sich die Modellberechnung für einen dieser Momentaufnahmen im Vergleich zur später tatsächlich eingetretenen Fahrsituation verschieben. Diese ist in Abbildung 6.7 dargestellt. Es ist gut ersichtlich, wie aus Stillstandsphasen mit $v_{Fzg} = 0\,\mathrm{km/h}$ eine Verschiebung der berechneten Maxima resultiert. In Abbildung 6.8 ist die Anwendung der RPA auf Berechnungen des FNN für Inferenz mit ADAS-Horizontdaten dargestellt. Die bereits beschriebenen Verschiebungen zwischen prädizierter und gefahrener Strecke sind auch hier deutlich sichtbar. Für die ersten ungefähr 40s liefert die RPA eine gute Plausibilisierung. Die in Abbildung 6.7 dargestellten Abschnitte mit $\dot{Q}_{\mathrm{Zyl}} = 0\,\mathrm{kW}$ resultieren aus der *Start-Stopp-Automatik* im Fahrzeug. Durch die Abschaltung des Verbrennungsmotors während Standphasen ist der Wärmestrom aus dem Zylinderkopf in den KKL $\dot{Q}_{\mathrm{Zyl}} = 0\,\mathrm{kW}$. In der Anwendung im prädiktiven TM wird diese Situation

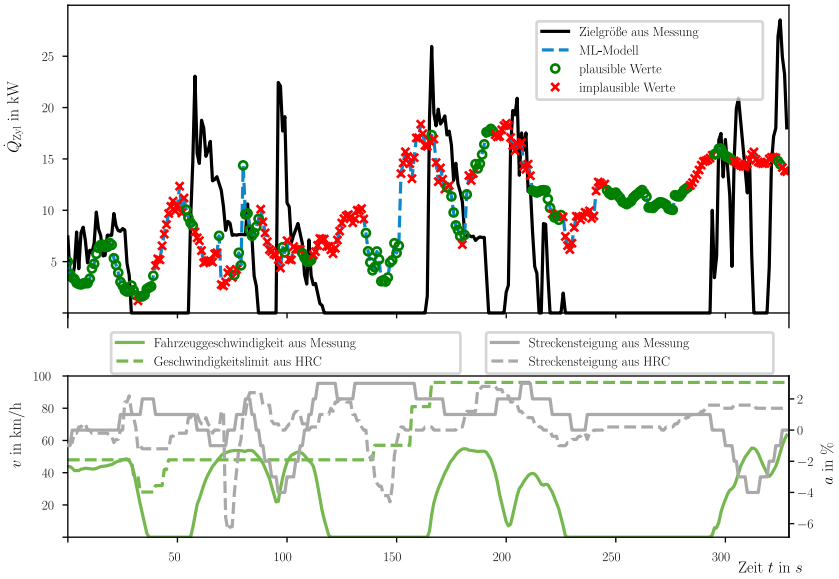


Abbildung 6.8: Anwendung der RPA auf Ergebnisse der FNN-Inferenz mit ADAS-Horizontdaten. Der aus den Messungen ermittelte Zielwert zeigt den tatsächlichen Verlauf des Wärmestroms $\dot{Q}_{Zyl}$.

entschärft, da der ADAS-Horizont mit einer Frequenz von 10 Hz aktualisiert wird, somit werden aktuell herrschende Stillstands- oder Konstantphasen als neuer Startwert $t_{0,HRC}$ berücksichtigt. Für die weiter in der Zukunft liegenden Ereignisse kann dies jedoch nicht vorab geschehen. Durch diesen Umstand in Kombination mit den grob abgestuften Eingangssignalen lässt sich nur eine eingeschränkte Modellgüte erreichen. In *Fallstudie II* zeigt sich, dass RNN mit einem großen Zeitfenster für die Berücksichtigung vorangegangener Rechenschritte die Dynamik der realen Messgrößen nicht in vollem Umfang abbilden. Das Niveau des Wärmestroms errechnen diese Modelle im Mittel jedoch genauer als die PE-Modelle des prädiktiven TM. Dadurch kann die über die Zeit in den KKL eingetragene Energie (integrierter Wärmestrom) gut abgeschätzt werden. Dieser ist eine essentielle Größe, um die Auswirkungen auf die Bauteile des Kühlsystems für den MPP im Voraus abzuschätzen. Eine hohe Dynamik ist per se nicht von den eingegebenen ADAS-Horizontdaten abzuleiten. Die PE-Modelle liefern zwar sehr dynamische Ergebnisse, über- und unterschätzen jedoch häufig Extrempunkte und liefern so - über die Zeit des Vorausschauhorizonts betrachtet - keine genaue

Vorhersage über dieeingetragene Energie (integrierter Wärmestrom) für den MPP.

6.3 Diskussion der Ergebnisse

Die vorgestellten Ergebnisse zeigen eine automatisierte Modellierung, basierend auf Fahrzeugmessdaten. Die ML-Pipeline liefert KNN und eine RPA zur effizienten Plausibilisierung der KNN-Berechnungen. Die Automatisierung der Modellierung ist in vollem Umfang durchgeführt und erfüllt die zugehörige Forschungsfrage aus Kapitel 1.2. Der manuelle Aufwand zur Kalibrierung von Modellparametern entfällt. Zusätzlich ermöglicht die Anwendung der RPA eine Plausibilisierung der Modellberechnungen. Implausible Werte können entweder durch Ersatzwerte substituiert werden oder sie aktivieren die vorhandene Rückfallebene des prädiktiven TM. Die Bewertung der Inferenz anhand von Testdaten (vgl. Kap. 6.2 & 6.2.1) zeigt, dass die ML-Modelle eine mit den PE-Modellen vergleichbare Modellgüte erreichen. Im vorgesehenen Anwendungsfall, der Inferenz unter Verwendung von ADAS-Horizontdaten im prädiktiven TM, erzielen die ML-Modelle eine bessere Modellgüte als die PE-Modelle (vgl. Kap. 6.2.2). In Kombination mit der RPA kann eine weitere Steigerung in der Modellgüte erzeugt werden. Das als Ergebnis der *Fallstudie II* erstellte RNN besitzt mit einer Parameteranzahl von $n_{\text{param}} = 2563$ weniger als die Hälfte der Parameter des bisher verwendeten PE-Modells des Verbrennungsmotors. Das RNN liefert als Modellausgang direkt die Komponenten des Wärmestroms $\dot{Q}_{\text{in}}$ in den HT-KKL. Bisher kamen dafür im prädiktiven TM neben dem PE-Modell des Verbrennungsmotors noch weitere Modelle zum Einsatz. Diese PE-Modelle benötigen zusätzlich Kennfelder und Kennlinien zur Berechnung des Wärmestroms. Durch eine hohe Anzahl an Stützstellen ermöglichen die Kennfelder eine hohe Genauigkeit in der Berechnung mit exakten Fahrzeugsignalen als Modelleingang. Im Vergleich dazu vereinen die ML-Modelle all diese Informationen und Abhängigkeiten in einem KNN.
Die Ergebnisse für die Anwendung auf den Testdatensatz $\mathbb{S}_{\text{test}}$ zeigen, dass die FNN und RNN eine ähnliche Modellqualität bei unterschiedlicher Modellgröße und Speicherbedarf erreichen können (vgl. Kap. 6.2.1).
Die Bewertung der Modellgüte für die Anwendung auf ADAS-Horizontdaten zeigt, dass die PE-Modelle eine niedrigere Modellgüte erreichen. Für diesen Anwendungsfall liefern die ML-Modelle eine deutlich höhere Genauigkeit über weitere Abschnitte des ADAS-Horizonts (vgl. Kap. 6.2.2).
Die Auswertung der Ergebnisse zeigt, dass die Substitution der derzeit im prädiktiven TM verwendeten PE-Modelle durch ML-Modelle einen deutlichen Nutzen in der Anwendung hat. Für die folgenden Bereiche in der Modellentwicklung des PKW-TM konnte damit ein Mehrwert generiert werden:

1. Die Sammlung der benötigten Trainingsdaten auf dem Cloud-Backend findet kontinuierlich und automatisiert während jeder Messfahrt mit dem Entwicklungs-

fahrzeug statt. Dafür werden das serienmäßig verfügbare Kommunikationsmodul und das Cloud-Backend der *Mercedes-Benz Group AG* Fahrzeugentwicklung genutzt. Damit kann die entwickelte Methodik im Arbeitsalltag direkt eingesetzt werden.
2. Der manuelle Kalibrierungsaufwand der Applikationsparameter wird durch die Verwendung der ML-Pipeline reduziert. Anstatt einer aufwendigen Modellierung und Kalibrierung der Applikationsparameter müssen lediglich Ein- und Ausgangswerte sowie die Hyperparameter definiert werden.
3. Die automatisiert erstellten ML-Modelle bieten im vorgesehenen Anwendungsfall auf ADAS-Horizontdaten durchgängig geringere $RMSE$-Werte und damit eine höhere Modellgüte für das prädiktive TM.
4. Durch die automatisch erstellte RPA kann jeder berechnete Datenpunkt mit OCSVMs plausibilisiert werden. Die OCSVMs weisen sich durch eine geringe Speicher- und Rechenkomplexität in der Anwendung aus.
5. Die Automatisierung und Standardisierung des Modellierungsprozesses anhand datenbasierter Methoden fügt sich in die bisherige Entwicklung von Softwarefunktionen für das PKW-TM ein. Teile der entwickelten Methodik werden bereits in der Entwicklungsabteilung angewendet. Dazu gehört das automatisierte Sammeln und Aufbereiten der Trainingsdaten im Cloud-Backend. Das automatisierte Modelltraining wird in einer Pilotanwendung eingesetzt.
6. Die Modellauswahl erfolgt automatisiert anhand skalarer Gütekriterien und ist durchgängig nachvollziehbar. Die Methodik stellt abschließend das ausgewählte beste Modell als Ergebnis bereit.

Der Einsatz von ML-Modellen bietet Vorteile in Bezug auf eine einfache Skalierbarkeit, den hohen Automatisierungsgrad und die einfache Anpassungsfähigkeit an sich verändernde Anwendungsfälle. So kann durch Erstellen einer neuen Datenbasis und Definition der Eingangs- und Zielgrößen sehr einfach eine automatisierte Modellerstellung vorgenommen werden. Dadurch kann die Methodik zur Entwicklung neuer ML-Modelle auf zukünftige Problemstellungen übertragen werden. Da die ML-Pipeline modular aufgebaut ist, kann sie dafür erweitert oder verändert werden. Beispielsweise kann die verwendete Gitterstudie zur Optimierung der Hyperparameter bei Bedarf durch einen EA oder eine PSO ersetzt werden.
Die Auslagerung des Modelltrainings auf das Cloud-Backend ermöglicht ein effizientes Training von ML-Modellen. Dadurch dass Hyperparameterkombinationen auf performanten Rechenknoten parallelisiert bewertet werden können, kann die Entwicklungszeit im Vergleich zur bisherigen manuellen Modellierung reduziert werden. Durch die kontinuierliche Erweiterung des Trainingsdatensatzes und ein iteratives Training können verbesserte ML-Modelle während des Entwicklungszyklus eines Fahrzeugs parallel zu Fahrzeugerprobung trainiert und bereitgestellt werden.
In dieser Arbeit wurde berücksichtigt, dass aktuelle Steuergeräte meist wenig freien Speicherplatz und Rechenleistung bieten. Daher wurden bevorzugt kleinere Modelle

niedriger Komplexität erstellt. Für kommende Steuergerätegenerationen und Fahrzeugrechner kann die entwickelte Methodik ML-Modelle höherer Komplexität und Größe liefern. Dadurch lässt sich die Modellgüte voraussichtlich weiter steigern.
Die entwickelte Methodik zeigt, wie zukünftige Entwicklungsmethoden und die Weiterentwicklung von bereits ausgelieferter Software aussehen können.

7 Zusammenfassung und Ausblick

In dieser Arbeit wurde eine Methodik zur datenbasierten Entwicklung von Regelungsmodellen im PKW-TM entwickelt. Dadurch wurde der bisherige manuelle Modellierungsprozess der PE-Modelle durch eine ML-Pipeline ersetzt. Damit wurde die Entwicklung unterschiedlicher ML-Modelle, welche für die große Zahl an Fahrzeug- und Antriebsvarianten nötig sind, vereinfacht und beschleunigt.
Um eine performante und skalierbare Lösung bereitzustellen, wurden die datenbasierten Entwicklungsmethoden auf einem Cloud-Backend implementiert. Dort konnte die Entwicklung neuer ML-Modelle mittels der ML-Pipeline automatisiert durchgeführt werden. Als Datengrundlage für das Training dieser Modelle wurden Fahrzeugmessdaten verwendet, welche kontinuierlich über das serienmäßig verfügbare Kommunikationsmodul vom Fahrzeug in das Cloud-Backend übertragen wurden.
Die Entwicklung der ML-Pipeline und die durchgeführten *Fallstudien I & II* dienten dabei zur Untersuchung der ersten beiden Forschungsfragen:

- Kann der manuelle Kalibrierungsaufwand für Regelungsmodelle durch den Einsatz von ML-Modellen reduziert werden?
- Kann mit dem Einsatz von ML-Modellen eine vergleichbare Modellqualität erzielt werden?

Hierfür wurden innerhalb der ML-Pipeline eine Datenvorverarbeitung und ein *feature engineering* durchgeführt. Daraufhin folgte die Auswahl und das Training der ML-Modelle, deren Modellgüte mit den bisher verwendeten PE-Modellen verglichen wurde. Die Zielanwendung der ML-Modelle ist die Inferenz mit Eingangssignalen aus dem ADAS-Horizont. Dieser stellt Informationen über die wahrscheinlichste Fahrstrecke zur Verfügung.
Um den Einsatz der ML-Modelle im PKW-TM plausibilisieren zu können, wurde zudem eine neue Methodik entwickelt. Diese stellte eine Lösung für die dritte Forschungsfrage dar:

- Können die Modellberechnungen der ML-Modelle mit geringem Rechenaufwand plausibilisiert werden?

Die entwickelte RPA klassifiziert die Berechnungen der ML-Modelle als plausibel oder implausibel.

Für die Entwicklung der ML-Modelle wurden zwei für das Gesamtsystem geeignete Modellarchitekturen ausgewählt und in zwei Fallstudien verglichen.

In *Fallstudie I* wurden FNN untersucht. Sie eignen sich auf Grund ihrer universellen Fähigkeit, beliebige mathematische Funktionen und Zusammenhänge approximieren zu können. Ihr Aufbau bietet die Möglichkeit, kleine Modelle geringer Komplexität zu erstellen. In Kombination mit ReLUs sind FNN Lipschitz-stetig. Dies ist die Basis für die Entwicklung und Anwendung der RPA.
In *Fallstudie II* wurden RNN verwendet. Die rekurrente Strukturen ermöglichen es, Informationen aus vorangegangenen Rechenschritten für die aktuelle Berechnung des Modellausgangs zu berücksichtigen. Damit können sie zum Preis höherer Modellkomplexität eine besser Modellgüte liefern.
Jede der beiden *Fallstudien I & II* liefert als Ergebnis ein ML-Modell. Diese beiden ML-Modelle wurden mit den bisher eingesetzten PE-Modellen verglichen. Für die Inferenz mit Testdaten lieferten die ML-Modelle eine vergleichbare Modellgüte. Für die Inferenz mit ADAS-Horizontdaten gaben die erzeugten ML-Modelle eine genauere Berechnung des zu erwartenden Wärmestroms in den Kühlkreislauf des Fahrzeugs über die Länge des prädizierten ADAS-Horizonts wieder. Dies unterstützt eine energieeffizientere Regelung der Aktoren in den Kühlkreisläufen, da mit erhöhter Modellgüte die ADAS-Horizontdaten zuverlässiger interpretiert werden können. In der Praxis kann dadurch eine genauere Vorkonditionierung der Komponenten des Kühlkreislaufs stattfinden. Besonders die Temperaturregelung von Komponenten wie der Traktionsbatterie profitiert von einer verbesserten Berechnung des erwarteten Wärmestroms in den KKL. Die Kenntnis des erwarteten Wärmestroms hat Einfluss auf die Ladezeit und den Energieverbrauch, beispielsweise bei der Vorkonditionierung der Traktionsbatterie durch den Chiller. Dies gelingt bei gleichzeitig reduziertem Modellierungsaufwand durch die automatisierte Datensammlung und Modellierung mittels der ML-Pipeline.

Um die Modellberechnungen zu plausibilisieren, wurde die RPA entwickelt und angewendet. Durch den Einsatz der RPA konnten die berechneten Modellausgänge als plausibel oder implausibel klassifiziert werden. Der Algorithmus arbeitet mit OCSVMs und klassifiziert Eingangsvektoren als *innerhalb* oder *außerhalb* eines multidimensionalen Datenraums. Dazu wurden valide Ausgangsdatenräume anhand der Trainingsdaten definiert und mit BRSs verknüpft. Das rechenzeitintensive Training der OCSVMs wurde auf dem Cloud-Backend durchgeführt. Dagegen konnte die Ausführung der trainierten OCSVMs mit geringem Speicher- und Ressourcenbedarf durchgeführt werden, da sie mittels weniger Stützvektoren eine multidimensionale Hyperebene (Entscheidungsgrenze) beschreiben. Die Klassifizierung von Datenpunkten als *innerhalb* oder *außerhalb* der EG ist dank des sogenannten *kernel tricks* rechenzeiteffizient möglich. In der Anwendung auf die Modellberechnungen für Test- und ADAS-Daten konnte die RPA vor allem unter- bzw. überschätzte Minima bzw. Maxima als implausibel erkennen. Zusätzlich ermöglichte die RPA die Klassifizierung implausibler Modellberechnungen für längere Konstantphasen. Der Algorithmus trug so zu einer höheren Modellgüte für die prädiktive Berechnung des Wärmestroms in die Kühlflüssigkeit bei.

In einer abschließenden Betrachtung der in Kapitel 1.2 definierten Forschungsziele zeigte sich, dass die entwickelte ML-Pipeline eine automatisierte Modellentwicklung ermöglicht. Die damit erstellten ML-Modelle konnten für den Zielanwendungsfall mit ADAS-Horizontdaten eine verbesserte Modellgüte für das prädiktive TM liefern. Mit der RPA wurde eine effiziente Möglichkeit zur Plausibilisierung von ML-Modellen bereitgestellt. Die nun verfügbare Methodik kann in Zukunft zum Training von ML-Modellen für verschiedene Fahrzeugbaureihen mit verschiedenen Antriebstechnologien verwendet werden. Dadurch bietet sie einen großen Mehrwert über den untersuchten Anwendungsfall hinaus. Die ML-Pipeline und die Logik zur Plausibilisierung der Berechnungen der ML-Modelle kann auf andere datenbasierte Modellierungsansätze übertragen werden.

Ausblick Die Erstellung einer Datenbasis für verschiedene Fahrzeuge oder Fahrzeugflotten eröffnet die Option, über unterschiedliche Fahrzeugvarianten hinweg Modelle mit guter Generalisierungsfähigkeit zu erstellen. Diese könnten in der Entwicklung kommender Fahrzeuggenerationen als Grundlage für ein neues Modelltraining verwendet werden. Darüber hinaus birgt die Klassifikation verschiedener Kundengruppen und deren Nutzungsgewohnheiten ein großes Potenzial. Für verschieden Gruppen könnten optimierte ML-Modelle bereitgestellt werden, welche die Komponenten im KKL länger im optimalen Betriebsbereich halten und so Alterung und Verschleiß reduzieren. Dadurch ergibt sich die Möglichkeit, kundenindividuelle Lösungen zu erstellen, die über die Funktionsentwicklung hinaus im Kundenfahrzeug aktualisiert werden können.
Mit Hinblick auf kommende Steuergerätegenerationen im PKW und einer damit einhergehenden Steigerungen der Rechenleistung sowie des Speicherplatzes lassen sich in Zukunft komplexere und größere Modelle verwenden. Die entstandene ML-Pipeline kann als Grundlage für die Entwicklung vieler weiterer ML-Anwendungen, über das PKW-TM hinaus, in der Antriebsstrangsoftware eingesetzt zu werden.

A Anhang

Physikalische Ersatzmodelle des prädiktiven TM

Zur Berechnung der Motordrehzahl N_{mot} wird nach [16] in den TM-Modellen Gl. A.1 verwendet.

$$N_{\text{mot}} = \frac{v_{Fzg} \cdot 30 \cdot i_G \cdot i_A}{3,6 \cdot \pi \cdot r_{dyn}} \tag{A.1}$$

Dabei ist i_A die Achsübersetzung und i_G die Getriebeübersetzung. Die Größe i_G lässt sich direkt aus dem prädizierten Getriebegang ableiten. Dieser wird über statische Kennlinien in Abhängigkeit von der prädizierten Fahrzeuggeschwindigkeit v_{HRC} bestimmt [16]. Diese Kennlinie wurde in [29] erfolgreich mit einem KNN ersetzt. Diese Lösung bietet den Vorteil, dass keine manuelle Ermittlung der Stützstellen für die Kennlinie durchgeführt werden muss, sondern anhand von Messdaten aus dem Erprobungsträger die Beziehung zwischen Gangzustand und den verfügbaren Eingangsgrößen aus dem HRC gelernt werden kann.
Der dynamische Radhalbmesser r_{dyn} ergibt sich aus

$$r_{dyn} = \frac{s_{Fzg}}{2 \cdot \pi \cdot n_{Rad}}. \tag{A.2}$$

Die zurückgelegte Wegstrecke s_{Fzg} des Fahrzeugs und die benötigte Anzahl an Radumdrehungen sind die einzigen Eingänge [16]. Diese können anhand der prädizierten Fahrtstrecke aus dem HRC sowie dem konstanten und als bekannt vorauszusetzenden Reifendurchmesser bestimmt werden. Ein weiterer Faktor ist die (prädizierte) Fahrzeuggeschwindigkeit. Das Motordrehmoment M_{mot} wird über die Fahrwiderstände ermittelt.

$$M_{\text{mot}} = \frac{r_{dyn} \cdot (F_M + F_S + F_R + F_L)}{i_A \cdot i_G \cdot \eta_{Antr}} \tag{A.3}$$

Diese hängen vom Fahrzeugaufbau, der Motorleistung und dem Gewicht sowie von der Steigung der Fahrtstrecke ab. Die Widerstandsgröße F_M bezeichnet die d'Alembertsche Trägheitskraft. F_S repräsentiert den Steigungswiderstand; F_R den Rollwiderstand und F_L den Luftwiderstand des Fahrzeugs. Genauere Ausführungen sind in [16] und in weiterführender Fachliteratur zu finden. Der Wirkungsgrad des gesamten Fahrzeugantriebsstrangs ist η_{Antr}. Die an einem Rad anliegende Leistung ist durch

$$P_{Rad_{max}} = P_{mot_{max}} \cdot \eta_{\text{Antr}} = (F_M + F_S + F_R + F_L) \cdot v_{Fzg} \tag{A.4}$$

definiert.
Mittels Gleichungen A.4 und den zugehörigen Fahrwiderständen und Konstanten werden die TM-Modelle in [16] gebildet.

B Literaturverzeichnis

Literatur

[1] *Verordnung (EU) 2019/631 des Europäischen Parlaments und des Rates vom 17. April 2019 zur Festsetzung von CO2-Emissionsnormen für neue Personenkraftwagen und für neue leichte Nutzfahrzeuge und zur Aufhebung der Verordnungen (EG) Nr. 443/2009 und (EU) Nr. 510/2011 (ABl. L111 vom 25.04.2019)*, 2019.

[2] K. Schmiederer, „Thermomanagement als Zukunftsaufgabe im Automobilbau“, *ATZextra*, Jg. 16, Nr. 6, 2011. DOI: 10.1365/s35778-011-0596-0.

[3] C. Rammer, *Auf Künstliche Intelligenz kommt es an*, Bundesministerium für Wirtschaft und Energie (BMWi), Hrsg. Berlin, 2020.

[4] D. Czarnitzki, G. P. Fernández und C. Rammer, „Artificial Intelligence and Firm-level Productivity“, *ZEW - Centre for European Economic Research Discussion Paper No. 22-005*, 2022. DOI: 10.2139/ssrn.4049824.

[5] Mercedes-Benz Group AG. „Die wichtigsten Fragen zu Flottenverbrauch/-emissionen: CO2-Ziele werden weltweit immer anspruchsvoller“. (online), Adresse: https://group-media.mercedes-benz.com/marsMediaSite/de/instance/ko/Die-wichtigsten-Fragen-zu-Flottenverbrauch-emissionen-CO2-Ziele-werden-weltweit-immer-anspruchsvoller.xhtml?oid=42508702 (besucht am 12. 02. 2023).

[6] Mercedes-Benz Group AG. „Mercedes Steigert E-Absatz Um Fast 70 Prozent“. (online), Adresse: https://www.electrive.net/2022/01/10/mercedes-steigert-e-absatz-um-fast-70-prozent/ (besucht am 21. 08. 2022).

[7] Mercedes-Benz Group AG. „Mercedes-Benz Strategy Update: Electric drive“. (online), Adresse: https://group.mercedes-benz.com/company/strategy/mercedes-benz-strategy-update-electric-drive.html (besucht am 12. 02. 2023).

[8] Mercedes-Benz Group AG. „Absatz Mercedes-Benz 2021“. (online), Adresse: https://group.mercedes-benz.com/system/browser-not-supported/de/?t=0.d580317.1661089608.90e08f8d (besucht am 21. 08. 2022).

[9] C. Kerkhoff, „Technologieoffenheit ist notwengig - Interview mit Prof. L. Eckstein“, ATZ extra, Jg. Sonderheft für den VDI-FVT - Künstliche Intelligenz, 2022.

[10] ISO, „Road vehicles – Functional safety“, Genf, Schweiz, ISO 26262, 2018.

[11] F. Blank, F. Hüger, M. Mock und T. Stauner, „Methodik zur Absicherung von KI iIm Fahrzeug“, *ATZ - Automobiltechnische Zeitschrift*, Jg. 124, Nr. 7, 2022. DOI: 10.1007/s35148-022-0879-3.

[12] A. Kampker, D. Vallée und A. Schnettler, Hrsg., *Elektromobilität: Grundlagen einer Zukunftstechnologie*. Berlin, Heidelberg: Springer, 2018. DOI: 10.1007/978-3-662-53137-2.

[13] D. Watzenig und B. Brandstätter, *Comprehensive Energy Management - Safe Adaptation, Predictive Control and Thermal Management*. Cham: Springer International Publishing, 2018. DOI: 10.1007/978-3-319-57445-5.

[14] S. Pischinger und U. Seiffert, Hrsg., *Vieweg Handbuch Kraftfahrzeugtechnik*. Wiesbaden: Springer Fachmedien Wiesbaden, 2021. DOI: 10.1007/978-3-658-25557-2.

[15] F. Wolf, *Fahrzeuginformatik*. Wiesbaden: Springer Fachmedien Wiesbaden, 2018. DOI: 10.1007/978-3-658-21224-7.

[16] M. Stöcker, „Modellbasiertes Thermomanagement mit Navigationsdaten zur Energieeffizienzsteigerung der Fahrzeugkühlung“, Diss., Technische Universität Darmstadt, 2018.

[17] M. Reichenbach, „Emissionsfreie Mobilität durch intelligentes Thermomanagement - Interview mit Dr. A. Wiebelt“, *ATZ - Automobiltechnische Zeitschrift*, Jg. 116, Nr. 7-8, 2014. DOI: 10.1007/s35148-014-0444-9.

[18] Q. Hu, M. R. Amini, H. Wang, I. Kolmanovsky und J. Sun, „Integrated Power and Thermal Management of Connected HEVs via Multi-Horizon MPC“, *American Control Conference (ACC)*, 2020.

[19] H. Wang, Y. Meng, Q. Zhang et al., „MPC-Based Precision Cooling Strategy (PCS) for Efficient Thermal Management of Automotive Air Conditioning System“, *CoRR*, 2019. DOI: 10.48550/arXiv.1906.04006.

[20] M. Altendorfner, „Pareto-Optimierung komplexer Thermo-Fluiddynamischer Systeme auf Basis numerischer Berechnungen“, Diss., Universität Erlangen-Nürnberg, 2008.

[21] S. Singh Romana, „Automated Optimization for Model Parameters of Predictive Cooling Circuit Models“, Masterarbeit, Technische Universität Darmstadt, 2019.

[22] S. Zaglauer, „Methode zur multikriteriellen Optimierung des Motorverhaltens anhand physikalisch motivierter Modelle“, Diss., Friedrich-Alexander-Universität Erlangen-Nürnberg, 2014.

[23] R. Sauermann, D. Boja, F. Kirschbaum und O. Nelles, „Particle Swarm Optimization for Automotive Model-Based Calibration“, *IFAC Proceedings Volumes*, Jg. 43, Nr. 7, 2010. DOI: 10.3182/20100712-3-DE-2013.00099.

[24] O. Nelles, *Nonlinear System Identification: From Classical Approaches to Neural Networks and Fuzzy Models*. Springer, 2001.

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

[26] C. C. Aggarwal, *Artificial Intelligence*. Cham: Springer International Publishing, 2021. DOI: 10.1007/978-3-030-72357-6.

[27] M. Sturm, „Neuronale Netze Zur Modellbildung in Der Regelungstechnik“, Diss., Technische Universität München, 2000.

[28] M. Handreg, „Entwurf von Künstlichen Neuronalen Netzen zur Regelung von Prozessgrößen in einer Schmelzwanne für Flachglas“, Diss., Technische Universität Cottbus-Senftenberg, 2016.

[29] D. Wolf, „Entwurf, Verknüpfung und Bewertung von Machine Learning-Modellen Für Den Einsatz Im Vorausschauenden Thermomanagement“, Masterarbeit, Universität Stuttgart, 2020.

[30] S. Wohlang und Bundesregierung, „Strategie Künstliche Intelligenz der Bundesregierung“, Bundesministerium für Wirtschaft und Klimaschutz (BMWK), 2020.

[31] S. Studer, T. Bui, C. Drescher et al., „Towards CRISP-ML(Q): A Machine Learning Process Model with Quality Assurance Methodology“, *Machine Learning and Knowledge Extraction*, Jg. 3, Nr. 2, 2021. DOI: 10.3390/make3020020.

[32] J. Kleiner, M. Stuckenberger, L. Komsiyska und C. Endisch, „Advanced Monitoring and Prediction of the Thermal State of Intelligent Battery Cells in Electric Vehicles by Physics-Based and Data-Driven Modeling“, *Batteries*, Jg. 7, Nr. 2, 2021. DOI: 10.3390/batteries7020031.

[33] H. Schiefer und F. Schiefer, *Statistik Für Ingenieure: Eine Einführung Mit Beispielen Aus Der Praxis*. Wiesbaden: Springer Fachmedien Wiesbaden, 2018. DOI: 10.1007/978-3-658-20640-6.

[34] F. P. Ramsey, „Truth and Probability“, *The Foundations of Mathematics and other Logical Essays*, 1926.

[35] I. N. Bronstein, K. A. Semendjaev, G. Musiol und H. Mühlig, Hrsg., *Taschenbuch der Mathematik*, 7., vollst. überarb. und erg. Aufl. Frankfurt am Main: Harri Deutsch, 2008.

[36] H. Benker, *Ingenieurmathematik Kompakt – Problemlösungen mit MATLAB*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. DOI: 10.1007/978-3-642-05453-2.

[37] I. Rot, „Methode zur modellbasierten Kalibrierung der Schaltablaufsteuerung von Getriebesteuergeräten in virtueller Umgebung“, Diss., Technische Universität Darmstadt, 2017.

[38] S. Deser, M. Gall, S. Seidl und J. Hofmann, „Effizientes Thermomanagement bei Hochvoltbatterien“, *ATZ - Automobiltechnische Zeitschrift*, Jg. 15, Nr. 04, 2020. DOI: 10.1007/s35658-020-0178-2.

[39] ADASIS. „Advancing map-enhanced driver assistance systems“. (online), Adresse: https://adasis.org/ (besucht am 24. 02. 2023).

[40] M. Lenzen, *Künstliche Intelligenz: Fakten, Chancen, Risiken*, Originalausgabe. München: C.H. Beck, 2020.

[41] DIN e.V. und DKE, *German Standardization Roadmap on Artificial Intelligence*, W. Wahlster und C. Winterhalter, Hrsg., Federal Ministry for Economic Affairs and Energy, 2020.

[42] P. M. Winter, S. Eder, J. Weissenböck et al., *White Paper - Trusted Artificial Intelligence: Towards Certification of Machine Learning Applications*, TÜV Österreich, Hrsg., 2021.

[43] H. Gouk, E. Frank, B. Pfahringer und M. J. Cree, „Regularisation of Neural Networks by Enforcing Lipschitz Continuity“, *Machine Learning*, Jg. 44, Nr. 2, 2020. DOI: 10.1007/s10994-020-05929-w.

[44] A. Zielesny, *From Curve Fitting to Machine Learning: An Illustrative Guide to Scientific Data Analysis and Computational Intelligence* (Intelligent Systems Reference Library), Second edition. Cham: Springer, 2016.

[45] F. Kirschbaum, „Modellbasierte Applikationsverfahren“, Vorlesungsskript, Stuttgart, 2017.

[46] P. Chapman, J. Clinton, R. Kerber et al., *CRISP-DM 1.0: Step-by-step data mining guide*, 2000.

[47] D. Wolpert und W. Macready, „No Free Lunch Theorems for Optimization“, *IEEE Transactions on Evolutionary Computation*, Jg. 1, Nr. 1, 1997. DOI: 10.1109/4235.585893.

[48] M. J. Kochenderfer und T. A. Wheeler, *Algorithms for Optimization*. Cambridge, Massachusetts: The MIT Press, 2019.

[49] R. Kimball und J. Caserta, *The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data*. Indianapolis, IN: Wiley, 2004.

[50] W. H. Press, *Numerical Recipes in C: The Art of Scientific Computing*, 2nd ed. Cambridge; New York: Cambridge University Press, 1992.

[51] E. Jong und M. van der Loo, „An introduction to data cleaning with R: Discussion Paper“, *Statistics Netherlands*, 2013.

[52] I. Sobol′, „Global Sensitivity Indices for Nonlinear Mathematical Models and Their Monte Carlo Estimates“, *Mathematics and Computers in Simulation*, Jg. 55, Nr. 1, 2001. DOI: 10.1016/S0378-4754(00)00270-6.

[53] D. C. Corrales, A. Ledezma und J. C. Corrales, „A case-based reasoning system for recommendation of data cleaning algorithms in classification and regression tasks“, *Applied Soft Computing*, Jg. 90, Nr. 4, 2020. DOI: 10.1016/j.asoc.2020.106180.

[54] A. Géron, *Praxiseinstieg Machine Learning mit Scikit-Learn und TensorFlow, Konzepte, Tools und Techniken für intelligente Systeme*, 1. Auflage. Heidelberg: O'Reilly, 2018.

[55] Y. Bengio, „Practical Recommendations for Gradient-Based Training of Deep Architectures“, *CoRR*, 2012. DOI: 10.48550/arXiv.1206.5533.

[56] E. Bisong, *Building Machine Learning and Deep Learning Models on Google Cloud Platform*. Berkeley, CA: Apress, 2019. DOI: 10.1007/978-1-4842-4470-8.

[57] Hadoop. „Speicherformat Apache Parquet wird Top-Level-Projekt“. (online), Adresse: https://www.heise.de/developer/meldung/Hadoop-Speicherformat-Apache-Parquet-wird-Top-Level-Projekt-2626929.html (besucht am 16. 10. 2022).

[58] Apache Spark™. „Unified Engine for Large-Scale Data Analytics“. (online), Adresse: https://spark.apache.org/ (besucht am 16. 10. 2022).

[59] A. C. Müller und S. Guido, *Einführung in Machine Learning Mit Python: Praxiswissen Data Science*, 1. Auflage. Heidelberg: O'Reilly, 2017.

[60] J. Serna, S. Diemert, L. Millet, R. Debouk, R. S und J. Joyce, „Bridging the Gap between ISO 26262 and Machine Learning: A Survey of Techniques for Developing Confidence in Machine Learning Systems“, in *SAE Technical Paper Series*, SAE International, Warrendale, PA, United States, 2020. DOI: 10.4271/2020-01-0738.

[61] C. J. Tomlin, I. Mitchell, A. M. Bayen und M. Oishi, „Computational techniques for the verification of hybrid systems“, *Proceedings of the IEEE*, Jg. 91, Nr. 7, 2003. DOI: 10.1109/JPROC.2003.814621.

[62] B. Schölkopf, R. C. Williamson, A. J. Smola, J. Shawe-Taylor und J. C. Platt, „Support Vector Method for Novelty Detection“, *NIPS'99: Proceedings of the 12th International Conference on Neural Information Processing Systems*, 1999. DOI: 10.5555/3009657.3009740.

[63] A. Kowalczyk, *Support Vector Machines Succinctly*. Syncfusion Inc., 2017.

[64] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola und R. C. Williamson, „Estimating the Support of a High-Dimensional Distribution“, *Neural Computation*, Jg. 13, 2001.

[65] C. Cortes und V. Vapnik, „Support-Vector Networks“, *Machine Learning*, Jg. 20, Nr. 3, 1995. DOI: 10.1007/BF00994018.

[66] Y. Xiao, H. Wang, L. Zhang und W. Xu, „Two Methods of Selecting Gaussian Kernel Parameters for One-Class SVM and Their Application to Fault Detection“, *Knowledge-Based Systems*, Jg. 59, 2014. DOI: 10.1016/j.knosys.2014.01.020.

[67] T. J. Santner, B. J. Williams und W. I. Notz, *Space-Filling Designs for Computer Experiments: In: The Design and Analysis of Computer Experiments*. Springer, 2003. DOI: 10.1007/978-1-4757-3799-8_5.

[68] D. M. J. Tax, „One-Class Classification“, Diss., University of Technology Delft, 2001.

[69] A. Thomas, V. Feuillard und A. Gramfort, *Calibration of One-Class SVM for MV Set Estimation*. Piscataway, NJ, USA: IEEE, 2015. DOI: 10.48550/arXiv.1508.07535.

[70] P. Merkert, „Statistik Ist Nicht Denken: Wie Sich Künstliche Intelligenz von Menschlicher Unterscheidet“, *c't*, Nr. 24, 2018.

[71] C. N. Nguyen und O. Zeigermann, *Machine Learning – Kurz & Gut: Eine Einführung Mit Python, Pandas Und Scikit-Learn* (Kurz & Gut). Heidelberg: O'Reilly, 2018.

[72] M. Schmidt-Schauß und D. Sabel, „Einführung in die Methoden der Künstlichen Intelligenz“, Vorlesungsskript, Goethe-Universität, Fachbereich Informatik und Mathematik, Frankfurt am Main, 2013.

[73] S. A. Seshia, D. Sadigh und S. S. Sastry, „Towards Verified Artificial Intelligence“, *CoRR*, 2016. DOI: 10.48550/arXiv.1606.08514.

[74] L. Igual und S. Seguí, *Introduction to Data Science*. Cham: Springer International Publishing, 2017. DOI: 10.1007/978-3-319-50017-1.

[75] I. Brahma, „Extending the Range of Data-Based Empirical Models Used for Diesel Engine Calibration by Using Physics to Transform Feature Space“, *SAE International Journal of Engines*, Jg. 12, Nr. 2, 2019. DOI: 10.4271/03-12-02-0014.

[76] K. Siebertz, D. van Bebber und T. Hochkirchen, *Statistische Versuchsplanung: Design of Experiments (DoE)* (VDI-Buch), 2. Auflage. Berlin, Heidelberg: Springer Vieweg, 2017. DOI: 10.1524/9783486843965-008.

[77] W. Ruan, X. Huang und M. Kwiatkowska, „Reachability Analysis of Deep Neural Networks with Provable Guarantees“, *CoRR*, 2018. DOI: 10.48550/arXiv.1805.02242.

[78] N. B. Erichson, O. Azencot und A. Queiruga, „Lipschitz Recurrent Neural Networks“, in *Interntational Conference on Learning Representations ICLR*, 2021.

[79] S. Hochreiter und J. Schmidhuber, „Long Short-Term Memory“, *Neural Computation*, Neural Computation, Jg. 9, 1997.

[80] D. E. Rumelhart, G. E. Hinton und R. J. Williams, „Learning Internal Representations by Error Propagation“, *ICS Report 8506*, 1985.

[81] V. Tresp, „Modelling Time Series with Neural Networks“, Vorlesungsskript, LMU München, 2017.

[82] S. Hochreiter, Y. Bengio, P. Frasconi und J. Schmidhuber, „Gradient Flow in Recurrent Nets: The Difficulty of Learning Long-Term Dependencies“, in *A Field Guide to Dynamical Recurrent Networks*, IEEE, 2009. DOI: 10.1109/9780470544037.ch14.

[83] R. H. R. Hahnloser, R. Sarpeshkar, M. A. Mahowald, R. J. Douglas und H. S. Seung, „Digital selection and analogue ampli®cation coexist in a cortex-inspired silicon circuit“, *Nature*, Jg. 405, 2000.

[84] X. Glorot, A. Bordes und Y. Bengio, „Deep Sparse Rectifier Neural Networks“, in *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS)*, Bd. Volume 15 of JMLR: W&CP 15, Fort Lauderdale, FL, USA, 2011.

[85] X. Glorot und Y. Bengio, „Understanding the difficulty of training deep feedforward neural networks“, in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, Y. W. Teh und M. Titterington, Hrsg., Ser. Proceedings of Machine Learning Research, Bd. 9, Chia Laguna Resort, Sardinia, Italy: PMLR, 2010.

[86] K. He, X. Zhang, S. Ren und J. Sun, „Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification“, in *2015 IEEE International Conference on Computer Vision (ICCV)*, Santiago, Chile: IEEE, 2015. DOI: 10.1109/ICCV.2015.123.

[87] R. H. Myers, D. C. Montgomery, G. G. Vining und T. J. Robinson, *Generalized Linear Models: With Applications in Engineering and the Sciences* (Wiley Series in Probability and Statistics). Hoboken, NJ, USA: John Wiley & Sons, Inc., 2010. DOI: 10.1002/9780470556986.

[88] Z. C. Lipton, J. Berkowitz und C. Elkan, „A Critical Review of Recurrent Neural Networks for Sequence Learning“, 2015. DOI: 10.48550/arXiv.1506.00019.

[89] K. Morik, C. Bockermann und S. Buschjäger, „Big Data Science“, *KI - Künstliche Intelligenz*, Jg. 32, Nr. 1, 2018. DOI: 10.1007/s13218-017-0522-8.

[90] D. P. Kingma und J. Ba, „Adam: A Method for Stochastic Optimization“, 2014. DOI: 10.48550/arXiv.1412.6980.

[91] T. Tieleman, G. Hinton et al., „Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude“, *COURSERA: Neural networks for machine learning*, Jg. 4, Nr. 2, 2012.

[92] F. M. Bianchi, E. Maiorino, M. C. Kampffmeyer, A. Rizzi und R. Jenssen, „An Overview and Comparative Analysis of Recurrent Neural Networks for Short Term Load Forecasting“, *CoRR*, 2017. DOI: 10.48550/arXiv.1705.04378.

[93] J. Erickson, *Algorithms*, 1st paperback edition. Selbstverlag, 2019.

[94] TensorFlow. „TensorFlow“. (online), Adresse: https://www.tensorflow.org/ (besucht am 02. 01. 2023).

[95] A. K. Shekar, C. R. de Sa, H. Ferreira und C. Soares, „Building Robust Prediction Models for Defective Sensor Data Using Artificial Neural Networks“, *CoRR*, 2018. DOI: 10.48550/arXiv.1804.05544.

[96] S. Mohseni, M. Pitale, V. Singh und Z. Wang, „Practical Solutions for Machine Learning Safety in Autonomous Vehicles“, *Association for the Advancement of Artificial Intelligence*, 2019. DOI: 10.48550/arXiv.1912.09630.

[97] C.-C. Chang und C.-J. Lin, „Training ν-Support Vector Classifiers: Theory and Algorithms“, *Neural Computation*, Jg. 13, 2001.

[98] P. Pauli, A. Koch, J. Berberich, P. Kohler und F. Allgöwer, „Training Robust Neural Networks Using Lipschitz Bounds“, 2020. DOI: 10.48550/arXiv.2005.02929.

[99] M. Fazlyab, A. Robey, H. Hassani, M. Morari und G. J. Pappas, „Efficient and Accurate Estimation of Lipschitz Constants for Deep Neural Networks“, 2019. DOI: 10.48550/arXiv.1906.04893.

[100] K. Scaman und A. Virmaux, „Lipschitz Regularity of Deep Neural Networks: Analysis and Efficient Estimation“, *32nd Conference on Neural Information Processing Systems*, 2018.

[101] C. Szegedy, W. Zaremba, I. Sutskever et al., „Intriguing properties of neural networks“, 2014. DOI: 10.48550/arXiv.1312.6199.

Eigene Veröffentlichungen

[102] F. Korthals und M. Stöcker, „Verfahren zur Nutzung künstlicher neuronaler Netze zur Berechnung von Motorbetriebspunkten im Betrieb eines Kraftfahrzeugs mit zumindest einem Antriebsmotor“, DE 10 2019 008 212 A1, 2019.

[103] F. Korthals, M. Stöcker und S. Rinderknecht, „Artificial Intelligence in predictive thermal management for passenger cars“, in *20. Internationales Stuttgarter Symposium: Automobil- und Motorentechnik*, M. Bargende, H.-C. Reuss und A. Wagner, Hrsg., Wiesbaden: Springer Fachmedien, 2020. DOI: 10.1007/978-3-658-30995-4_46.

[104] F. Korthals, M. Stöcker und S. Rinderknecht, „Plausibility Assessment and Validation of Deep Learning Algorithms in Automotive Software Development“, in *21. Internationales Stuttgarter Symposium: Automobil- und Motorentechnik*, M. Bargende, H.-C. Reuss und A. Wagner, Hrsg., Wiesbaden: Springer Fachmedien, 2021. DOI: 10.1007/978-3-658-33466-6_7.