Constrained Neural Networks for Safety-Critical Environments—In the Context of Automated Driving and Driver Assistance

Zur Erlangung des akademischen Grades eines

DOKTORS DER INGENIEURWISSENSCHAFTEN (Dr.-Ing.)

von der KIT-Fakultät für Wirtschaftswissenschaften
des Karlsruher Instituts für Technologie (KIT)

genehmigte

DISSERTATION

von

M.Sc. Mathis Brosowsky
geb. in Flensburg

Tag der mündlichen Prüfung:	14.11.2023
Hauptreferent:	Prof. Dr.-Ing. Johann Marius Zöllner
Korreferent:	Prof. Dr. techn. Gerhard Neumann

Abstract

Neural networks have been becoming a main driving factor for automated driving and driver assistance and achieve state-of-the-art performance in highly relevant learning tasks. Unfortunately, the black box character of neural networks impedes safety assurance fundamentally and slows down the approval of systems towards higher levels of automation. For safety-critical environments, constraints are an established concept to prevent unintended and hazardous behavior. However, embedding constraints in neural networks is challenging and research is in an early stage.

In this thesis, the novel neural network architecture ConstraintNet is proposed, which enforces sample-specific output constraints by construction. The constraints allow a flexible balancing between learning behavior implicitly from data and imposing explicit relationships based on prior knowledge. ConstraintNet applies an input-dependent parametrization of the constrained output space as the final layer, the so-called constraint guard layer. Furthermore, the input is extended for specifying the constraint. Thereby, a broad class of parametrizable geometries is supported as constraints and almost no computational overhead is required. For constraints in the form of convex polytopes, a compact parametrization based on the vertex representation is proposed. Constraint satisfaction, the low computational costs, and a high performance are demonstrated on facial landmark detection experiments. Furthermore, a general mathematical formalization is developed and aims to reach researchers of different domains. In the experimental part of this thesis, constrained neural networks including ConstraintNet are evaluated on two safety-relevant tasks of automated driving and driver assistance: 1) Learning a vehicle following controller to keep safe distances to vehicles ahead with deep reinforcement learning. 2) Learning a joint vehicle trajectory and cut-in prediction for safe and predictive planning and control. For the vehicle following controller, constraints on the control input, the safe sets, are derived by extending the responsibility-sensitive safety model. The safe sets ensure the avoidance of rear-end collisions formally and are imposed as hard output constraints on the policy. Finally, the effectiveness of constrained neural networks is demonstrated with regard to collision avoidance. The second task addresses safety by proposing an early, reliable, and interpretable behavior prediction for surrounding vehicles. The model's output is a bimodal distribution over trajectories and the modes are associated with a cut-in and a passing maneuver. Soft and hard output constraints are leveraged to improve the lateral separation between the cut-in and passing trajectory. In both tasks, the constraints prevent unintended behavior and the design choices of ConstraintNet result in a high performance.

In conclusion, this thesis makes important contributions to the state-of-the-art of embedding constraints in neural networks. In addition, the great potential of constrained neural networks towards safe artificial intelligence for advanced driver assistance, automated driving, and the dream of self-driving vehicles is demonstrated.

Zusammenfassung

Neuronale Netzwerke sind zu einer Schlüsseltechnologie für das automatisierte und assistierte Fahren geworden und stellen den Stand der Technik in vielen relevanten Lernaufgaben dar. Leider verändert und erschwert der Black Box Charakter neuronaler Netzwerke die Absicherung erheblich und verlangsamt die Zulassung von Systemen mit höheren Automatisierungsstufen. In sicherheitskritischen Systemen sind Einschränkungen ein etabliertes Konzept um unbeabsichtigtes und gefährliches Verhalten zu vermeiden. Allerdings ist das Integrieren von Einschränkungen in neuronale Netzwerke nicht trivial und die Forschung befindet sich noch in einem frühen Stadium.

In dieser Arbeit wird die neuartige neuronale Netzwerkarchitektur ConstraintNet vorgestellt um eingabespezifische Einschränkungen der Ausgabe per Konstruktion sicherzustellen. Dadurch können aus Daten impliziert gelerntes Verhalten und a priori bekannte explizite Zusammenhänge flexibel kombiniert werden. ConstraintNet wendet eine eingabeabhängige Parametrisierung des eingeschränkten Ausgaberaumes als finale Schicht an. Zudem ist die Eingabe erweitert um die Einschränkung zu spezifizieren. Dadurch wird eine große Klasse von parametrisierbaren Geometrien unterstützt und der zusätzliche Rechenaufwand ist minimal. Für Einschränkungen in Form von konvexen Polytopen wird eine kompakte Parametrisierung basierend auf der Vertex-Beschreibung vorgestellt. Die Erfüllung der Einschränkungen, der geringe Rechenaufwand und eine hohe Performance werden anhand von Experimenten zur Detektion von Markierungspunkten im Gesicht gezeigt. Darüber hinaus wird eine präzise mathematische Formalisierung entwickelt mit dem Ziel Anwender aus möglichst vielen Gebieten zu erreichen. Im anwendungsbezogenen Teil der Arbeit werden neuronale Netzwerke mit Einschränkungen, inklusive ConstraintNet, bezüglich zwei sicherheitsrelevanter Aufgaben für das automatisierte und assistierte Fahren evaluiert: 1) Das Lernen eines Folgereglers mithilfe von Deep Reinforcement Learning um sichere Abstände zu vorausfahrenden Fahrzeugen einzuhalten. 2) Das Lernen einer Prädiktion von Fahrzeugtrajektorien und Einscher-Manövern um sichere und prädiktive Planer und Regler zu ermöglichen. Für die Folgeregelung werden Einschränkungen der Stellgröße durch Erweiterung des Responsibility-Sensitive Safety-Modells hergeleitet. Die Einschränkungen garantieren die Vermeidung von Auffahrunfällen und werden dem Regler als harte Ausgabeeinschränkungen auferlegt. Der Nutzen von neuronalen Netzwerken mit Einschränkungen wird in Hinblick auf Kollisionsvermeidung gezeigt. Die zweite Anwendung unterstützt Sicherheit durch eine frühe, zuverlässige und interpretierbare Verhaltensprädiktion für umgebende Fahrzeuge. Die Ausgabe ist als eine bimodale Wahrscheinlichkeitsverteilung über Trajektorien modelliert und die Moden sind mit einem Einscher- und einem Vorbeifahrmanöver assoziiert. Weiche und harte Ausgabeeinschränkungen werden für eine verbesserte laterale Trennung der Einscher- und Vorbeifahrtrajektorie angewendet. In beiden betrachteten Anwendungen vermeiden die Einschränkungen unbeabsichtigtes Verhalten und ConstraintNet erreicht eine hohe Performance.

Insgesamt leistet diese Arbeit wichtige Beiträge zum Stand der Technik von neuronalen Netzwerken mit integrierten Einschränkungen. Zudem wird das große Potential von neuronalen Netzwerken mit integrierten Einschränkungen für die Sicherheit von künstlicher Intelligenz und für den Einsatz im automatisierten und assistierten Fahren aufgezeigt.

Acknowledgments

First and primarily, I sincerely thank Prof. Dr. Johann Marius Zöllner for the continuous and reliable supervision from day one, the detailed and lucid discussions, the pushing towards rethinking and improving, the valuable suggestions, and the continuous feedback. Furthermore, I enjoyed the always friendly and demanding atmosphere. This enabled me to contribute to scientific research and helped me to develop myself further. I am very grateful for this opportunity.

Moreover, special thanks go to Dr. Daniel Slieter for the excellent support, the frequent and profound discussions, steady thinking about advancement, and being always motivating with his passion regarding driver assistance and machine learning. I consider myself lucky to had him as being always available and supporting no matter how busy he was.

I thank Dr. Eric Wahl for proposing the subject area Safe AI, which is both future-oriented and challenging. In our frequent and regular discussions, he supported me in providing room for creativity, encouraging theoretical work, sharing his experience, and guiding the research towards industrial applications.

In addition, I thank the Porsche AG for the great opportunity and the environment to do this research. Especially, I thank Dr. Sebastian Söhner for the overall coordination of the Porsche Innovation Campus and the collaboration of the Porsche AG with the Forschungszentrum Informatik (FZI). Thereby, I enjoyed an inspiring working environment and state-of-the-art equipment including a cutting-edge test vehicle. I thank the full team of the Porsche Innovation Campus for detailed discussions, helping each other, and having a great time together. Throughout developing this thesis, I collaborated with talented, ambitious, and nice students. I thank Paul Orschau, Olaf Dünkel, Florian Keck, Fabrizio Puzzo, and Jakob Ketterer for excellent support. Moreover, I thank Christian Hubschneider for being a reliable contact person at FZI. Especially, I am thankful for his valuable and detailed reviewing, which helped me to improve my work. Furthermore, I thank Abhishek Vivekanandan and my colleagues from FZI for profound discussions and separate calls regarding specific research topics. Additionally, I thank Dr. Görkem Büyükyildiz for support and thinking together outside the box. Even though we worked only for a short time together, I am grateful for the support of Dr. Michael Grimm and sharing his technical expertise. I thank Dr. Benjamin Gutjahr for supporting me in finishing the thesis while working at CARIAD.

Last but not least, I am grateful to my family who supported me throughout my whole education. I appreciate the full freedom I received to follow my passion and skills. Special thanks go to my parents, my grandparents, my wonderful grandma, and my sisters Jana, Marie, and Anne.

Table of Contents

Acronyms and Mathematical Notation . . . . **xi**

1 Introduction **1**
1.1 Motivation 1
1.2 Advanced Driver Assistance and Automated Driving Systems 3
1.2.1 Levels of Driving Automation 3
1.2.2 Processing Chain of Driving Automation Systems 5
1.3 Contributions and Structure of the Thesis 6

2 Preliminaries **9**
2.1 Standards, Guidelines, and Methods for the Development of Safe Driving Automation Systems 10
2.1.1 Overview 10
2.1.2 Safety of the Intended Functionality 13
2.1.3 Formal Methods for the Safety of the Intended Functionality of Driving Policies 18
2.2 Neural Networks 22
2.2.1 Basics: Artificial Neurons, Layers, and Neural Networks 22
2.2.2 Optimization 28
2.2.3 Bayesian Deep Learning 33
2.3 Reinforcement Learning 35
2.3.1 Basics and Intuition 35
2.3.2 Markov Decision Process 37
2.3.3 General Concepts for Solving Markov Decision Processes 39
2.3.4 Reinforcement Learning Algorithms 44

3 Sample-Specific Output Constraints for Neural Networks: ConstraintNet **49**
3.1 Motivation 50
3.2 Related Work 54
3.3 ConstraintNet 60
3.3.1 Sample-Specific Output Constraints for Neural Networks 60
3.3.2 Architecture and Construction 61
3.3.3 Constraint Guard Layer 63
3.3.4 Training 65
3.3.5 Supported Constraints and Generalizations 66

3.4 Facial Landmark Detection Experiments 68
3.4.1 Overview 68
3.4.2 Output Constraints 69
3.4.3 Training and Quantitative Results 72
3.4.4 Qualitative Results 75
3.5 Conclusion 79

4 Safe Reinforcement Learning with Constrained Neural Networks: Vehicle Following Controller 81
4.1 Motivation 82
4.2 Related Work 85
4.2.1 Safe Reinforcement Learning 85
4.2.2 Adaptive Cruise Control 87
4.3 Methods 88
4.3.1 Vehicle Following Controller 88
4.3.2 State-Specific Safe Sets 90
4.3.3 Twin Delayed Deep Deterministic Policy Gradient Algorithm . . 98
4.4 Experiments 100
4.4.1 Simulator and Reward Function 100
4.4.2 Training 103
4.4.3 Results 105
4.5 Conclusion 108

5 Behavior Prediction for Safe Driving with Constrained Neural Networks: Joint Vehicle Trajectory and Cut-In Prediction 109
5.1 Motivation 110
5.2 Related Work 113
5.3 Methods 116
5.3.1 Data Set and Automatic Scenario Detection 116
5.3.2 Model Input 123
5.3.3 Encoder-Decoder Architecture 125
5.3.4 Soft and Hard Output Constraints 128
5.3.5 Loss 129
5.4 Experiments 131
5.4.1 Evaluation Metrics 131
5.4.2 Preprocessing 134
5.4.3 Training 138
5.4.4 Results 139
5.5 Conclusion 148

6 Conclusion and Outlook 151

List of Figures 155

List of Tables **157**

Publications, Patents, and Supervised Theses **159**

References **161**

Acronyms and Mathematical Notation

Acronyms

1D	One Dimensional
2D	Two Dimensional
3D	Three Dimensional
ACC	Adaptive Cruise Control
ACC 4S	Adaptive Cruise Control with State-Specific Safe Sets
AD	Automated Driving
Adam	Adaptive moment estimation
ADAS	Advanced Driver Assistance System
ADS	Automated Driving System
AEB	Automatic Emergency Braking
AI	Artificial Intelligence
ANN	Artificial Neural Network
ASIL	Automotive Safety Integrity Level
AV	Automated Vehicle
BDL	Bayesian Deep Learning
BRNN	Bidirectional Recurrent Neural Network
CIPV	Closest In-Path Vehicle
CLM	Constrained Local Model
CMDP	Constrained Markov Decision Process
CNN	Convolutional Neural Network
CV	Computer Vision
DAG	Directed Acyclic Graph
DAS	Driving Automation System
DDPG	Deep Deterministic Policy Gradient
DL	Deep Learning
DNN	Deep Neural Network
DQN	Deep Q Network
DRL	Deep Reinforcement Learning
E/E	Electrical and/or Electronical
FLMDE	Final Lateral Mean Displacement Error
FMDE	Final Mean Displacement Error
FuSa	Functional Safety

GCN	Graph Convolutional Network
GNN	Graph Neural Network
GRU	Gated Recurrent Unit
HIL	Hardware In the Loop
ID	Identifier
IPM	Interior Point Method
KKT	Karush-Kuhn-Tucker
LiDAR	Light Detection And Ranging
LMDE	Lateral Mean Displacement Error
LSTM	Long Short-Term Memory
MAP	Maximum A Posteriori
MDE	Mean Displacement Error
MDN	Mixture Density Network
MDP	Markov Decision Process
M-FLMDE	Final Lateral Mean Displacement between Modes
MIL	Model In the Loop
ML	Machine Learning
MLE	Maximum Likelihood Estimation
M-LMDE	Lateral Mean Displacement between Modes
MPC	Model Predictive Control
MSE	Mean Squared Error
NLL	Negative Log-Likelihood
NLP	Natural Language Processing
NN	Neural Network
ODD	Operational Design Domain
QM	Quality Management
QP	Quadratic Program
RADAR	RAdio Detection And Ranging
ReLU	Rectified Linear Unit
RL	Reinforcement Learning
RMSprop	Root Mean Squared propagation
RNN	Recurrent Neural Network
ROS	Robot Operating System
RSS	Responsibility-Sensitive Safety
SAE	Society of Automotive Engineers
SFF	Safety Force Field
SGD	Stochastic Gradient Descent
SIL	Software In the Loop
SOTIF	Safety of the Intended Functionality
TD3	Twin Delayed Deep Deterministic Policy Gradient
TPE	Tree-structured Parzen Estimator

UNECE	United Nations Economic Commission for Europe
VLSS	Vehicle Level Safety Strategy
V&V	Verification and Validation
WTA	Winner-Takes-All

Mathematical Notation

Lower Case Letters

a	Acceleration
b	Bias term of a model, a vector in general
c	Index for entries of vectors/matrices/tensors, acceleration bound for vehicle following controller
d	Distance
$f_\theta(\cdot)$, $n_\theta(\cdot)$	Model with parameters θ
$g(\cdot)$	Activation function of a neuron, tensor representation of a constraint in ConstraintNet
h	Hidden state of a model, kernel of a convolution, index for time steps
i	Index for training steps, index for samples in a data set
j	Jerk (time derivative of the acceleration)
k	Index for time steps, index for entries of vectors/matrices/tensors, variable for number of incidents
l	Index for time steps, index for entries of vectors/matrices/tensors
$\ell(\cdot)$	Loss function
o	Index for entries of vectors/matrices/tensors
$p(\cdot)$	Probability density function
$r(\cdot)$	Reward function
s	Length of a route, constraint parameter of ConstraintNet, scenario metadata
$\mathrm{sig}(\cdot)$	Sigmoid function $\mathrm{sig}(t) = 1/(1 + \exp(-t))$
t	Time point
u	Control input, action
v	Velocity, vertex of a polytope
w	Weight of a model, weighting factor
x	Input of a model

y	Output of a model, label
$\hat{y}$	Prediction of a model
z	Intermediate or latent variable

Capital Letters

A	Matrix $A \in \mathbb{R}^{K \times N}$
C	Constrained set, set of safe actions
$\mathfrak{C}$	Class of output constraints
$\mathcal{D}$	Data set
$E[\cdot]$	Expectation value
$G_k(\cdot)$	Random variable for the return at time step k
H, K, L	Number of elements in a sequence
I	Identity matrix
$J(\cdot)$	Cost function, expected discounted return
$\mathcal{K}$	Set of time step indices
$L(\cdot)$	Loss
$\mathbb{N}$	Set of natural numbers
$\mathbb{N}_0$	Set of natural numbers including zero
N	Number of samples in a data set
$\mathcal{N}(\cdot, \Sigma)$	Gaussian distribution with covariance matrix Σ
$\mathcal{O}$	Sector of a circle
$\mathcal{P}$	Convex polytope
$P(\cdot)$	Probability function
$Q^{\pi}(x, u)$	Action-value function for policy π, state x, and action u
R	Radius
$R(\cdot)$	Regularization term, random variable for a reward
$\mathbb{R}$	Set of real numbers
$\mathcal{S}$	Set of constraint parameters
T	Time interval
$\mathcal{T}$	Set of time points
$\mathcal{U}$	Action space

$U(\cdot)$	Random variable for an action
$V^{\pi}(x)$	State-value function for policy π and state x
W	Weight matrix
$\mathcal{X}$	Domain of a function, state space
$X(\cdot)$	Random variable for a state
$\mathcal{Y}$	Range of a function
$\mathcal{Z}$	Range of an intermediate variable z

Greek Letters

α	Tuning parameter for leaky ReLU, coefficients of a mixture model
β	Factor for exponential decay
γ	Discount factor, confidence level for statistical hypothesis tests
ϵ	Small positive real number
η	Learning rate
θ	Parameters of a model
Θ	Set of parameters
λ	Weighting factor, parameter of a probability distribution
μ	Mean value
π	Pi
$\pi(\cdot)$	Policy
ρ	Correlation coefficient of a 2D Gaussian distribution
σ	Standard deviation
$\sigma(\cdot)$	Softmax function
Σ	Covariance matrix
τ	Time constant of a first-order lag, roll-out in reinforcement learning
$\phi(z, s)$	Constraint guard layer of ConstraintNet applied on the intermediate variable z and the constraint parameter s
φ	Angle

Operators

∇	Gradient
$\frac{\partial}{\partial x}$	Partial derivative *w.r.t.* x
$f \otimes g$	Cross-correlation of function f and g
$f * g$	Convolution of function f and g
$\mathcal{S}^{(1)} \times \mathcal{S}^{(2)}$	Cartesian product of set $\mathcal{S}^{(1)}$ and $\mathcal{S}^{(2)}$
$\odot$	Element-wise multiplication
A^{-1}	Inverse matrix $A^{-1}A = AA^{-1} = I$
$v^\intercal, A^\intercal$	Transpose of a vector and transpose of a matrix

1 Introduction

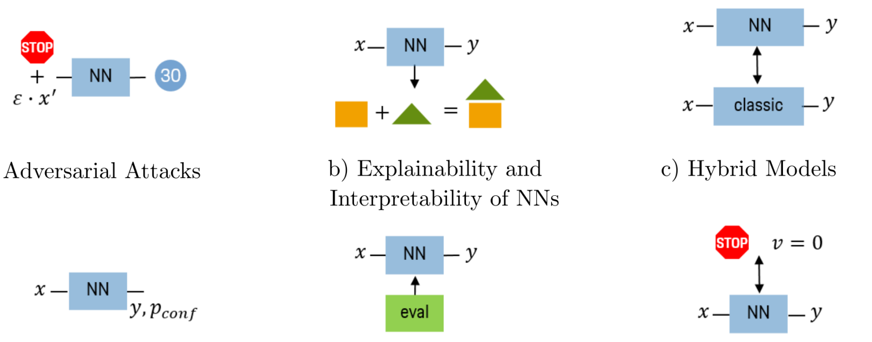
1.1 Motivation

In the last decade, Automated Driving Systems (ADSs) and Advanced Driver Assistance Systems (ADASs) have gained large attention beyond the automotive industry in research, technology and software companies, the start-up sector, the judiciary, the society, and politics. The attention on ADSs and ADASs—in the following referred to collectively as Driving Automation Systems (DASs)—results from manifold and interconnected driving factors.

From a customer and an economic perspective, new mobility concepts are expected and fundamental transformations of the transport sector are conceivable [Stegmüller et al., 2019]. In private mobility, many drivers already benefit from ADASs when facing uncomfortable or time-consuming driving situations like long journeys, reversing into a parking space, or traffic jams. However, until today car manufacturers have extended mainly the functionality of ADASs, which always require the human driver as a fallback. In comparison, in vehicles with activated ADSs the driver is allowed to do other activities such as reading E-mails or even may be discharged from the driving task at all. In public transport, highly automated shuttles could close the last mile between home and tram [Barthelmes et al., 2022]. In freight transport, truck companies work on solutions for Automated Driving (AD) between logistic centers—the so-called hub-to-hub transport [Gernant et al., 2022].

From a technological perspective, synergies between AD and Artificial Intelligence (AI) have been becoming a major driving factor. In recent years, Deep Learning (DL) achieved major progress in different fields like Computer Vision (CV) [He et al., 2016] and Natural Language Processing (NLP) [Vaswani et al., 2017]. On the one hand, these findings contribute significantly to the state-of-the-art of AD [Grigorescu et al., 2020], *e.g.* in perception, situation interpretation, and behavior prediction. On the other hand, safety concerns of AI and DL slow down the development of systems with higher level of automation and extended operational domains. *E.g.* Neural Networks (NNs) are susceptible to small input perturbations, out-of-distribution data, and domain-shifts [Stage et al., 2022; Klingner and Fingscheidt, 2022]. These challenges are of fundamental importance as safety is an indispensable prerequisite for the acceptance of AD.

Therefore, striving for improved road safety and reducing traffic accidents compared to human driven vehicles is both a fundamental challenge and chance of AD. In Germany alone, in 2021 2562 people died because of traffic accidents [Destatis, 2022, p.54] and 88.0% of the accidents with personal damage were caused by human failure [Destatis, 2022, p.49]. Already today, ADASs like Automatic Emergency Braking (AEB) actively



a) Adversarial Attacks
b) Explainability and Interpretability of NNs
c) Hybrid Models
d) Bayesian Deep Learning
e) Verification of NNs
f) NNs and Prior Knowledge

Figure 1.1: Overview about research fields addressing the black box character of NNs.

prevent accidents. However, it remains a large potential when ADSs overcome weaknesses of human drivers like susceptibility to tiredness, inattentiveness, distraction, or the limited field of view.

Improving road safety with DASs requires systematic approaches for safety assurance. Addressing this, international development standards, *e.g.* the *Safety of the Intended Functionality (SOTIF)* [ISO21448, 2022], and regulatory frameworks [BGBl, 2017, 2021] have been extended. Furthermore, white papers [SaFAD, 2019; Nistér et al., 2019], publications [Shalev-Shwartz et al., 2017], and research projects [PEGASUS, 2019; KI-Absicherung, 2022] contribute to the state-of-the-art of safety assurance. A key challenge stems from the use of AI and in particular NNs. The behavior of NNs is learned implicitly from data and the decision-making process remains opaque [Rudin, 2019; Gilpin et al., 2018]. This so-called black box character has fundamental implications for the safety assurance. In comparison to classical development processes, the implementation and testing according to detailed specifications must be rethought. Furthermore, identified functional insufficiencies are not explainable by default and retraining the NN may cause other unintended behavior.

For assuring safe intended behavior of NNs, two classes of approaches can be distinguished. The first class of approaches proposes to mitigate the risk by an appropriate model selection and training process [ISO21448, 2022, p.172-175], [SaFAD, 2019, p.116-130]. This includes carefully selecting the model, preparing the data set, performing the actual training, and validating and verifying the trained models. Functional insufficiencies are addressed by iteratively updating the data set, retraining, and changing the model if required. The second class of approaches directly strives to reduce the black box character of NNs. An overview about related research fields is visualized in Figure 1.1 and the following categories can be identified: a) NNs are susceptible to small input manipulations [Szegedy et al., 2013] and defense algorithms against adversarial attacks are an active research field, *e.g.* adversarial training methods [Madry, 2018]. b) The decision-making of conventional NNs is not transparent. Thus, approaches addressing the interpretability of NNs strive for representations and relations that are understandable by humans, *e.g.* by generating saliency maps [Zeiler and Fergus, 2014; Simonyan et al., 2014]. According to Gilpin et al. [2018], explainable AI includes interpretable AI but requires additionally a sufficient detailed description of the NN behavior. c) Hybrid models combine NNs with classical rule-based models, *e.g*. constrained local models with NNs as detectors [Zadeh et al., 2017]. d) For assessing the reliability of NNs' predictions, well-calibrated estimations of uncertainty are important. Bayesian Deep Learning (BDL) focuses on approaches to capture *aleatoric* and *epistemic* uncertainty in NNs [Gal and Ghahramani, 2016]. e) Verification methods aim to prove the validity of formal properties in NNs [Huang et al., 2017; Katz et al., 2017]. f) Other approaches incorporate prior knowledge in NNs to combine the data-driven training with known explicit relationships. This can be achieved by imposing constraints on NNs [Karpatne et al., 2017; Pham et al., 2018].

In safety-critical tasks of DASs, constraints are an important concept to exclude hazardous behavior. *E.g*. for the safety of driving policies, the Responsibility-Sensitive Safety (RSS) model [Shalev-Shwartz et al., 2017] and the Safety Force Field (SFF) [Nistér et al., 2019] provide control constraints with formal collision avoidance guarantees. However, imposing constraints on NNs is challenging and existing approaches are limited in different aspects, *e.g*. relaxed guarantees [Karpatne et al., 2017; Márquez-Neila et al., 2017] or computational costs [Amos and Kolter, 2017; Agrawal et al., 2019]. Addressing this, the goal of the thesis is twofold. First, an efficient methodology to embed hard output constraints in NNs has to be proposed. Second, the performance and safety of conventional and constrained NNs including NNs according to the proposed methodology have to be evaluated on two safety-relevant learning tasks of DASs: 1) Learning a vehicle following controller to keep safe distances to vehicles ahead with Deep Reinforcement Learning (DRL). 2) Learning a joint vehicle trajectory and cut-in prediction for safe and predictive planning and control of Automated Vehicles (AVs).

The remainder of the introduction is structured as follows. In Section 1.2, the six SAE-Levels [SAE-J3016, 2021] and the processing chain of DASs are explained. In Section 1.3, the contributions are stated and the structure of the thesis is outlined.

1.2 Advanced Driver Assistance and Automated Driving Systems

1.2.1 Levels of Driving Automation

DASs support or automate the driving task on a sustained basis. According to the Society of Automotive Engineers (SAE) [SAE-J3016, 2021], DASs are classified in six levels depending on the automation of longitudinal and/or lateral control, the responsibility of the driver, and the scope of the Operational Design Domain (ODD). The ODD describes the well-defined environmental conditions for which the DAS is designed to operate, *e.g*. certain road types and weather conditions. The taxonomy of automation levels ranges from SAE-Level 0 (no driving automation) to SAE-Level 5 (full driving automation) and is visualized in Figure 1.2. SAE-Level 0 includes also systems that do not control on a sustained basis, *e.g*. automatic emergency braking. SAE-Level 1 (driver assistance) and SAE-Level 2 (partial driving automation) support the driver in longitudinal and/or

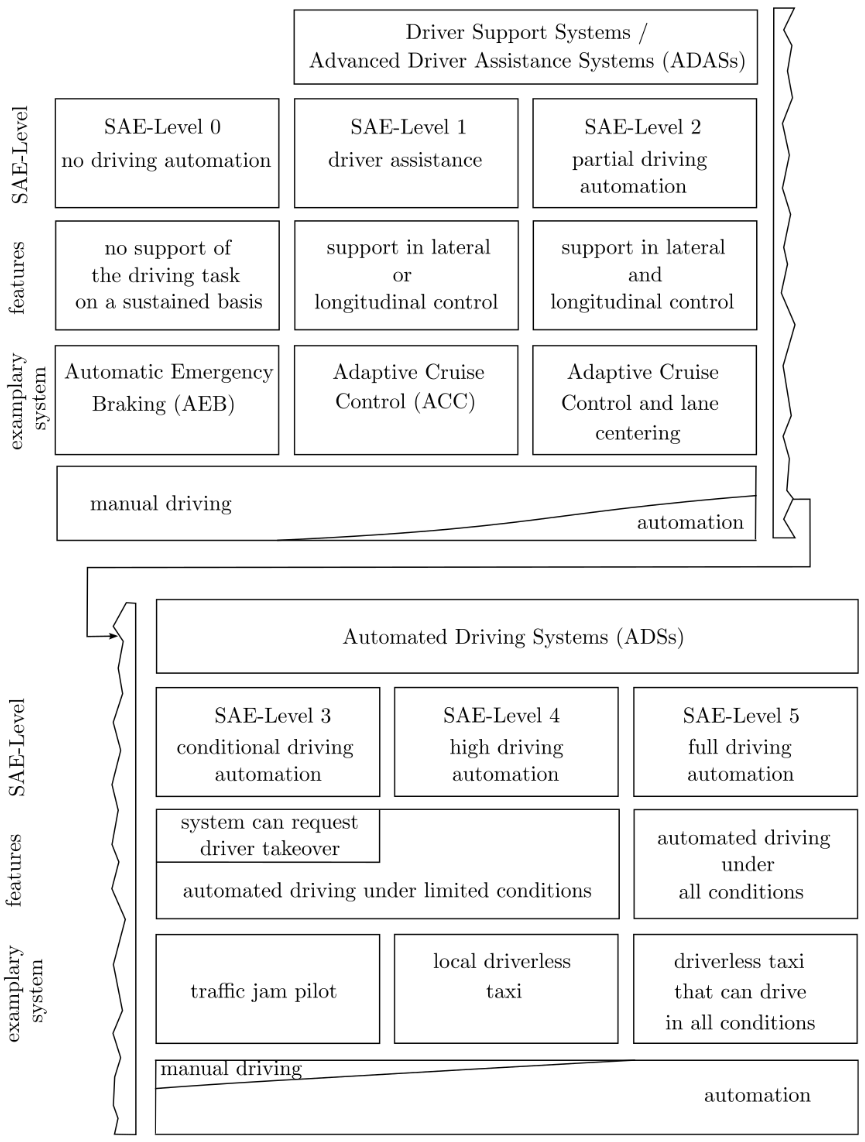

Driver Support Systems / Advanced Driver Assistance Systems (ADASs)
SAE-Level
SAE-Level 0
no driving automation
SAE-Level 1
driver assistance
SAE-Level 2
partial driving automation
features
no support of the driving task on a sustained basis
support in lateral or longitudinal control
support in lateral and longitudinal control
examplary system
Automatic Emergency Braking (AEB)
Adaptive Cruise Control (ACC)
Adaptive Cruise Control and lane centering
manual driving
automation
Automated Driving Systems (ADSs)
SAE-Level
SAE-Level 3
conditional driving automation
SAE-Level 4
high driving automation
SAE-Level 5
full driving automation
features
system can request driver takeover
automated driving under limited conditions
automated driving under all conditions
examplary system
traffic jam pilot
local driverless taxi
driverless taxi that can drive in all conditions
manual driving
automation

Figure 1.2: The six levels of driving automation according to SAE-J3016 [2021].

lateral control while the driver is still in charge to monitor the environment permanently. Furthermore, the driver must be ready to take over control at any time. SAE-Level 1 refers to systems that assist the driver either in longitudinal *or* lateral control. *E.g*. Adaptive Cruise Control (ACC) controls the velocity to a set value and keeps a velocity-dependent distance to a potential vehicle in front. In comparison, SAE-Level 2 systems support in longitudinal *and* lateral control, *e.g*. lane centering in combination with ACC. For DASs with SAE-Level 3 (conditional driving automation) or higher, the driver is no longer in charge to supervise the driving task permanently if the system is engaged. The driver is allowed do other activities. However, systems of SAE-Level 3 may request the driver to take over control within a certain amount of time. On German roads, Mercedes-Benz AG

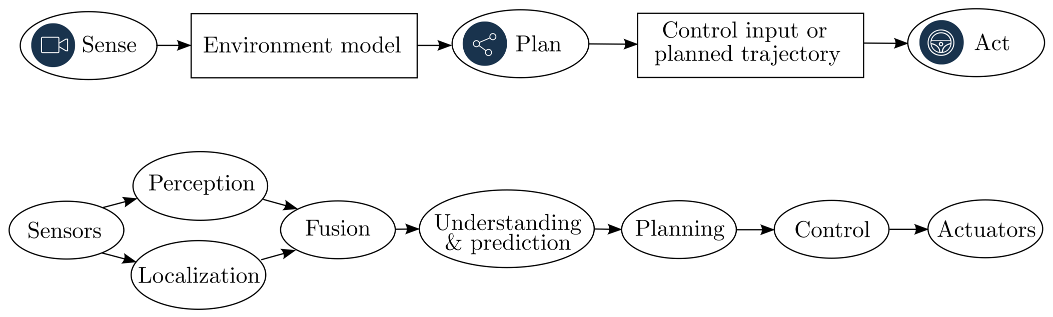


Figure 1.3: Sense-plan-act model and processing chain of DASs.

has approved the first SAE-Level 3 system with the DRIVE PILOT[1]. The DRIVE PILOT operates in a strongly restricted ODD on highways up to 60 km h^{-1} and the driver must be ready to take over control within 10 s. In comparison to SAE-Level 2 systems, car manufacturers accept liability for collision caused by SAE-Level 3 systems. Thus, instead of offering DASs with SAE-Level 3 many car manufacturer extend the functionality and ODD of SAE-Level 2 systems, which are frequently called Level 2+ systems. DASs with SAE-Level 4 (high driving automation) handle the driving task in certain ODDs without any driver takeover. Finally, SAE-Level 5 (full driving automation) extends SAE-Level 4 by automatization of the driving task to unlimited conditions. In the remainder of the thesis, the term ADASs is used for DASs up to SAE-Level 2 and the term ADSs refers to DASs with SAE-Level 3 or higher. In *Gesetz zum autonomen Fahren* [BGBl, 2021], an autonomous vehicle is defined as a vehicle with a certain technical equipment that is capable of handling the driving task in a certain ODD without the necessity of a human driver (corresponds to SAE-Level 4). However, in this thesis the term *autonomous driving* and *autonomous vehicle* is avoided due to ambiguous definitions and discussions.

1.2.2 Processing Chain of Driving Automation Systems

DASs scan the environment with sensors, process the recorded data, compute action commands like steering and braking, and finally execute them. There are approaches that propose to apply NNs for this whole processing chain at once [Bojarski et al., 2016]. These so-called end-to-end approaches compute the control input directly from the sensor raw data. However, the trust in end-to-end approaches suffers from missing interpretability and typically modular approaches are used instead. The *sense-plan-act model* [ISO21448, 2022, p.15] abstracts three modules on a coarse level. The module *sense* processes sensor data and generates an environment model. Next, the module *plan* computes the ego vehicle's future trajectory or directly the control input from the environment model. Finally, the module *act* executes the planned behavior. The sense-plan-act model can be decomposed in further submodules as shown in Figure 1.3 [Amersbach and Winner, 2017]. For an accurate model of the environment, typically several sensors are installed and perception algorithms extract semantic information like object detections from the

[1] DRIVE PILOT of Mercedes-Benz `https://group.mercedes-benz.com/innovation/case/autonomous/drive-pilot.html`, accessed on 10/10/2022

raw data. Different sensor technologies may be combined to complement each other or for redundancy. Furthermore, a localization module tracks the ego vehicle's position and motion and allows to enrich the environment model with map data. The sensor fusion aggregates the information of the perception and localization and creates one holistic model of the environment. The knowledge about the traffic situation is further extended with the situation understanding and prediction module. This module extracts high level information like expected maneuvers of surrounding vehicles. Given the holistic environment model and the situational awareness, the planning module computes an optimal trajectory. Finally, the control module computes commands regarding longitudinal and lateral motion and the actuators execute them.

1.3 Contributions and Structure of the Thesis

In safety-critical tasks of DASs, modeling constraints and solving an objective *w.r.t.* these constraints is a common approach to implement safety requirements. *E.g.* in trajectory planning algorithms based on Model Predictive Control (MPC), constraints are leveraged to ensure collision avoidance [Gutjahr, 2019, p.18]. However, research on constrained NNs is in an early stage and existing approaches are limited in different aspects, *e.g.* relaxed guarantees [Karpatne et al., 2017; Márquez-Neila et al., 2017] or computational costs [Amos and Kolter, 2017; Agrawal et al., 2019]. Adressing this, the contributions of the thesis are as follows.

First, the NN architecture ConstraintNet [Brosowsky et al., 2021a] is proposed, which is capable of ensuring hard output constraints. ConstraintNet applies an input-dependent parametrization of the constrained output space and requires almost no computational overhead. This allows a flexible balancing between learning behavior implicitly from data and imposing explicit relationships based on prior knowledge. In addition, conventional and constrained NNs including ConstraintNet are evaluated on two tasks of DASs. The first task is a controller for keeping safe distances to vehicles ahead. The control behavior is learned with DRL. For avoiding rear-end collisions, the RSS model [Shalev-Shwartz et al., 2017] is extended and the modeled output constraints are imposed on the policy [Brosowsky et al., 2021b]. The second task is a prediction of surrounding vehicles' behavior based on the ego vehicle's perception. A vehicle-wise bimodal trajectory prediction with an interpretable passing and cut-in mode is proposed [Brosowsky et al., 2021c]. The prediction is based on Long Short-Term Memorys (LSTMs) and intended for real-world application on highways. Output constraints are modeled and analyzed for improving the lateral separation of the cut-in and passing trajectory. An early and reliable behavior prediction is crucial for safe and predictive planning and control of AVs.

To summarize, this thesis comprises a methodological and an experimental part. In the methodological part, ConstraintNet—a novel NN architecture with embedded output constraints—is proposed and different types of constraints are modeled. In the experimental part, conventional and constrained NNs are evaluated on two safety-relevant tasks of DASs. This results in a structure of the thesis into the following three main chapters.

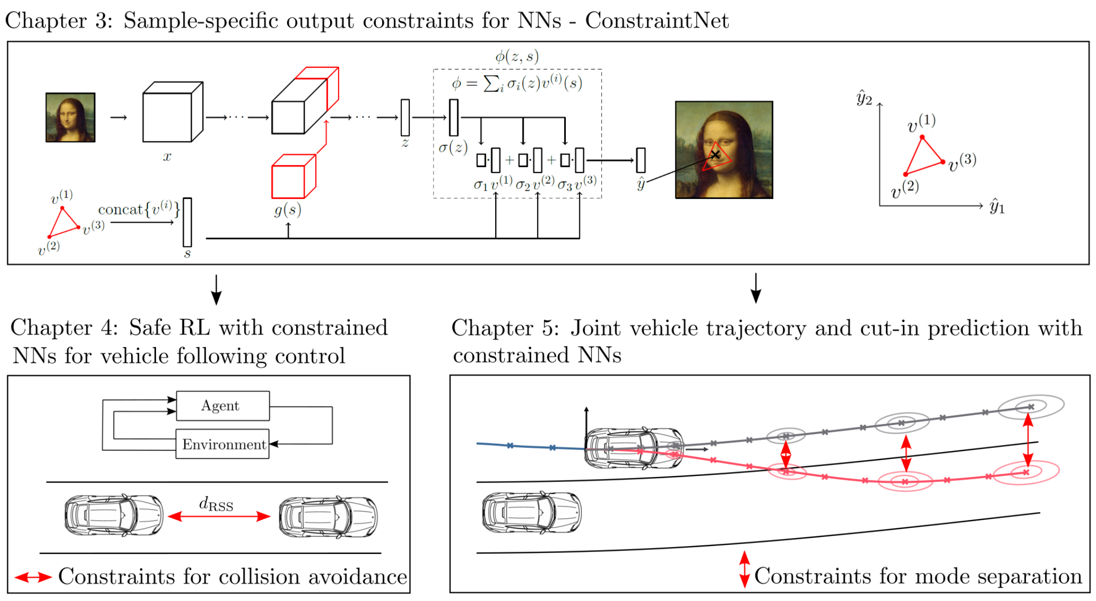

Chapter 3: Sample-specific output constraints for NNs - ConstraintNet
$\phi(z,s)$
$\phi = \sum_i \sigma_i(z) v^{(i)}(s)$
x
z
$\sigma(z)$
$g(s)$
$\sigma_1 v^{(1)}$ $\sigma_2 v^{(2)}$ $\sigma_3 v^{(3)}$
$\hat{y}$
concat$\{v^{(i)}\}$
s
$v^{(1)}$
$v^{(2)}$
$v^{(3)}$
$\hat{y}_1$
$\hat{y}_2$
Chapter 4: Safe RL with constrained NNs for vehicle following control
Agent
Environment
d_{RSS}
Constraints for collision avoidance
Chapter 5: Joint vehicle trajectory and cut-in prediction with constrained NNs
Constraints for mode separation

Figure 1.4: Overview of the chapters in this thesis.

- Sample-Specific Output Constraints for Neural Networks: ConstraintNet [Brosowsky et al., 2021a] (Chapter 3)
- Safe Reinforcement Learning with Constrained Neural Networks: Vehicle Following Controller [Brosowsky et al., 2021b] (Chapter 4)
- Behavior Prediction for Safe Driving with Constrained Neural Networks: Joint Vehicle Trajectory and Cut-In Prediction [Brosowsky et al., 2021c] (Chapter 5)

The structure of the thesis is visualized in Figure 1.4. In the following, the contributions of each chapter are stated explicitly.

Sample-Specific Output Constraints for Neural Networks: ConstraintNet

- The novel NN architecture ConstraintNet is proposed, which enforces hard output constraints by construction. The capability of imposing sample-specific output constraints allows to incorporate prior knowledge in a flexible way. The more prior knowledge exists, the more specific the constraints can be modeled and *vice versa*.
- For imposing the output constraints, an input-dependent parametrization of the constrained output space is leveraged. Thereby, the complete interior of the constrained output space is covered and almost no additional computational costs are required.
- Multiple constraints for the same input are applicable. The output constraint is set in each forward pass independently by specifying a tensor description of the constraint.
- ConstraintNet supports a broad class of constraints. For constraints in form of convex polytopes, a compact parametrization based on the vertex representation is proposed. Furthermore, a general mathematical formalization for output constraints of arbitrary

parametrizable geometries including non-convex polytopes and unbounded regions is provided and encourages the modeling of problem-specific constraints.

Safe Reinforcement Learning with Constrained Neural Networks: Vehicle Following Controller

- State-specific sets of safe actions, the safe sets, are derived by building on the RSS model. Collision avoidance is formally ensured by imposing the state-specific safe set as hard output constraints on the vehicle following policy.
- The Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm [Fujimoto et al., 2018] is leveraged to learn safe, accurate, and comfortable vehicle following policies. For the policies, an unconstrained NN and three NNs with imposed safe sets are applied. For the latter, ConstraintNet is leveraged and compared to two clipping approaches: clipping as post-processing and with a projection layer.
- The effectiveness of the safe sets is validated by measuring a constant crash rate of zero for all constrained policies.
- The training behavior and the performance of the unconstrained and constrained policies are evaluated. Compared to the other approaches, ConstraintNet has the most stable and fastest training behavior. Furthermore, the performances of the evaluated policies are similar with slight advantages for ConstraintNet and the approach with clipping as post-processing.

Behavior Prediction for Safe Driving with Constrained Neural Networks: Joint Vehicle Trajectory and Cut-In Prediction

- For a probabilistic and interpretable behavior prediction, LSTM-based encoder-decoder models are leveraged and the output is modeled as a bimodal distribution over trajectories. The two modes of the distribution are semantically assigned to a cut-in and a passing maneuver.
- Soft and hard output constraints are modeled for improving the lateral separation of the cut-in and passing mode. The soft constraints are implemented with an additional loss term and the hard constraints are imposed by applying ConstraintNet.
- Automatic labeling functions and a retrospective view are leveraged and a data set with 1856 cut-in and an equal number of passing maneuvers is created. The models depend only on the ego vehicle's sensor measurements and are intended for real-world application on highways.
- All LSTM-based models predict the cut-ins far before the physical baseline and the lateral trajectory error is reduced by a factor of two over a constant velocity model. It is shown that ConstraintNet achieves the best overall performance *w.r.t.* cut-in prediction, trajectory accuracy, and mode separation. Furthermore, the lateral and longitudinal uncertainties of the trajectory waypoints are well-interpretable and increase with larger time horizons as expected.

2 Preliminaries

Contents

2.1 Standards, Guidelines, and Methods for the Development of Safe Driving Automation Systems . . . 10

2.1.1 Overview . . . 10

2.1.2 Safety of the Intended Functionality . . . 13

2.1.3 Formal Methods for the Safety of the Intended Functionality of Driving Policies . . . 18

2.2 Neural Networks . . . 22

2.2.1 Basics: Artificial Neurons, Layers, and Neural Networks . . . 22

2.2.2 Optimization . . . 28

2.2.3 Bayesian Deep Learning . . . 33

2.3 Reinforcement Learning . . . 35

2.3.1 Basics and Intuition . . . 35

2.3.2 Markov Decision Process . . . 37

2.3.3 General Concepts for Solving Markov Decision Processes . . . 39

2.3.4 Reinforcement Learning Algorithms . . . 44

Improving road safety is a main driving factor and a big challenge of DASs. Chapter 2.1 covers relevant standards, guidelines, and methods that have been developed to address the safety of DASs. First, an overview is provided. Then, important concepts of the SOTIF standard [ISO21448, 2022] are explained. The standard includes a chapter about considerations for Machine Learning (ML) and for the design of safe driving policies [ISO21448, 2022, p.159-175]. In Chapter 2.2, the basic concepts of NNs are explained. This includes the training of NNs in a supervised manner. For modeling aleatoric and epistemic uncertainty, a probabilistic interpretation is derived from Bayes' Theorem. Chapter 2.3 provides an introduction to Reinforcement Learning (RL). Contrary to supervised learning, in RL agents learn purely from experience and interacting with the environment. Mathematically, the problem can be formalized as a Markov Decision Process (MDP). Finally, general concepts and specific classes of RL algorithms are presented to solve MDPs approximately.

2.1 Standards, Guidelines, and Methods for the Development of Safe Driving Automation Systems

2.1.1 Overview

A major motivation of DASs is the goal to improve safety and to reduce the number of accidents caused by human failure. Already today, assuring safety of DASs requires an enormous development and approval effort. With increasing level of automation, safety assurance becomes even more challenging. For the safety of DASs, a development process and approval according to standards, regulations, and guidelines is required. These frameworks are under continuous development and represent the state-of-the-art in industry, law, and research. The following of this section serves as a general overview, which highlights different aspects, dimensions, and perspectives of safety for DASs. First, a general definition of safety is provided. Next, the status quo and the need of harmonised regulations and standards for AD are explained. There exist essential differences between safety assurance of ADASs and ADSs, which are highlighted. Furthermore, the common categorization of safety and security in functional safety, SOTIF, and cybersecurity is presented. Finally, existing guidelines towards trustworthy AI are explained. The next Section 2.1.2 outlines important measures and methods for the SOTIF more specifically as the crucial safety domain of AD. Section 2.1.3, in turn, is more specific about two formal methods of the SOTIF for safe planning and control and summarizes the RSS [Shalev-Shwartz et al., 2017] model and the SFF [Nistér et al., 2019]. These models can be interpreted as a safety layer for ADSs.

Safety as an Acceptable Level of Risk

A common definition of safety is the reduction of risk to an acceptable level [Wachenfeld and Winner, 2015, p.440 ff.]. Both terms, *risk* and *acceptable level*, require further explanation. First, *risk* increases with the probability of damage and its severity. For ADASs, the probability of damage, in turn, can be estimated by evaluating the exposure of the corresponding hazardous behavior in combination with its controllability. In Wachenfeld and Winner [2015, p.441], the following classes are used to categorize the accident *severity* with increasing order of harm: accidents with material damage, with slightly injured people, with severely injured people, and with fatalities. An exact metric to combine the probability and the severity of an accident to a number that represents the risk is controversially discussed and is not part of this work. To approve DASs, an *acceptable level* of risk must be ensured. For AD without a human driver as supervisor, this requires a difficult balancing of diverse perspectives including expert assessments, societal acceptance, ethical questions, additional benefits of the technology, and legal foundations. One common opinion is that the reduced risk due to automation needs to be greater than the additional risk caused by it. Frequently, traffic and accident statistics provide the relevant data of the status quo. Currently, the predominant proportion of traffic accidents is caused by humans. In 2021, 88.0% of all accidents on German public roads with personal damage were caused by misbehavior of drivers [Destatis, 2022, p.49]. Thus, reducing the number and severity of accidents is one of the main driving forces for DASs. Driven by this goal, the General Safety Regulation [EU2019/2144, 2019] defines mandatory safety features and requirements. *E.g.* in the European Union, an intelligent speed assist is mandatory for vehicle type approval since July 2022.

Demand of Harmonised Regulations and Standards for Automated Driving

Nowadays, many vehicles are equipped with ADASs and regulations (*e.g.* UN-R79 [2018]) and standards (*e.g.* ISO15622 [2018]) guide the development and approval. In comparison, ADSs are still rarely in use. In December 2021, the DRIVE PILOT of Mercedes-Benz AG has been approved in Germany as the first SAE-Level 3 system[1] and the system has been available for customers since 2022. On the one hand, higher automation levels require still further technical progress. On the other hand, regulations and standards have to be extended and guidelines developed to provide a solid framework for the development and approval of ADSs. With increasing complexity of ADSs unified regulations, *e.g.* by the United Nations Economic Commission for Europe (UNECE), become important. A first step in this direction is the UN-R157 [2021] for Automated Lane Keeping Systems with SAE-Level 3. However, international regulations and standards for ADSs have deficits [Koller and Matawa, 2020]. Until harmonised regulations exist, the German government passed laws for automated [BGBl, 2017] and autonomous driving [BGBl, 2021] in the form of modifications of the Straßenverkehrsgesetz (StVG). Furthermore, car manufacturers work on additional guidelines like white papers, *e.g.* SaFAD [2019], to address gaps of existing regulations and standards.

Safety Assurance for Automated Driving and Driver Assistance

A key factor for the difficulty of safety assurance of ADSs ($\geq$ SAE-Level 3) is a significant difference in the automation level compared to ADASs ($\leq$ SAE-Level 2). For ADASs, the human driver is in full charge. This requires that the driver continuously supervises the driving task and must be prepared to take over control at any time. Contrary, ADSs do not have the human fallback and the car manufacturer takes responsibility for safe driving if the ADS is active. This fact implies significant differences for the safe development and the approval of ADASs and ADSs. Approving the safety of an ADAS focuses on the *controllability* and *takeover* by the driver and the reduction of the software and hardware related *failure rate*. To evaluate controllability, test drives are performed and it is assumed that the assessment of the test driver can be transferred to the customer. The reduction of Electrical and/or Electronical (E/E) malfunction is addressed by the functional safety [ISO26262, 2018]. For ADSs, the main challenge is to ensure a *safe intended behavior* even under the assumption that no software and hardware malfunction occurs. This is challenging due to the huge diversity of driving scenarios and the requirement that ADSs must handle these scenarios safely on their own. Furthermore, not all scenarios can be

[1] Type approval of DRIVE PILOT by Kraftfahrt-Bundesamt `https://www.kba.de/DE/Presse/Pressemitteilungen/Allgemein/2021/pm49_2021_erste_Genehmigung_automatisiertes_Fahren.html`, accessed on 25/12/2022

specified and a challenge is to reduce the risk of unknown and rare scenarios. These aspects are part of the SOTIF [ISO21448, 2022], which includes foreseeable misuse by persons as well.

Dimensions of Safety and Corresponding Standards

Main aspects of safety and security are covered by the standards for Functional Safety (FuSa) [ISO26262, 2018], SOTIF [ISO21448, 2022], and cybersecurity [ISO/SAE21434, 2021]. Further, standards exist or are under development with more specific scopes, *e.g.* for software updates ISO24089 [2023]. The cited standards relate to E/E systems within road vehicles. Central parts of the FuSa are the identification of hazards due to E/E faults, the assessment of these hazards into Quality Management (QM) and Automotive Safety Integrity Level (ASIL) A to D, a concept to detect, to avoid, and to mitigate the hazards, and Verification and Validation (V&V). The V&V process examines whether the system meets the imposed requirements, *e.g.* a maximum permitted failure rate. Contrary, SOTIF focuses on safety under the assumption that no malfunction occurs and the function works as intended. The focus is on systems that require situational awareness and on foreseeable misuse by persons. This includes the mitigation of risks due to performance limitations and due to technological limitations of sensors and actors. ISO21448 [2022] distinguishes a design and V&V phase. Finally, cybersecurity considers hazards that are caused by a third party or intruder. The third party manipulates or attacks the systems.

Guidelines for Trustworthy Artificial Intelligence

A key technology for ADSs is AI, in particular ML and DL. ML algorithms and NNs learn the behavior from data and are characterized by an opaque decision-making process, the so-called black box character [Rudin, 2019]. This black box character implies additional challenges for the SOTIF and cybersecurity. However, the corresponding standards [ISO21448, 2022; ISO/SAE21434, 2021] consider AI only partially. Therefore, fundamental research (see Figure 1.1), additional guidelines, and extended standards and regulations are required for trustworthy applying AI.

The white paper SaFAD [2019] draws attention to a lack of guidelines, standards, and regulations for a responsible usage of AI. In the appendix of SaFAD [2019], an own guideline for the integration of DL in ADSs is proposed. On the example of a Three Dimensional (3D) object detection, safety requirements, measures and guidelines are explained on a technical level. Good practices and state-of-the-art approaches are summarized, structured, and assigned to the development steps 1) define, 2) specify, 3) develop and evaluate, and 4) deploy and monitor. The focus is on a proper data set creation, runtime monitoring, and on mechanisms for plausibilization, robustness, and uncertainty estimation.

The European Commission announced a proposal for the first regulation of AI worldwide [European-Commission, 2021]. The proposal categorizes systems according to the risk in three classes: unacceptable risk, high risk, and low or minimal risk. Safety-critical systems for ADSs are expected to be systems with high risk. These systems would need to comply with AI requirements and undergo a conformity assessment. Central is the establishment of a risk management system (Article 9) to identify, analyze, and evaluate risks and to adopt appropriate measures. A post-market monitoring system should be used, it is referred to state-of-the-art measures including harmonised standards, and test procedures shall be consistently applied. Further requirements address high quality data (Article 10), technical documentation (Article 11), record-keeping (Article 12), transparency and provision of information to users (Article 13), human oversight (Article 14), and accuracy, robustness, and cybersecurity (Article 15).

Conclusion

The presented overview indicates the high complexity of safety assurance for ADASs and ADSs. The development of safe DASs requires state-of-the-art measures and practices, should be treated holistically, covers the whole development process, relies on quantitative and qualitative approaches, needs to be continuously adapted due to the fast dynamic in the field of AD and AI, and must still be manageable in practice.

2.1.2 Safety of the Intended Functionality

The SOTIF [ISO21448, 2022] addresses the minimization of the residual risk to an acceptable level and considers hazards caused by

- insufficiencies of the specification of the intended functionality at vehicle level and
- insufficiencies of the specification and performance limitations of E/E elements.

In particular, this includes systems that require a proper situational awareness. Furthermore, SOTIF covers reasonable foreseeable misuse, *e.g.* due to lack of driver attention while using a SAE-Level 2 system. Hazards due to intentional misuse, lack of security [ISO/SAE21434, 2021], and malfunction of the E/E system [ISO26262, 2018] are out of scope of SOTIF. Moreover, ML-based algorithms have become important for safe DASs. Compared to rule-based algorithms, essential differences need to be considered because these algorithms learn complex and usually not interpretable representations from data. In the appendix, the SOTIF standard addresses aspects for the safe development of ML-based algorithms.

A central element of the SOTIF are scenarios to describe different use cases and foreseeable misuse of the system. They can include triggering conditions for SOTIF-related hazardous behavior. The scenarios are categorized into four categories: 1) known and not hazardous, 2) known and hazardous, 3) unknown and hazardous, and 4) unknown and not hazardous. The goal of SOTIF is the maximization of known scenarios and the minimization of hazardous scenarios to achieve an acceptable residual level of risk. This principle is visualized in Figure 2.1 with a Venn diagram.

To achieve the SOTIF the standard proposes a process (Clauses 5-13 in ISO21448 [2022]), which is visualized in Figure 2.2. The process includes the identification of hazards and their evaluation *w.r.t.* the acceptable risk (Clause 6), the analysis of possible root

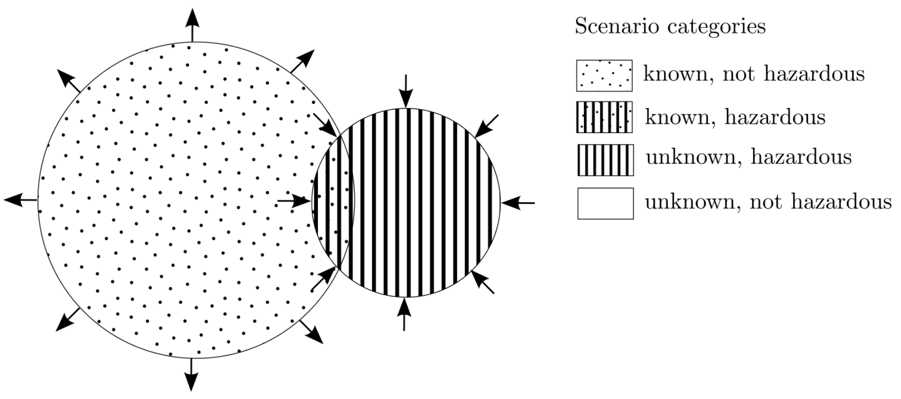


Figure 2.1: The abstract goal of SOTIF activities is the minimization of known and unknown hazardous scenarios and the maximization of known scenarios. The figure has been adapted from Figure 5 in ISO21448 [2022, p.13].

causes and triggering conditions (Clause 7), activities related to the specification and design (Clause 5), V&V measures (Clause 9,10,11), the operation phase (Clause 13), the evaluation of the residual risk for the SOTIF release (Clause 12), and modifications of the functionality if necessary (Clause 8). In the following, essential aspects of the design and the V&V phase are described. Finally, the suggestions of SOTIF for ML-based algorithms are presented.

Safety by Design

The starting point of a SOTIF conform development is the specification and design phase (Clause 5). This step includes the specification of the intended functionality and foreseeable misuse at vehicle and subsystem level, the dependencies of the intended functionality with interacting systems (*e.g.* driver, environment), the system design and architecture to implement the intended functionality, and the performance targets. Furthermore, performance limitations and countermeasures, a warning and degradation concept (*e.g.* by a driver takeover strategy or minimal risk maneuvers), and a data collection system (*e.g.* to record critical scenarios) should be considered. Safety is explicitly addressed by stating requirements at vehicle-level to achieve SOTIF. The set of these requirements is called Vehicle Level Safety Strategy (VLSS).

A crucial part of the development of ADSs is the design of a safe driving policy. ISO21448 [2022, p.159-168] proposes the analysis of different areas of concern to ensure the completeness of the VLSS and the definition of a safe driving policy. The areas of concerns can be classified depending on whether they are derived from the AV operating environment (*e.g.* ODD, road infrastructure), the interaction between the AV and the occupants of the AV (*e.g.* driver takeover), or the interactions between the AV and other participants (*e.g.* defensive driving policy). The RSS model [Shalev-Shwartz et al., 2017] and the SFF [Nistér et al., 2019] are approaches to implement safe driving policies by taking other traffic participants and the road infrastructure into account and cover thereby two areas of concern. Both approaches are presented in Section 2.1.3 and impose sets of safe actions.

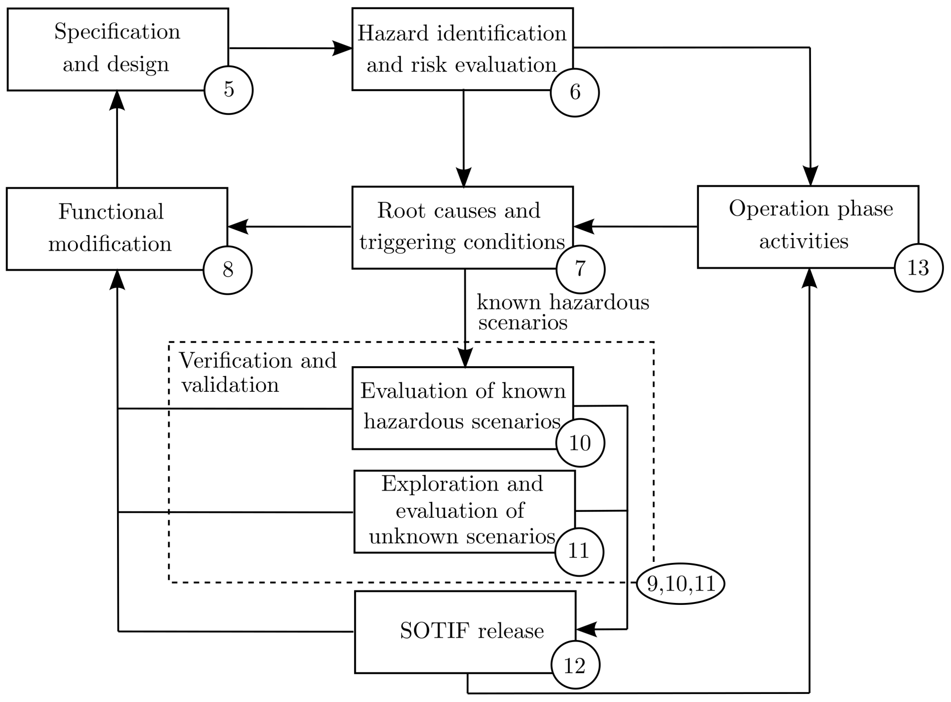

Specification and design
5
Hazard identification and risk evaluation
6
Functional modification
8
Root causes and triggering conditions
7
Operation phase activities
13
known hazardous scenarios
Verification and validation
Evaluation of known hazardous scenarios
10
Exploration and evaluation of unknown scenarios
11
9,10,11
SOTIF release
12

Figure 2.2: The SOTIF standard is structured according to the visualized process. The SOTIF activities (rectangles) are described in Clauses 5-13 of ISO21448 [2022]. The corresponding clause of each activity is denoted in the bottom right circle of each rectangle. The process starts with a safe design of the system (Clause 5). Next, hazards are identified, evaluated (Clause 6) and analyzed *w.r.t.* functional insufficiencies and initiating conditions (Clause 7). Either a functional modification (Clause 8) is required or a V&V strategy (Clause 9,10,11) is applied to provide a rationale that the residual risk is sufficiently low. If the V&V activities are successful the system is ready for release (Clause 12) and operation phase activities (Clause 13) observe the SOTIF after release. The figure has been adapted from Figure 5 in ISO21448 [2022, p.18].

The specification and design phase is closely related with other SOTIF activities and should be understood as part of an iterative process. *E.g.* if a hazardous behavior with unreasonable risk is detected (Clause 6) (*e.g.* a sudden braking request due to a false positive detection of the perception can lead to a rear collision), the insufficiencies of specification, performance limitations and triggering conditions need to be identified (Clause 7), countermeasures (*e.g.* changing the sensor technology) identified and applied (Clause 8) and the specification and design documents updated (Clause 8). The work products of the specification and design phase are detailed documentations.

Verification and Validation

The goal of V&V (Clause 9,10,11) is providing evidence that the residual risk of hazards due to functional insufficiencies is below an acceptable level. To evaluate the residual risk, tests are performed on known hazardous (Clause 10) and unknown (*e.g.* randomly generated or explored) potentially hazardous scenarios (Clause 11). For a successful V&V, validation targets need to be achieved, which are derived from acceptance criteria of identified risks (Clause 6).

The V&V phase builds on the working products of previous SOTIF activities. The hazard and risk analysis (Clause 6) identifies SOTIF-related hazards without specifying the scenarios and the functional insufficiencies that cause them. For each hazard, the risk is evaluated and an acceptable level of risk is defined. These acceptance criteria define the validation targets. Next, the scenarios and conditions that trigger hazardous behavior and functional insufficiencies (Clause 7) are identified and thereby the area of known hazardous scenarios is extended. To transform known hazardous scenarios to known not hazardous scenarios or to reduce the risk of known hazardous scenarios, an appropriate functional modification (Clause 8) is derived and applied. After a functional modification, the activities in Clause 6 and 7 must be adapted as well. If the SOTIF is considered to be achievable, the verification procedures (Clause 10) comprise tests on the identified known hazardous scenarios. It shall be demonstrated that the functionality at vehicle and component level behaves as specified and that the residual risk is below the defined acceptable level. The goal of validation is to provide a rationale that the residual risk of unknown scenarios is acceptable and meets the validation targets. Unknown scenarios can be generated by sampling scenario parameters, combining known scenarios, or by exploring new scenarios in real-world testing. Comprehensive operation conditions are important for sufficient scenario coverage. Common methods of V&V are Model In the Loop (MIL), Software In the Loop (SIL), and Hardware In the Loop (HIL) testing as well as real-world vehicle testing.

For demonstration, let us consider a statistical validation method [Wachenfeld and Winner, 2015, p.454-458]. The method is a hypothesis test with the null hypothesis that the DAS performs worse than an accepted threshold. Given the null hypothesis, the probability to observe less or equal a fixed number of incidents k_0 during validation rides of length s_0 is evaluated. If the probability is small and indeed less or equal to k_0 incidents are observed during validation rides of length s_0, the null hypothesis can be rejected in favor of the alternative hypothesis that the DAS is better than the accepted threshold. In the context of hypothesis testing, small means that the probability is smaller than $1-\gamma$ with γ being a set confidence, *e.g.* $\gamma=95\%$. The validation rides are intended to be performed in the ODD of the DAS and the acceptance criterium is defined as a required minimum average distance s_{accepted} between critical incidents (*e.g.* hazardous events or events with harm). For ADSs, the value can be chosen as the corresponding distance for humans s_{human} (*e.g.* from traffic statistics) multiplied with a safety margin $a_{\text{safety}} \geq 1$: $s_{\text{accepted}} = s_{\text{human}} \cdot a_{\text{safety}}$. Next, for a set confidence γ and a fixed maximally allowed number of incidents k_0 (*e.g.* $k_0=0$) the distance s_0 of the validation rides is determined. In the derivation, it is assumed that the number of observed incidents k is distributed according to the Poisson distribution:

$$P_\lambda(k) = \frac{\lambda^k}{k!}\exp(-\lambda), \tag{2.1}$$

with $\lambda = s_0/s$ the mean number of incidents and s the unknown mean distance between incidents of the DAS. Given the null hypothesis $\lambda > s_0/s_{\text{accepted}}$, the probability for observing less or equal to k_0 incidents $P_\lambda(k \leq k_0)$ is evaluated:

$$\forall \lambda > s_0/s_{\text{accepted}} : \quad P_\lambda(k \leq k_0) < 1 - \gamma. \tag{2.2}$$

If the equation holds and indeed less or equal to k_0 incidents are observed, the null hypothesis can be rejected and the alternative hypothesis is assumed to be true. *I.e.* the DAS is at least as good as the reference $\lambda \leq s_0/s_{\text{accepted}}$. Consequently, the minimum required distance for the validation rides is the smallest distance s_0 that fulfills Equation (2.2).

E.g. the required distance of validation rides that must be driven without observing any incident ($k_0 = 0$) derives to $s_0 \geq -\log(1-\gamma)s_{\text{accepted}}$. For $\gamma = 95\%$, this is roughly three times the reference distance s_{accepted}. If the validation rides are designed to allow an increasing number of incidents k_0, the required distance increases too. However, the validation is with a higher probability successful as well. Wachenfeld and Winner [2015, p.456-458] pointed out that this statistics-based approach might prove impracticable if no further considerations are included. In particular, hazards with high severity (*e.g.* fatalities) and large demanded distances between occurrences require a huge validation effort. To reduce the amount of validation kilometers, several measures are of interest. The following examples summarize possible measures to accelerate validation.

- The functional behavior *w.r.t.* generated unknown scenarios can be validated partially with MIL, SIL, HIL testing. However, real-world vehicle testing can not be substituted because simulation always simplifies the environment, vehicle, and sensors.
- Potentially, validation rides can be optimized by focusing on scenarios with higher criticality. *E.g.* to analyze rear-end collisions, scenarios without surrounding vehicles are not of interest and scenarios with short distances to a vehicle in front should be stressed. The driving routes and simulations may be optimized to capture scenarios of interest faster.
- The rate of hazardous behavior is typically significantly higher than the rate of events with harm and requires consequently less validation effort. The conditional probability of an event with harm given a hazardous behavior can be estimated and the validation effort can be reduced accordingly.
- The validation effort can be reduced by analyzing the system architecture. *E.g.* if a function is redundantly implemented, the subsystems can be validated with less effort and combined [Shalev-Shwartz et al., 2017].
- Regression testing after functional modifications can potentially be reduced by proper argumentation and referring to previous and transferable validation activities.

Considerations for Machine Learning-based Algorithms

The functionality of ML-based algorithms is not specified by the developer explicitly but rather learned implicitly from data during training. The final model appears typically as a black box. On the one hand, this black box character makes safety assurance significantly more difficult. On the other hand, these algorithms achieve in many domains state-of-the-art performance. Therefore, not applying these algorithms would be irresponsible as well. In Appendix D.2 of ISO21448 [2022], this challenge is addressed by a first guidance and additional considerations for ML-based algorithms.

Despite the impracticability of specifying the functionality of ML-based algorithms on a low level, the specification of use cases, scenarios, and the ODD is still the starting point of the safety assurance process (Clause 5). The specification and design phase is also the basis for the collection of an appropriate data set. Furthermore, ML techniques to reduce the black box character can be a crucial part of the safety strategy (*e.g.* Bayesian Deep Learning for uncertainty estimation) and the analysis of potential ML-specific limitations is important. However, functional insufficiencies may be difficult to identify by analysis (Clause 6 and 7). Therefore, the identification of relevant test cases and scenarios becomes the main objective of Clause 6 and 7. The functional insufficiencies are mainly revealed by V&V (clause 9,10,11) and operational phase activities (Clause 13). Identified functional insufficiencies can be addressed with functional modifications (Clause 8) that involve typically a retraining. The opaque decision-making process of ML-based algorithms makes the comparison of the new model to the previous one typically impracticable. This implies that previous tests are in general no longer valid and must be repeated. Moreover, the SOTIF standard summarizes important guidelines for correctly performing the training process as the crucial part to achieve a safe model behavior. *E.g.* data set annotations should be checked, rare but critical scenarios appropriately represented, and training, validation, and test sets collected with sufficient scenario diversity and coverage.

2.1.3 Formal Methods for the Safety of the Intended Functionality of Driving Policies

In this section, two formal methods to ensure the SOTIF of driving policies are presented, namely the RSS model [Shalev-Shwartz et al., 2017] and the SFF [Nistér et al., 2019]. The goal is the avoidance of collisions that are caused by the ego vehicle's driving policy. Addressing this, constraints are computed from the current state of the perceived environment and imposed on the control input. The control input comprises demanded accelerations and/or steering commands for controlling the lateral and longitudinal motion. Delays for setting the control input are taken into account. In the following, the feasible set of control commands that results from the constraints is referred to as safe set C. The principle of deriving the control constraints from kinematic relations and imposing the corresponding safe set is visualized in Figure 2.3. The RSS model and the SFF depend on a perception that detects and localizes the surrounding actors and obstacles reliable and accurately, as well as their states of movement. Additionally, the RSS model requires a reliable detection of the road geometry because it builds on a lane-based coordinate system. Contrary, the SFF argues that solely the movement and the inertia of the road-users are fundamental for collision avoidance and static infrastructure is incorporated on a secondary level of importance. The key challenges for methods for the SOTIF of driving policies can be summarized as follows:

- Assumptions about the model of the environment must be balanced. Note, it is impossible to achieve complete safety independently of the behavior of other vehicles. *E.g.* if the ego vehicle is surrounded by other vehicles, a possibly necessary evasive maneuver becomes impossible. Consequently, the RSS model and the SFF propose to consider safety under a symmetry assumption. If all traffic participants

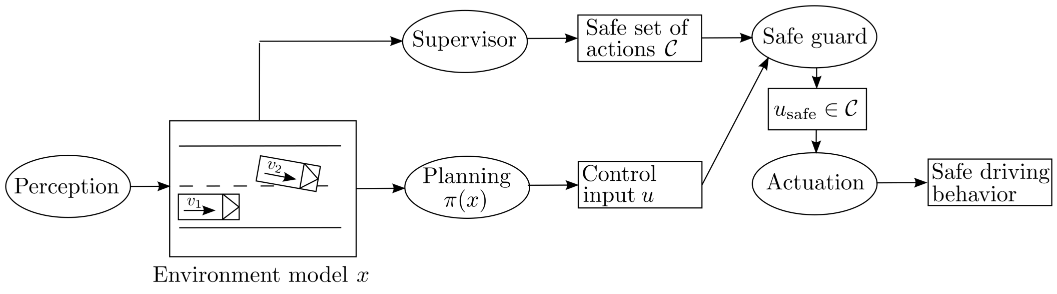


Figure 2.3: An approach for the SOTIF of the driving policy is the determination of a safe set C and imposing it on the control input u. The common architecture of an AV according to the sense-plan-act model is extended by a supervisor and a safe guard. The supervisor computes constraints on the control input based on the perceived model of the environment x. The resulting feasible set of safe control inputs is denoted as safe set C. Typically, the supervisor is a rule-based model (*e.g.* RSS model or SFF) and incorporates the movement and the inertia of surrounding actors among other information. The safe guard or safety layer ensures that the final control input u_{safe} is within the safe set C and corrects the original control input u of the planning step if necessary. This safety concept is designed to be independent of the prediction and planning logic.

obey the safety rules, zero accidents should be guaranteed. However, safety under the symmetry assumption is not sufficient, *e.g.* crashes due to misbehavior of other road-users shall be prevented in a reasonable extent as well.

- The safest driving policy is not moving at all. Obviously, an acceptable level of traffic flow is required that passengers reach their destination in a reasonable time. This requires balancing between safety and traffic flow.
- Formal safety models require a high-quality perception *w.r.t.* recall, confidence, and accuracy. In particular, false negatives may lead to undetected dangerous scenarios. Performance limitations in the perception can not be compensated fully on a later stage. Thus, performance targets must be validated with an efficient concept.

Responsibility-Sensitive Safety Model and Safe Sensing

The RSS model derives so-called *proper responses* that overrule the control input if dangerous situations are detected. If all road-users behave according to the proper responses, collision avoidance is ensured under the assumption of no sensing errors. Furthermore, the proper responses are extended for evasive maneuvers to react properly on improper behavior of other road-users. On a high level, the RSS model implements the following five common sense rules:

1. "Do not hit someone from behind."
2. "Do not cut-in recklessly."
3. "Right-of-way is given, not taken."
4. "Be careful of areas with limited visibility."
5. "If you can avoid an accident without causing another one, you must do it."

These rules are cited literally from Shalev-Shwartz et al. [2017, p.6]. The proper responses are formalizations of these rules with collision avoidance guarantees. However, the guarantees rely on a perfect perception of the environment. This is non-realistic and in practice typically a low probability of sensing errors is evaluated as acceptable. Thus, Shalev-Shwartz et al. [2017] propose additionally a concept to achieve a high-quality perception and to validate the performance efficiently.

The RSS model starts by introducing a lane-based coordinate system with well-defined longitudinal and lateral axes. Then, lateral and longitudinal minimal safe distances are defined and serve as thresholds for activating a proper response. In the longitudinal case, the rear vehicle is responsible for collision avoidance even under a reasonable worst-case scenario of the front vehicle. The rear vehicle needs to brake at least within a response time after undershooting the longitudinal minimal safe distance (Rule 1). Contrary, in the lateral case both vehicles are supposed to apply a symmetric proper response to reduce the lateral velocity to zero. A vehicle that approaches another vehicle on the neighboring lane is responsible to brake in lateral direction within a response time after undershooting the minimal lateral distance (Rule 2). If the vehicle on the neighboring lane performs a lane keeping maneuver with a lateral velocity of almost zero, it must not change its state of movement except the cutting-in vehicle does not behave properly and Rule 5 is valid. Rule 5 states that an evasive maneuver is mandatory if other road-users do not respond properly and if the evasive maneuver does not violate the proper response with respect to all other road-users. For a scenario with multiple vehicles, the control constraints are computed for each vehicle separately and then the minimum is taken. Thereby, all individual constraints are satisfied. This method is referred to as *star-shape computations* in Shalev-Shwartz et al. [2017]. Furthermore, the rules are extended for intersecting routes with a priority concept (Rule 3) and a careful behavior for scenes with occlusion is defined (Rule 4). All rules depend on parameters, which have to be chosen carefully. Shalev-Shwartz et al. [2017] propose to use reasonable worst-case assumptions as a trade-off between safety and traffic flow.

The RSS model provides explicit rules and formal guarantees for collision avoidance given a perfect perception. However, for the application in AVs a high-quality sensing technology and a validation of the performance is required. Shalev-Shwartz et al. [2017] address this as well by introducing a scalable concept. A main difficulty is the validation of rare event rates, which is here the rate of sensing errors. As demonstrated in Section 2.1.2, the validation of rare events requires an enormous amount of test data. Thus, Shalev-Shwartz et al. [2017] propose a redundant perception consisting of several subsystems. The subsystems should be independently developed and should rely on different technologies. For the redundant perception, a c-approximated independence assumption of sensing errors is reasonable ($p(x_1,x_2) \leq c \cdot p(x_1) \cdot p(x_2)$). *E.g.* a sensing system with three c-approximated independent subsystems, each with error rate e_i, results in a tremendously lower error rate $e \leq 3ce_i^2$ after fusion, *i.e.* $e_i = 10^{-5}$ results in $e = 10^{-9}$ ($c \approx 3$). Consequently, the validation effort can be significantly reduced by a validation of the acceptable higher error rates of the subsystems compared to a validation of the much lower error rate of the total system. In particular, a complementary sensing system based on camera, Light Detection And Ranging (LiDAR), and RAdio Detection And Ranging (RADAR) is suggested as optimal. Furthermore, a mapping and localization system is recommended and called Road Experiment Management.

The Safety Force Field

Similar to the RSS model, the SFF Nistér et al. [2019] defines so-called *safety procedures*, which can overrule the control input if a dangerous situation is detected. If all road-users behave accordingly, collision avoidance is ensured under the assumption of no sensing errors. However, this is not sufficient and it is important to define proper behavior for improper behavior of other road-users. Instead of defining exemption rules like in the RSS model, the safety procedure is considered just as a baseline and fallback in favor of finding a control policy that is at least as good or even better. A compact mathematical expression is derived to compare another control policy against the safety procedure. This capability improves the situational awareness of the approach and leads to a less defensive driving policy. Furthermore and contrary to the RSS model, the SFF does not necessarily require a lane-based coordinate system. Instead, the SFF focuses intentionally on a simple as possible environment model, which includes at least the static and dynamic objects as well as their state of movements. Thereby, the SFF depends less on complex external structures, which are difficult to perceive reliably. Nistér et al. [2019] argue that this is an important strength because the perception and its validation is a key challenge of AD. On the one hand, the SFF is formulated general and is applicable to arbitrary safety procedures and vehicle models. On the other hand, the implementation for reasonable safety procedures and realistic vehicle models is not clear and Nistér et al. [2019] show only the computations for a simplified example.

The core concept of the SFF can be summarized as follows. First, a safety procedure is defined, which is typically the reduction of the lateral motion and a braking to standstill. The safety procedure describes a family of trajectories, which cover a certain volume in space-time. The volume is called *claimed-set* of an actor. The basic principle is that all road-users perform their safety procedure or a better control policy just before and whenever the claimed sets overlap. Thereby, collision avoidance is guaranteed if no errors in the perception are assumed. A so-called *safety potential* is introduced for two actors and serves as a soft indicator function for whether the claimed sets of the actors overlap or not. The SFF of the ego vehicle induced by another road-user is defined as the negative gradient of their safety potential *w.r.t.* the ego vehicle's state. Intuitively, moving along the SFF is beneficial because it repels the claimed sets from overlapping. This is formalized mathematically as follows. If the dot product of the SFF and a control policy is greater or equal than the dot product of the SFF and the safety procedure, the control policy is called safe. A safe control policy inherits the collision avoidance guarantee from the safety procedure and can even improve safety. This can be seen in the following example. If another road-user violates the rules and approaches the ego vehicle from the side, moving over is frequently a safe control policy and better than keeping straight driving according to the safety procedure. For modeling of realistic conditions, the SFF is extended to take limited visibility and delays in the perception and control into account. Traffic rules are intended to be stacked on top of the model. This clearly separates collision avoidance from complex dependencies on the environment.

2.2 Neural Networks

In the last decades, Artificial Neural Networks (ANNs) have received significant attention in research. Improvements in the architecture of ANNs [LeCun et al., 1998; He et al., 2016; Vaswani et al., 2017], in optimization and regularization techniques [Kingma and Ba, 2014; Srivastava et al., 2014], and in the increasing power of computer hardware have been important factors for this development and the success of ANNs. Nowadays, ANNs represent the state-of-the-art in many competitive learning challenges [Russakovsky et al., 2015]. ANNs are used as flexible function approximators and successful training algorithms have been developed in the field of supervised learning [LeCun et al., 1998], unsupervised learning [Kingma and Welling, 2014], and RL [Mnih et al., 2015]. In this chapter, the focus is on optimizing ANNs with supervised learning. Chapter 2.3 provides a general overview of RL including DRL, *i.e*. RL algorithms that use ANNs as function approximators.

2.2.1 Basics: Artificial Neurons, Layers, and Neural Networks

The term NN refers either to a network of biological or artificial neurons. Artificial neurons are inspired by their biological counterparts but are strongly simplified in their behavior. Despite the simplification, ANNs are able to perform complex tasks on high dimensional data like object detection on images [Redmon et al., 2016] or language translation [Vaswani et al., 2017]. For the remainder of the thesis, NNs refer to ANNs.

Artificial Neurons

An artificial neuron combines K inputs $\{x_k\}$ linearly and subsequently applies a non-linear function, the so-called *activation function* g:

$$y = g\left(\sum_{k=1}^{K} w_k \cdot x_k + b\right), \tag{2.3}$$

with y the output and w_k, b the weights or parameters of the neuron. The parameter b is called *bias* and the input of the activation function *pre-activation score*. An artificial neuron is visualized in Figure 2.4 a). Common activation functions are:

- the Rectified Linear Unit (ReLU) $g(z) = \max(0, z)$,
- the leaky ReLU $g(z) = \max(\alpha\, z, z)$ with a small α $(0 < \alpha \ll 1)$,
- the sigmoid function $g(z) = \mathrm{sig}(z) = 1/(\exp(-z) + 1)$,
- and the hyperbolic tangent $g(z) = \tanh(z)$.

A predecessor of the artificial neuron is the *perceptron* [Rosenblatt, 1958]. The perceptron can be considered as an artificial neuron with the Heaviside step function as activation

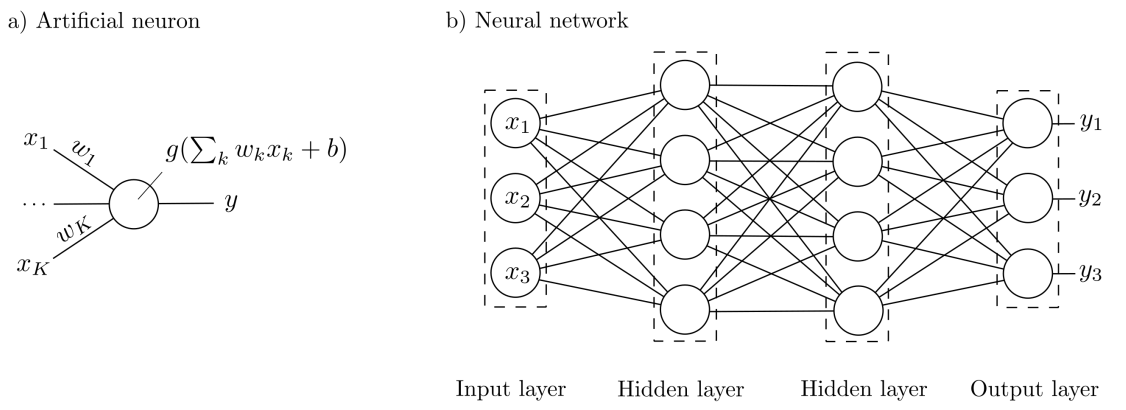


Figure 2.4: In this figure, neurons are represented as circles, inputs as edges on the left side of neurons, and outputs as edges on the right side of neurons. a) An artificial neuron generates the output by applying a non-linear activation function g to the weighted sum of the inputs x. Optionally, a bias b may be added as an additional parameter to the weights w_k. b) An NN consists of several layers (dashed rectangles). Here, a feedforward NN is shown and dense layers are arranged in a sequence. The information is processed from the left to the right-hand side. Each neuron of the first layer—the *input layer*—represents one element of the input x. Then *hidden layers* are applied and generate intermediate representations. Finally, the *output layer* generates the output y of the NN by taking the output of the last hidden layer as input.

function. By performing this thresholding, the perceptron becomes a binary classifier. In other words, an artificial neuron is a generalization of the perceptron.

Layers

Layers represent the next higher building block of NNs and refer classically to a collection of neurons in a certain arrangement. A common class of layers are *dense layers* (also called fully-connected layers). These layers consist of J neurons with individual weights. The neurons process the same K-dimensional input x in parallel and generate a J-dimensional output vector y:

$$\begin{aligned} y_j &= g\left(\sum_{k=1}^{K} w_{j,k} \cdot x_k + b_j\right) \\ \Leftrightarrow y &= g\left(Wx + b\right) = g\left(\tilde{W}\tilde{x}\right). \end{aligned} \tag{2.4}$$

The second line in Equation (2.4) shows the matrix and vector notation: $(W)_{j,k} = w_{j,k}$, $(y)_j = y_j$, $(x)_j = x_j$, $(b)_j = b_j$, and g is applied element-wise. Frequently, an extended input $\tilde{x}$ ($\tilde{x}_0 = 1$ and $\tilde{x}_k = x_k$) and an extended weight matrix $\tilde{W}$ ($\tilde{W}_{j,0} = b_j$, $\tilde{W}_{j,k} = w_{j,k}$) are introduced to compute the pre-activation scores with a pure matrix multiplication.

A second important class of layers are *convolutional layers*, which build the core of many state-of-the-art NN architectures, *e.g.* ResNet [He et al., 2016]. In Figure 2.5, a 2D convolutional layer is illustrated. The input x and output y is a 3D tensor consisting of multiple 2D feature maps. The index of a feature map is called channel. *E.g.* the three color channels of a 2D RGB-image are a valid input for a 2D convolutional layer. Note, the

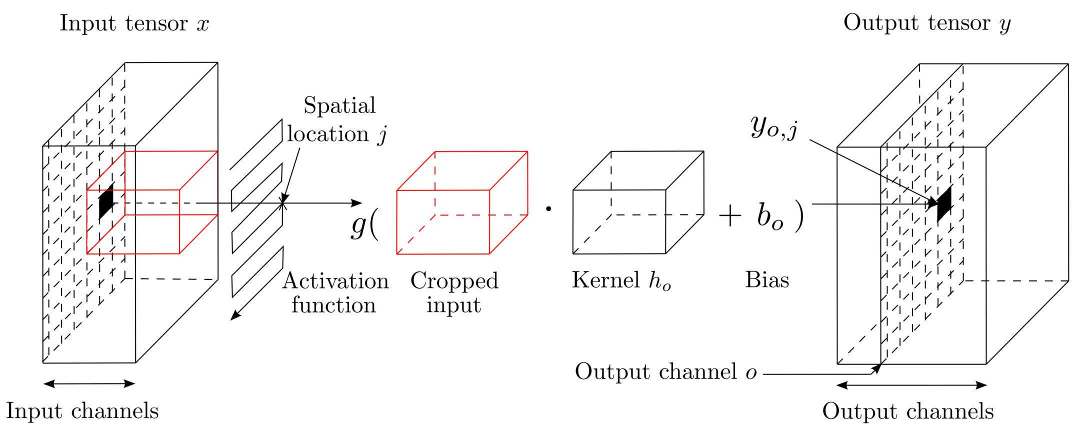


Figure 2.5: A Two Dimensional (2D) convolutional layer is illustrated. For each output channel o, a channel-specific kernel h_o is slid over the input tensor. The output value $y_{o,j}$ at the spatial location j is generated by computing the inner product of the kernel and the covered part of the input tensor (red), adding an optional bias b_o, and applying an activation function g. Whereas different output channels have separate kernel parameters, the kernel parameters within an output channel are shared across locations.

number of channels of the input and output tensors can be chosen independently. For each output channel o, a 3D kernel h_o with the same number of channels as the input tensor is slid over the input tensor x and the inner product of the kernel and the covered part of the input tensor is computed. This can be formalized as a sum of discrete cross-correlations ($\otimes$) over the input channels c:

$$\sum_c (h_{o,c} \otimes x_c)(j) = \sum_c \sum_k h_{o,c}(k) \cdot x_c(k+j) = \sum_c \sum_k h_{o,c}(k-j) \cdot x_c(k), \tag{2.5}$$

with $h_{o,c}$ being a kernel slice, x_c an input feature map, and k, j 2D spatial indices. In Figure 2.5, the kernel is slid only over spatial locations j in the interior of the input tensor so that the kernel does not exceed the input tensor boundaries. Consequently, the output tensor is smaller in the spatial dimensions. However, frequently the input tensor is enlarged by padding in order to maintain the spatial dimensions. Common padding operations are *e.g.* adding entries with zeros or mirroring the tensor at the boundary for the additional entries. The final output y_o of the channel o is generated by adding an optional and channel-specific bias b_o and applying an activation function g:

$$y_o = g\left(b_o + \sum_c (h_{o,c} \otimes x_c)\right). \tag{2.6}$$

For an n-dimensional convolutional layer, the formula is still valid. However, in this general case the input and output tensors are $n+1$ dimensional. Here, n is the dimension of the spatial indices and the additional dimension is for the channels. Although a cross-correlation is applied in Equation (2.6), the term convolutional layer is justified by the fact that a cross-correlation is equivalent to a convolution ($*$) with a flipped kernel $\tilde{h}_{o,c}(l) = h_{o,c}(-l)$:

$$(h_{o,c} \otimes x_c)(j) \overset{(2.5)}{=} \sum_k h_{o,c}(k-j)x_c(k) = \sum_k x_c(k)\tilde{h}_{o,c}(j-k) \tag{2.7}$$
$$\overset{\text{def}}{=} (x_c * \tilde{h}_{o,c})(j) = (\tilde{h}_{o,c} * x_c)(j).$$

Convolutional layers share the parameters of the kernel between neurons. This can be seen by substituting $h_{o,c} \otimes x_c$ in Equation (2.6) with Equation (2.5):

$$y_{o,j} = g\Big(b_o + \sum_c \sum_k \big(h_{o,c}(k-j) \cdot x_c(k)\big)\Big). \tag{2.8}$$

Both neurons, the one generating the output $y_{o,j}$ and the one generating the shifted output $y_{o,j+l}$, share the same weight $h_{o,c}(k-j)$ for the input $x_c(k)$ and the shifted input $x_c(k+l)$, respectively. Contrary, in a dense layer the weight of an input is not used by other neurons (compare with Equation 2.4). The parameter sharing is an important property of convolutional layers and reduces the number of learnable parameters significantly. Furthermore, convolutional layers are equivariant *w.r.t.* translations. *I.e.* shifting the input by l and then applying the convolutional layer is equivalent to first applying the convolutional layer to the original input and then shifting the output by l.

More generally, a layer can be defined as a differentiable function with optional learnable parameters θ mapping an input x to an output y. Differentiability is required for the optimization and training of the NN (see Section 2.2.2). The more general definition covers also special layers like differentiable optimization layers [Agrawal et al., 2019]. In these layers, the output is the solution of an optimization problem that is defined by the input and optional learnable parameters. In a classical layer, the learnable parameters comprise the parameters of the neurons. Shared parameters mean that the same parameters are used by several neurons.

Neural Networks

NNs consist of interconnected layers and the resulting network of neurons represents a flexible function approximator. Frequently, state-of-the-art performance requires NNs with a sufficient number of layers, so-called Deep Neural Networks (DNNs) [He et al., 2016; Vaswani et al., 2017]. However, theoretically an NN with only one hidden layer and a sufficient large number of neurons is capable of approximating any Borel measurable function to any degree of precision [Hornik et al., 1989].

In an NN, the non-linear activation functions of the neurons have an important role. Without them, a neuron would be a linear function and a network of neurons would remain linear. *E.g.* two dense layers without activation functions are equivalent to one linear layer

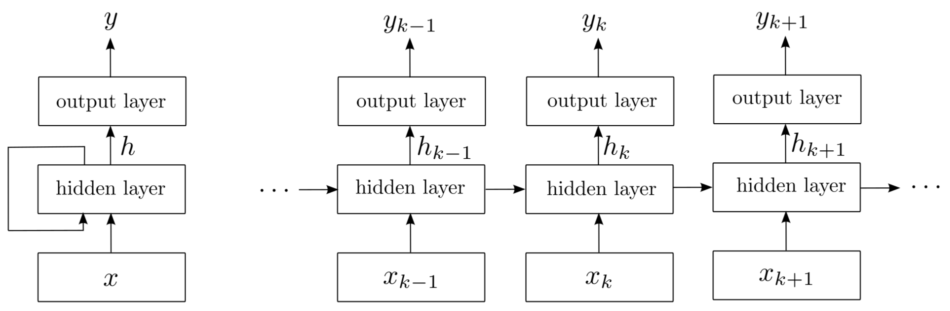


Figure 2.6: *Left:* In RNNs, outputs of neurons are fed back as input of the same or previous layer. Thereby, RNNs are able to process a sequence of inputs $(x_1, \ldots, x_k, \ldots, x_K)$ with variable length K and the recurrent connections allow to leverage the temporal context. *Right:* The RNN is unfolded into an NN without cycles by taking a copy of the network at each time step. In the shown architecture, the hidden state h_k at time step k is part of the input in the next time step $k+1$ and the output y_k is generated with an additional layer from the hidden state h_k.

with weight matrix $W = W^{(2)} W^{(1)}$ and bias vector $W^{(2)} b^{(1)} + b^{(2)}$. In classification tasks, it is common to apply the softmax function as final activation function:

$$y_j = \sigma_j(z) = \frac{\exp(z_j)}{\sum_i \exp(z_i)}, \tag{2.9}$$

with input z (the so-called logits), output y, and the in- and output dimensions correspond to the number of considered classes. Contrary to the previously introduced activation functions, the input of the softmax function are pre-activation scores of several neurons instead of a single one. The softmax function enforces normalized $\sum_j y_j = 1$ and positive $y_j > 0$ output values by construction. Consequently, the output values are valid parameters of a categorical distribution and the output components can be identified with class confidences. Finally, the classification can be performed by just taking the class with highest confidence. In regression tasks, it is common practice to apply a final activation function that matches the problem-specific output range. *E.g.* positive outputs can be enforced with a ReLU or an exponential activation function. An unconstrained output can be implemented by using the identity function as activation function.

An NN that processes information in one direction without cycles is called *feedforward NN*. The outputs of neurons are not fed back to neurons of the same or previous layers. The counterpart are *Recurrent Neural Networks (RNNs)* [Rumelhart et al., 1986; Elman, 1990], which allow these cyclic connections.

A simple feedforward NN architecture is visualized in Figure 2.4 b). The first layer represents just the input and is therefore called *input layer*. Then, a flexible number of *hidden layers* can be applied. Each hidden layer computes an intermediate representation by using the output of the previous layer as input. Finally, the *output layer* computes the output of the NN by using the output of the last hidden layer as input.

A three-layer vanilla RNN is visualized in Figure 2.6. RNNs process an input sequence $(x_1, \ldots, x_K)$ of variable length K by using the output or parts of the output of a layer at one time step as input of the same or previous layer in the next time step. Thereby, an internal state is passed from one time step to the next one and serves as memory. The three-layer RNN in Figure 2.6 computes the hidden state h_k and output y_k at time step k according to:

$$\begin{aligned} h_k &= g_h(W^{(xh)}\, x_k + W^{(hh)}\, h_{k-1} + b^{(h)})\ \forall k \in \{1, \ldots, K\}, \\ h_0 &= h_{\text{init}}, \\ y_k &= g_y(W^{(y)}\, h_k + b^{(y)})\ \forall k \in \{1, \ldots, K\}, \end{aligned} \tag{2.10}$$

with g for the activation functions, W for the weight matrices, and b for the bias vectors. If Equation (2.10) is applied on the input of the first time $k = 1$, the hidden state of the previous time step h_0 is set to an initial value, *e.g.* all values are set to zero. In principle, vanilla RNNs are able to consider long-term dependencies in the input sequence. However, in practice gradient-based training suffers from the so-called vanishing and exploding gradient problem [Bengio et al., 1994; Pascanu et al., 2013] (see Section 2.2.2). These training difficulties limit the performance on data with patterns spanning larger time intervals. Hochreiter and Schmidhuber [1997] introduced an RNN architecture, the LSTM, to overcome this problem. LSTMs leverage gates to store information over short- and long-term intervals (see Section 5.3.3). Further common and enhanced RNN architectures are Gated Recurrent Units (GRUs) [Cho et al., 2014] and Bidirectional Recurrent Neural Networks (BRNNs) [Schuster and Paliwal, 1997]. So far, RNNs are explained as models that assign each element of an input sequence exactly one output. However, with small modifications different RNN topologies can be created. Figure 2.7 visualizes an overview of principle topologies.

Another important class of NNs are Convolutional Neural Networks (CNNs), which are characterized as the name suggests by convolutional layers. Yann LeCun is considered a pioneer of CNNs and created with LeNet [LeCun et al., 1998] one of the first modern CNN architectures. Convolutions (compare Equation 2.7) extract the same features for multiple spatial locations and are equivariant *w.r.t.* translations. Thereby, CNNs are particularly suitable for image processing tasks, *e.g.* for semantic image segmentation [Ronneberger et al., 2015]. In classical image processing, convolutional operations are performed with manually engineered kernels, *e.g.* the Sobel filter for edge detection. The key difference of convolutional layers is that the kernel weights are not set by expert knowledge but instead learned from data. In many common CNN architectures like ResNet [He et al., 2016], the number of channels increases and the width and height of the tensors decreases starting from the first layer towards the output layer. The idea behind this approach is to extract an increasing number of features with an increasing level of abstraction and a decreasing spatial resolution. This is beneficial for tasks like image classification that require the extraction of information from the whole image for a semantic decision. In convolutional layers, so-called *pooling* layers and *strides* are common techniques to reduce the spatial dimensions. Pooling layers merge the information of neighboring tensor entries with an operation, *e.g.* taking the maximum or the average. Applying strides greater than one means that the kernel is slid over the input tensor with the specified step size instead of taking each spatial location. For the 2D case, a stride of two would mean that the output tensor is computed for only every second spatial location in x- and y-direction.

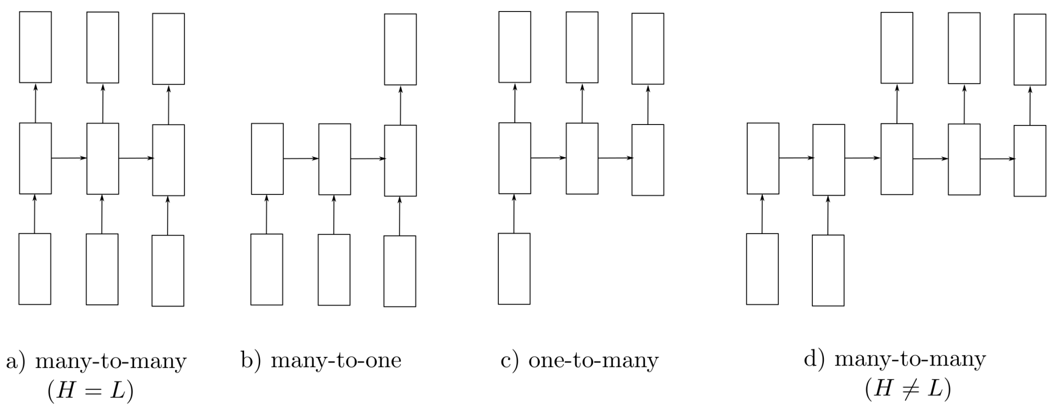


Figure 2.7: Visualization of different RNN topologies. The rectangles in the bottom represent the inputs, the rectangles in the middle the processing by the RNN, the rectangles in the top the outputs, and the arrows visualize the flow of information. a) many-to-many: This is the standard architecture, which assigns each element in the input sequence one output, *e.g.* for a permanent drowsiness detection of a driver from time series. b) many-to-one: A single output is generated from an input sequence. This topology can be obtained by considering only the output of the last time step and removing or discarding the output layers of the previous time steps. *e.g.* for a sentiment classification of sentences. c) one-to-many: An output sequence is generated from a single input. The output sequence can be generated by feeding back the output of the previous time step as input for the next time step *e.g.* for image captioning. d) many-to-many: An output sequence is generated from an input sequence. In comparison to a) the output length L is not restricted to the input length H, *e.g.* for translating sentences into a different language. A many-to-many model is typically implemented with an encoder-decoder architecture, which combines b) and c). This figure is based on slide twelve of Stanford CS231n lecture notes (`http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf`, accessed on 08/03/2022).

2.2.2 Optimization

Supervised Learning

The goal of *supervised learning* is the inference of a model from a set of inputs and associated labeled outputs—the training data set $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$. In this notation, N is the number of in- and output pairs, x_i the ith inputs and y_i the intended output or label for the input x_i. The model is a function $f : \mathcal{X} \rightarrow \mathcal{Y}$ that maps inputs $x \in \mathcal{X}$ to outputs $\hat{y} = f(x) \in \mathcal{Y}$. In the following, models in form of NNs are considered, which are parametrized functions $f_\theta(x)$ with parameters θ. Contrary to Section 2.2.1, the output of the NN is denoted with $\hat{y}$ instead of y to clearly distinguish between the prediction $\hat{y}$ and the label y.

Supervised learning algorithms fit a model $f_\theta(x)$ to the training data set $\mathcal{D}$ by minimizing the errors between the model's predictions $\{\hat{y}_i\}_{i=1}^N$ and the labels $\{y_i\}_{i=1}^N$. For a specific sample of the data set $\mathcal{D}$, the error is evaluated by the *loss function* ℓ, which compares the model's output $\hat{y}_i = f_\theta(x_i)$ and the label y_i. For a regression task, an example for the loss function is the absolute error $\ell(y, \hat{y}) = |y - \hat{y}|$. The expected value of the loss function on the distribution of the data $(x, y) \sim p$ is called *risk* $E_{(x,y)\sim p}[\ell(\hat{y}, y)]$. However, this quantity remains usually unknown and is approximated by the *empirical risk*, which is the mean value of the loss function ℓ applied on all samples of the training data set $\mathcal{D}$:

$$L(\theta) = \frac{1}{N} \sum_{i=1}^{N} \ell(y_i, \hat{y}_i) \text{ with } \hat{y}_i = f_\theta(x_i). \tag{2.11}$$

In the remainder of this thesis, the empirical risk is just called *loss* $L(\theta)$. In comparison, the term *loss function* is used for ℓ. For regression and classification tasks, common losses and the corresponding loss functions are:

- the Mean Squared Error: $L_{\text{MSE}} = \frac{1}{N} \sum_i \ell_{\text{MSE}}(y_i, \hat{y}_i)$ with $\ell_{\text{MSE}}(y, \hat{y}) = (y - \hat{y})^2$ (regression),
- the Mean Absolute Error: $L_{\text{MAE}} = \frac{1}{N} \sum_i \ell_{\text{MAE}}(y_i, \hat{y}_i)$ with $\ell_{\text{MAE}}(y, \hat{y}) = |y - \hat{y}|$ (regression),
- the Cross-Entropy Loss: $L_{\text{CE}} = \frac{1}{N} \sum_i \ell_{\text{CE}}(y_i, \hat{y}_i)$ with $\ell_{\text{CE}}(y, \hat{y}) = -\sum_j y_j \log(\hat{y}_j)$ (classification), and
- the Focal Loss: $L_{\text{FL}} = \frac{1}{N} \sum_i \ell_{\text{FL}}(y_i, \hat{y}_i)$ with $\ell_{\text{FL}}(y, \hat{y}) = -\sum_j y_j (1 - \hat{y}_j)^\gamma \log(\hat{y}_j)$ and $\gamma > 0$ (classification).

For the cross-entropy ℓ_{CE} and focal loss function ℓ_{FL}, the indices j are associated with the possible classes, $\hat{y}$ comprises the predicted class probabilities, and y is the one-hot encoded vector of a sample labeled with class index k, *i.e.* $y_j = \mathbf{1}(j = k)$ ($\mathbf{1}$ is the indicator function). In comparison to the cross-entropy, the focal loss puts more emphasis on difficult samples by multiplying the factor $(1 - \hat{y}_j)^\gamma$ [Lin et al., 2017].

Frequently, the loss is not directly minimized but instead a weighted sum of the loss $L(\theta)$ and a *regularization term* $R(\theta)$:

$$\theta^* = \arg\min_\theta \underbrace{L(\theta) + \lambda R(\theta)}_{J(\theta)}, \tag{2.12}$$

with θ^* being the optimal model parameters, $\lambda > 0$ a weighting factor, and $J(\theta)$ the final *cost* function. The regularization term $R(\theta)$ penalizes *overfitting* and supports the *generalization* capability of the model. As the name suggests, generalization refers to the model's performance for samples out of the training set. Contrary, overfitting describes the phenomenon if a model predicts the outputs of samples within the training set precisely, however, at the expense of the model's generalization capability. Typically, very flexible models with a large number of parameters like DNNs are prone to overfit. An approach to reduce overfitting is the reduction of the model's complexity by penalizing large parameter values, *e.g.* with the following regularization terms:

- the squared Euclidean norm $R(\theta) = \sum_j \theta_j^2$ or
- the Manhattan norm $R(\theta) = \sum_j |\theta_j|$.

Another effective method to reduce overfitting is *dropout* [Srivastava et al., 2014]. During training, neurons are randomly switched off by setting a neuron's output to zero with a certain probability p. A neuron remains active with the probability $1-p$. In the latter case, the output is upscaled with a factor of $1/(1-p)$ to ensure that the output's expectation value is not biased. During inference, dropout is not applied and all neurons remain active. According to a common interpretation, dropout effectively samples many different thinned NN architectures. Thus, predicting without dropout can be considered as an approximation for averaging over this ensemble of different architectures.

Stochastic Gradient Descent

The question remains how to solve the optimization problem in Equation (2.12) in order to find the optimal model parameters θ^*. NNs $f_\theta(x)$ are non-linear functions and have typically a large number of parameters θ. Thus, it becomes quickly infeasible to solve Equation (2.12) exactly. However, efficient supervised learning algorithms have been developed that solve the optimization problem approximately. The most common learning algorithms for NNs are different variants of *gradient descent* methods. These algorithms are of fundamental importance for the success of NNs.

The original gradient descent algorithm solves the minimization problem:

$$\arg\min_\theta J(\theta), \tag{2.13}$$

iteratively by updating the parameters in the direction of the negative gradient of the cost function $-\nabla_\theta J(\theta)$, *i.e.* in the direction with the steepest local decline of J. The cost function J includes the loss L, which is evaluated on the full training data set according to Equation (2.11). The algorithm starts from a random initialization θ_0 and updates the parameters with the following rule:

$$\theta_{i+1} = \theta_i - \eta \nabla_\theta J(\theta_i), \tag{2.14}$$

with *learning rate* $\eta > 0$. The choice of the learning rate is important for the convergence of the algorithm. Values that are too small or large can lead to slow convergence or to overshooting of the minimum. Frequently, the performance is improved by reducing the learning rate during training. This is called learning rate scheduling. The parameter update rule in Equation (2.14) is applied until a *stopping criterium* is reached. In the beginning, the loss decreases rapidly on the training set and on independent data. Independent but identically distributed data is important to evaluate the generalization capability of the NN and typically given by a separate *validation data set*. After a certain number of iterations, the loss reaches a minimum on the validation set while the loss on the training data continues to decrease. At this point of time, overfitting begins: The error of the NN decreases on the training data at the expense of the NN's generalization capability. The procedure of monitoring the loss on the validation set and stopping the training already when the loss on the validation set no longer decreases is called *early stopping* [Bishop, 2006, p.259].

Stochastic Gradient Descent (SGD) can be considered as a stochastic approximation of gradient descent. Instead of computing the gradient of the cost function exactly, the gradient is approximated from only a single sample or a subset of randomly drawn samples, the so-called mini-batch:

$$\nabla J(\theta) \approx \nabla L_{\text{batch}}(\theta) + \lambda \nabla R(\theta) \text{ with} \tag{2.15}$$

$$L_{\text{batch}} = \frac{1}{|I_{\text{batch}}|} \sum_{i \in I_{\text{batch}}} \ell(y_i, \hat{y}_i) \approx L(\theta), \tag{2.16}$$

with I_{batch} the indices of the mini-batch. For the training of NNs, the number of samples in the mini-batch is treated usually as a hyperparameter, *e.g.* $|I_{\text{batch}}| = 8, 16, 32, \dots, 512$. SGD is computationally less expensive and requires less memory than gradient descent because the gradient of the loss function must be computed only on a subset of the full data set. Furthermore, the noise in SGD is known to have a positive effect on convergence [Mertikopoulos et al., 2020] and generalization [Smith et al., 2020].

Several optimizers modify the parameter update rule in Equation (2.14) to address drawbacks of standard gradient descent. The standard update rule is prone to get stuck in local minima or saddle points, ravines in the loss landscape lead to oscillations, and applying the same learning rate to parameters independently of their impact on the cost function is suboptimal. A method, which is called *momentum*, addresses problems associated with vanishing gradients at local minima or at saddle points and the challenge with oscillations in ravines. Momentum takes the gradients of previous time steps into account by computing the moving exponential average:

$$\begin{aligned} \theta_{i+1} &= \theta_i - \eta m_i \text{ with} \\ m_i &= \beta_1 m_{i-1} + (1 - \beta_1) \nabla_\theta J(\theta_i), \end{aligned} \tag{2.17}$$

with $0 \le \beta_1 < 1$ the factor for the exponential decay, *e.g.* $\beta_1 = 0.9$. Computing the average gradient over subsequent time steps cancels oscillating components of the gradient out and accumulates components in the same direction. Thereby, momentum speeds up convergence significantly. Furthermore, it is beneficial to apply individual learning rates for each of the parameters. Parameters with a frequent and high impact on the cost function should be updated with a smaller learning rate than parameters with low impact on the cost function. This can be achieved by rescaling the learning rate of a parameter individually with the square root of the exponential average of the squared partial derivation of the cost function *w.r.t.* this parameter:

$$\begin{aligned} \theta_{i+1,j} &= \theta_{i,j} - \frac{\eta}{\sqrt{v_{i,j}} + \epsilon} \frac{\partial J}{\partial \theta_j}(\theta_i) \text{ with} \\ v_{i,j} &= \beta_2 v_{i-1,j} + (1 - \beta_2) \left(\frac{\partial J}{\partial \theta_j}(\theta_i) \right)^2, \end{aligned} \tag{2.18}$$

with $0 \le \beta_2 < 1$ the factor for the exponential decay, *e.g.* $\beta_2 = 0.999$, and $\epsilon > 0$ a small number for numerical stability. In the formula, the index i is used to refer to a parameter at a certain step and the index j is used to point on an individual parameter of the parameter vector θ. This adaptive learning rate schedule is called *Root Mean Squared propagation (RMSprop)* [Tieleman and Hinton, 2012]. The optimizer *Adaptive moment estimation (Adam)* [Kingma and Ba, 2014] combines momentum and RMSprop in one parameter update rule:

$$\begin{aligned} \theta_{i+1,j} &= \theta_{i,j} - \frac{\eta}{\sqrt{\hat{v}_{i,j}} + \epsilon} \hat{m}_{i,j} \text{ with} \\ \hat{m}_{i,j} &= m_{i,j} / (1 - (\beta_1)^i), \\ \hat{v}_{i,j} &= v_{i,j} / (1 - (\beta_2)^i). \end{aligned} \tag{2.19}$$

The exponential moving average of the gradient and of the squared gradient are initialized with zeros, *i.e.* $m_0 = 0$ and $v_0 = 0$. For unbiased estimates of the first moment $\hat{m}_i$ and of the second raw moment $\hat{v}_i$ of the gradient, m_i and v_i are divided by the correction term $1 - (\beta_{1/2})^i$. This explains the name Adam as abbreviation derived from Adaptive moment estimation. In many experiments, Adam has proven to be state-of-the-art and is a frequently applied optimizer.

Independently of the used optimizer, the core of the learning rule is the computation of the cost function's gradient *w.r.t.* the model's parameters $\nabla_\theta J(\theta)$. *Backpropagation* is an algorithm to perform the required automatic differentiation (see Figure 2.8). The cost function $J(\theta)$ is considered to be a nested function, which can be represented by a Directed Acyclic Graph (DAG) with several roots and a single leaf. The roots represent the parameters θ and the leaf is associated with the cost J. Except the roots, each node j processes inputs given by the parent nodes to an output z_j by applying a differentiable function. The output z_j of each of the roots is defined as the parameter θ_j that it represents. From this graph-based representation of the cost function J, the automatic differentiation $\nabla_\theta J(\theta)$ is performed in two steps: First, in the *forward pass* a topological order is defined on the DAG and the node's outputs $\{z_j\}_j$ are determined in the defined order. The order ensures that the required inputs, which are the outputs of the parent nodes, have already been computed in a previous step. Finally, the cost is obtained when the leaf of the DAG is reached. Second, in the *backward pass* the partial derivations of the cost function *w.r.t.* a node's output $\tilde{z}_j = \frac{\partial J}{\partial z_j}$ are propagated in reverse order towards $\frac{\partial J}{\partial \theta_j}$ at the roots. The partial derivations $\tilde{z}_j$ are computed according to the chain rule:

$$\tilde{z}_j = \frac{\partial J}{\partial z_j} = \sum_k \frac{\partial J}{\partial z_k} \frac{\partial z_k}{\partial z_j} = \sum_k \tilde{z}_k \frac{\partial z_k}{\partial z_j}, \tag{2.20}$$

with k being child nodes of node j. $\tilde{z}_j$ depends on $\left\{\frac{\partial z_k}{\partial z_j}\right\}_k$, which are easy to determine, and on $\{\tilde{z}_k\}_k$ of the child nodes, which are known due to the execution in reverse order. Furthermore, the results $\{z_j\}_j$ of the forward pass are required as arguments for evaluating the partial derivations $\left\{\frac{\partial z_k}{\partial z_j}\right\}_k$. Finally, the backward pass terminates with $\left\{\frac{\partial J}{\partial \theta_j}\right\}_j$ at the roots, which are the components of the gradient $\nabla_\theta J(\theta)$.

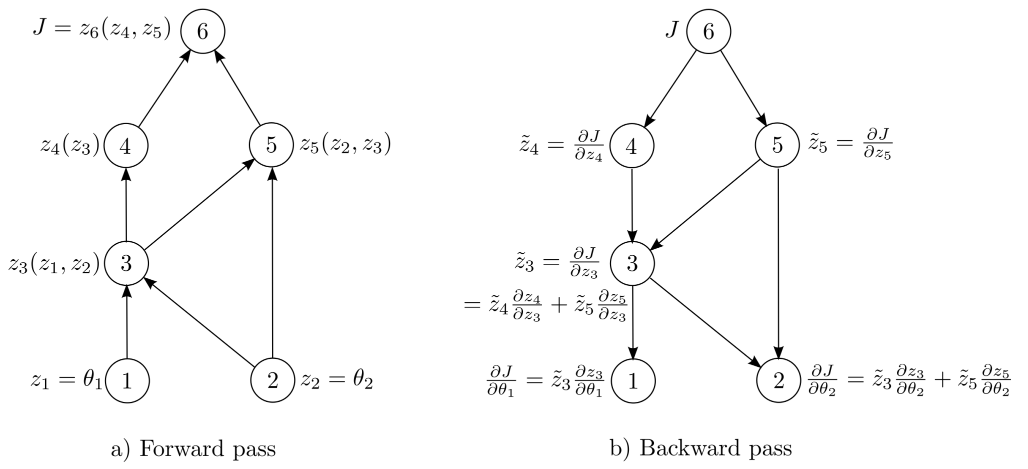

$J = z_6(z_4, z_5)$
$z_4(z_3)$
$z_5(z_2, z_3)$
$z_3(z_1, z_2)$
$z_1 = \theta_1$
$z_2 = \theta_2$
a) Forward pass
J
$\tilde{z}_4 = \frac{\partial J}{\partial z_4}$
$\tilde{z}_5 = \frac{\partial J}{\partial z_5}$
$\tilde{z}_3 = \frac{\partial J}{\partial z_3} = \tilde{z}_4 \frac{\partial z_4}{\partial z_3} + \tilde{z}_5 \frac{\partial z_5}{\partial z_3}$
$\frac{\partial J}{\partial \theta_1} = \tilde{z}_3 \frac{\partial z_3}{\partial \theta_1}$
$\frac{\partial J}{\partial \theta_2} = \tilde{z}_3 \frac{\partial z_3}{\partial \theta_2} + \tilde{z}_5 \frac{\partial z_5}{\partial \theta_2}$
b) Backward pass

Figure 2.8: In the backpropagation algorithm, the cost function $J(\theta)$ is considered as a nested function, which can be represented with a DAG. The roots of the DAG are identified with the parameters θ, the leaf represents the assigned cost J, and the intermediate nodes are associated with the nested functions z_j. From this graph representation, the gradient of the cost function $\nabla_\theta J(\theta)$ is computed in two steps: a) In the forward pass, the cost J is determined by evaluating the nested functions z_j in topological order from the roots towards the leaf. b) In the backward pass, the partial derivations of the cost function *w.r.t.* the outputs of the nested functions $\tilde{z}_j = \frac{\partial J}{\partial z_j}$ are propagated in reverse order from the leaf towards the roots. The results of the forward pass $\{z_j\}_j$ are required as arguments for evaluating the partial derivations. *E.g.* for $\tilde{z}_4 = \frac{\partial J}{\partial z_4}(z_4, z_5)$, the arguments z_4 and z_5 are known from the forward pass. The backward pass terminates at the leaves with the partial derivations of the cost function *w.r.t.* the parameters $\frac{\partial J}{\partial \theta_j}$.

2.2.3 Bayesian Deep Learning

A more general interpretation of the training and the prediction of NNs is obtained from a Bayesian perspective. Instead of a single prediction, in Bayesian Deep Learning (BDL) [Gal, 2016] a whole distribution over outputs is modeled, which enables the quantification of uncertainty. In general, two types of uncertainty are distinguished. *Epistemic* uncertainty is caused by the uncertainty of the model's parameters and can be reduced by increasing the training data. In comparison, *aleatoric* uncertainty represents the intrinsic uncertainty, which remains even in the limit of infinite data. In safety critical applications, the quantification of uncertainty is an important capability towards self-awareness. However, the training of well-calibrated models is challenging and an active field of research.

In BDL, the data $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$ is assumed to be independent and identically distributed and the model's parameters θ are treated as random variables with the prior distribution $p(\theta)$:

$$p(\mathcal{D}, \theta) = \prod_{i=1}^{N} p(y_i|x_i, \theta) p(x_i) p(\theta). \tag{2.21}$$

Typically, $p(y|x, \theta)$ is modeled as a distribution $p(y|\lambda = f_\theta(x))$ with parameters λ and the distribution's parameters are predicted by the model $f_\theta(x)$. *E.g.* if the output y is assumed to be Gaussian distributed, the model would predict the mean μ and variance σ^2, which are summarized in the parameter $\lambda = (\mu, \sigma^2)$. In DL, the model $f_\theta(x)$ is an NN. For given data, the distribution over the model parameters is updated and given by the posterior:

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}, \theta)}{p(\mathcal{D})} = \frac{\prod_{i=1}^{N} p(y_i|x_i, \theta) p(x_i) p(\theta)}{\int \prod_{i=1}^{N} p(y_i|x_i, \theta) p(x_i) p(\theta) d\theta} \propto \prod_{i=1}^{N} p(y_i|x_i, \theta) p(\theta). \tag{2.22}$$

However, for NNs the integration over the parameters θ to obtain the evidence $p(\mathcal{D})$ is intractable. Instead, approximative methods exist to estimate the posterior $p(\theta|\mathcal{D})$, *e.g.* variational inference with dropout [Gal and Ghahramani, 2016]. Finally, the posterior captures the relevant information to compute the output distribution for a new input x:

$$p(y|x, \mathcal{D}) = \int p(y, \theta|x, \mathcal{D}) d\theta = \int \frac{p(x, y, \mathcal{D}, \theta)}{p(x, \mathcal{D})} d\theta = \int p(y|x, \theta) p(\theta|\mathcal{D}) d\theta. \tag{2.23}$$

In Gal and Ghahramani [2016], the mean and the variance of the predictive distribution $p(y|x, \mathcal{D})$ are estimated with Monte Carlo integration. It is shown that the required samples can be generated with dropout during inference.

The conventional training of NNs, as described in Section 2.2.2, optimizes for a single best model parameter θ^* by minimizing a cost function $J(\theta)$ (see Equation 2.12). In the probabilistic framework, the minimization of the cost function can be interpreted as a Maximum A Posteriori (MAP) approach or a Maximum Likelihood Estimation (MLE). Instead of taking the full posterior $p(\theta|\mathcal{D})$ into account as in Equation (2.23), in the MAP approach only the parameters that maximize the posterior are considered:

$$\begin{aligned} \theta^* &= \arg\max_\theta p(\theta|\mathcal{D}) = \arg\min_\theta \left(-\log p(\theta|\mathcal{D})\right) \qquad (2.24) \\ &\overset{(2.22)}{=} \arg\min_\theta \left(-\sum_{i=1}^{N} \log \underbrace{p(y_i|x_i, \theta))}_{=p(y_i|\lambda = f_\theta(x_i))} - \log p(\theta) \right). \end{aligned}$$

If the term $-\log p(\theta)$ is dropped, the MAP approach turns into the MLE. The cost function $J(\theta)$ in Equation (2.12) is obtained (up to a scaling factor) by identifying $-\log p(y_i|\lambda = f_\theta(x_i))$ with the loss function ℓ and $-\log p(\theta)$ with the regularization term $R(\theta)$. The loss associated with $-\log p(y_i|\lambda = f_\theta(x_i))$ is also called *Negative Log-Likelihood (NLL)* loss. *E.g.* the Mean Squared Error (MSE) loss function ℓ_{MSE} results from a Gaussian distribution $p(y|\lambda = f_\theta(x)) = \mathcal{N}(y|\mu = f_\theta(x), \sigma^2)$ with input-dependent mean μ and constant isotropic variance σ^2 (*homoscedastic case*). In the *heteroscedastic case*, additionally an input-dependent covariance matrix Σ is assumed, *i.e.*

$p(y|\lambda = f_\theta(x)) = \mathcal{N}(y|(\mu, \Sigma) = f_\theta(x))$. A 2D Gaussian with mean values, variances, and correlation coefficient $\lambda = (\mu_x, \mu_y, \sigma_x, \sigma_y, \rho)^\intercal$ has the following NLL loss:

$$\ell_{\text{NLL}}(y, \lambda) = \log\left(2\pi\sigma_x\sigma_y\sqrt{1-\rho^2}\right) + \frac{1}{2(1-\rho^2)}\left(\frac{(y_x-\mu_x)^2}{\sigma_x^2} + \frac{(y_y-\mu_y)^2}{\sigma_y^2} - \frac{2\rho(y_x-\mu_x)(y_y-\mu_y)}{\sigma_x\sigma_y}\right). \tag{2.25}$$

In classification, it is reasonable to assume a categorial distribution for $p(y|\lambda = f_\theta(x))$ and the negative logarithm turns out to be the cross-entropy loss function ℓ_{CE}. Further loss functions can be derived analogously. Similarly, the regularization term $R(\theta)$ follows from the negative logarithm of the prior $p(\theta)$. The squared Euclidean norm $R(\theta) = \sum_j \theta_j^2$ is obtained if a Gaussian prior $p(\theta)$ with zero mean and isotropic variance is assumed. Analogously, the Manhattan norm $R(\theta) = \sum_j |\theta_j|$ is related to a Laplacian prior $p(\theta)$.

A general modeling of $p(y|\lambda = f_\theta(x))$ can be achieved by using mixture models [Bishop, 1994]. Mixture models are able to capture multimodal data and can approximate any density function with arbitrary accuracy if a large enough number of components is given [McLachlan and Basford, 1988]. Bishop [1994] focuses on Gaussian mixture models with m components:

$$p(y|\lambda = f_\theta(x)) = \sum_{i=1}^{m} \alpha_i(x)\mathcal{N}(y|\mu_i(x), \sigma_i^2(x)). \tag{2.26}$$

The parameters of the mixture model $\lambda = (\alpha_1, \ldots, \alpha_m, {\sigma_1}^\intercal, \ldots, {\sigma_m}^\intercal, {\mu_1}^\intercal \ldots, {\mu_m}^\intercal)^\intercal$ comprise the mixing coefficient α_i, the variance σ_i^2, and the mean value μ_i of each component. An NN is a flexible function approximator and thereby suitable to predict the parameters $\lambda = f_\theta(x)$. In this setting, the NN is called *Mixture Density Network (MDN)* [Bishop, 1994]. The MDN can be trained with stochastic gradient descent and the negative logarithm of Equation (2.26) as loss function.

2.3 Reinforcement Learning

This chapter summarizes concepts of RL and the book *Reinforcement Learning: An Introduction* [Sutton and Barto, 2018] has served as main source. For details and derivations, it is referred to the great original work.

2.3.1 Basics and Intuition

A fundamental principle of human intelligence is the learning by interaction with the environment. *E.g.* learning motor skills for table tennis requires many trials and failures. Instead of learning from pure supervision, different actions must be explored and experience is gained by observing short- and long-term consequences. In the beginning of the learning process, the responses of actions are pre-dominantly unknown and a typical strategy is some kind of random exploration. In the example of table tennis, probably

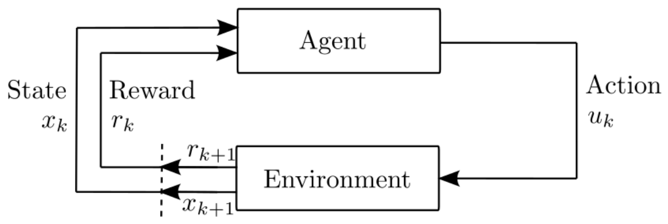


Figure 2.9: In RL, an agent interacts with the environment by choosing an action u_k according to its policy $u_k \sim \pi(u_k|x_k)$ and depending on the state x_k at time step k. For the current state x_k and action u_k, the environment samples the state x_{k+1} of the next time step $k+1$ according to a transition probability distribution $x_{k+1} \sim p(x_{k+1}|u_k, x_k)$ and determines the associated reward $r(x_k, u_k, x_{k+1})$. The figure has been adapted from Figure 3.1 in Sutton and Barto [2018, p.54].

the first trials look clumsy. Over time, the interactions become more goal-oriented and promising behavior is optimized. However, there still might be even better but unknown behavior patterns and exploration should be continued to a certain extend. *E.g.* a good table tennis player should continue to learn new strokes.

In RL, the setting of learning from interaction is formalized in an idealized process with discretized time steps as visualized in Figure 2.9. In each time step, an *agent* observes the state of the *environment* and chooses an *action* according to its *policy*. The policy can be deterministic or stochastic and the space of possible actions discrete or continuous. The agent's action results in a new state and the agent receives a real valued *reward*. Both, the transition of the environment and the reward function, may be probabilistic or deterministic. Finally, the objective is the optimization of the agent's policy to maximize the *cumulative and discounted reward*. Mathematically, the problem is described by a *Markov Decision Process (MDP)* and RL studies methods to approximate the optimal policy of the MDP. Analogously to humans, RL algorithms balance between exploring unknown actions and optimizing promising behavior. This is known as *exploration-exploitation trade-off*.

The output of the policy, the agent's action, influences the future states and the future states are fed back as inputs to the policy. This is analogously to closed-loop systems in control theory and RL can be leveraged to learn control laws of discrete-time stochastic control processes. If RL is applied on a control problem, the controller can be considered as the agent, the plant describes the dynamic of the environment, and the reward function defines the optimal behavior.

The remainder of this chapter is structured as follows. In Section 2.3.2, the objective of RL is formalized by defining the MDP mathematically. In Section 2.3.3, general concepts of RL algorithms are presented to solve MDPs. Many RL algorithms are based on the greedy learning scheme *general policy iteration* and leverage *value functions* to estimate the quality of policies. In Section 2.3.4, common classifications of RL algorithms are presented.

2.3.2 Markov Decision Process

The picture of an agent interacting with an environment can be formalized with an MDP. An MDP describes the dynamics with a sequence $\mathcal{K} = \{0, 1, \dots\}$ of random variables $\tau = \{X_k, U_k, R_{k+1}\}_{k\in\mathcal{K}}$, with X_k for the state of the environment, U_k for the action of the agent, and R_{k+1} for the received reward. The sequence τ is also called roll-out or trajectory and follows the joint probability distribution[2]:

$$P(X_0, U_0, R_1, X_1, U_1, R_2, \dots). \tag{2.27}$$

The Markov in MDP stands for the *Markov property* and means that the next state X_{k+1} and reward R_{k+1} depends only on the current state X_k and action U_k:

$$P(R_{k+1}, X_{k+1}|X_k, U_k) = P(R_{k+1}, X_{k+1}|X_0, U_0, R_1, X_1, U_1, \dots, R_k, X_k, U_k)\ \forall k \in \mathcal{K}. \tag{2.28}$$

In other words, the future states are independent of any states and actions of the history before the current time step. Thus, the joint probability distribution can be written as follows:

$$\begin{aligned}
&P(X_0, U_0, R_1, X_1, U_1, R_2, \dots) \\
=&P(X_0)P(U_0|X_0)P(R_1, X_1|X_0, U_0)P(U_1|X_0, U_0, R_1, X_1) \\
&P(R_2, X_2|X_0, U_0, R_1, X_1, U_1) \cdots, \\
\overset{(2.28)}{=}&P(X_0)P(U_0|X_0)P(R_1, X_1|X_0, U_0)P(U_1|X_0, U_0, R_1, X_1)P(R_2, X_2|X_1, U_1) \cdots,
\end{aligned} \tag{2.29}$$

with $P(R_{k+1}, X_{k+1}|X_k, U_k) = P(X_{k+1}|X_k, U_k)P(R_{k+1}|X_k, U_k, X_{k+1})$. In the following, standard MDPs with static state transition models are assumed:

$$p(x'|x, u) = P(X_{k+1} = x'|X_k = x, U_k = u)\ \forall k \in \mathcal{K}. \tag{2.30}$$

Contrary, in *time varying MDPs* [Liu and Sukhatme, 2018] the transition probability $P(X_{k+1} = x'|X_k = x, U_k = u)$ is not identical for different time steps k and the state transition model $p_k(x'|x, u)$ depends on k.

In general and according to Equation (2.29), a policy is defined as a sequence $\pi = \{\pi_k\}_{k\in\mathcal{K}}$ with π_k being the conditional probability distributions $P(U_k|X_0, U_0, R_1, \dots, U_{k-1}, R_k, X_k)$. The overall goal is the maximization of the *expected discounted return* by optimizing the policy π. The expected discounted return is defined as:

$$\begin{aligned}
J^\pi &= E\big[G_0\big] = E\left[\sum_{k\in\mathcal{K}} \gamma^k R_{k+1}\right], \\
G_k &= \sum_{k'\in\mathcal{K}} \gamma^{k'} R_{k+k'+1},
\end{aligned} \tag{2.31}$$

[2] For compactness, the extensive notation $P(X_0 = x_0, U_0 = u_0, R_1 = r_1, \dots)$ is used only if necessary.

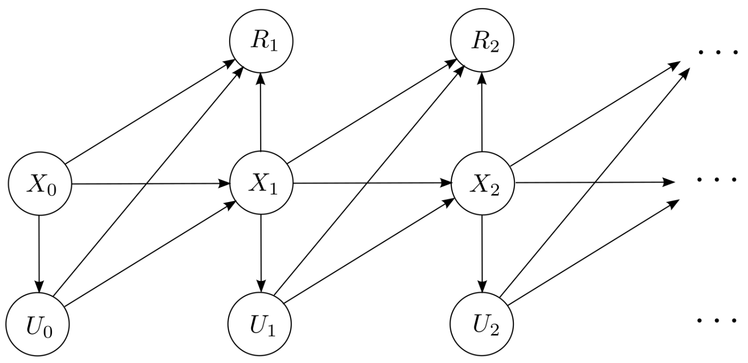


Figure 2.10: An MDP describes the states, actions, and rewards with a sequence of random variables $\{X_k, U_k, R_{k+1}\}_{k\in\mathcal{K}}$, X_k for the state, U_k for the action, and R_{k+1} for the reward at time step k or $k+1$, respectively. The graphical model shows the conditional dependencies between the random variables of an MDP with a Markovian policy $\pi(u|x)=P(U_k=u|X_k=x)$.

with G_k the *cumulative reward* at time step k and $\gamma\in[0,1]$ the discount factor. The discount factor controls the weighting of short- and long-term goals. An optimal policy is denoted with $\pi^*=\arg\max_\pi J^\pi$. It can be shown that an optimal policy π^* of standard MDPs with infinite time horizon ($\mathcal{K}=\mathbb{N}_0$) is within the class of stationary Markovian policies:

$$\text{Markovian policy: } P(U_k|X_k) = P(U_k|X_0, U_0, R_1, \dots, U_{k-1}, R_k, X_k)\ \forall k\in\mathcal{K}, \tag{2.32}$$
$$\text{stationary Markovian policy: } \pi(u|x) = P(U_k=u|X_k=x)\ \forall k\in\mathcal{K}. \tag{2.33}$$

Furthermore, at least one deterministic policy $\pi:\mathcal{X}\to\mathcal{U}$ with $P(U_k=u|X_k=x)=\delta(u-\pi(x))$ exists under the optimal policies [Puterman, 1994, Theorem 6.2.7]. For the remainder of this chapter, MDPs with infinite time horizon are assumed.

The conditional dependencies between the random variables $\{X_k, U_k, R_{k+1}\}_{k\in\mathcal{K}}$ are formalized in Equations (2.29) and (2.32) and visualized in Figure 2.10 with a graphical model.

To conclude, an MDP is specified by the tuple $(\mathcal{X},\mathcal{U},p,r,\gamma)$, with

- $\mathcal{X}$ the state space, *i.e.* the possible values of the random variables $\{X_k\}_{k\in\mathcal{K}}$,
- $\mathcal{U}$ the action space, *i.e.* the possible values of the random variables $\{U_k\}_{k\in\mathcal{K}}$,
- $p(x'|x,u)=P(X_{k+1}=x'|X_k=x, U_k=u)\ \forall k\in\mathcal{K}$ the static state transition model,
- $r(x,u,x')=E[R_{k+1}|X_k=x, U_k=u, X_{k+1}=x']\ \forall k\in\mathcal{K}$ the reward function, and
- $\gamma\in[0,1]$ the discount factor of the return.

The initial distribution $p_0(x_0)=P(X_0=x_0)$ over the states is not relevant because optimal policies maximize the expected discounted return independently of this distribution. In a *partially observable MDP*, states are not directly observable and additional random variables for state observations are introduced. In this thesis, fully observable MDPs are considered and in the next section general methods to solve them are presented.

2.3.3 General Concepts for Solving Markov Decision Processes

Value functions and Bellman Equations

In RL algorithms, value functions are used to evaluate the quality of a policy given a state or state-action pair. The *state-value function* is defined as the expected discounted return when starting from a state x at any time step k and then following the policy $\pi(u|x)$:

$$V_k^\pi(x) = E\left[G_k | X_k = x\right] = E\left[\sum_{k' \in \mathcal{K}} \gamma^{k'} R_{k+k'+1} | X_k = x\right]. \tag{2.34}$$

The *action-value function* is defined analogously, except that the first action after starting from state x is not sampled from the policy but set to a given action u:

$$Q_k^\pi(x,u) = E\left[G_k | X_k = x, U_k = u\right] = E\left[\sum_{k' \in \mathcal{K}} \gamma^{k'} R_{k+k'+1} | X_k = x, U_k = u\right]. \tag{2.35}$$

For the considered infinite time horizon $\mathcal{K} = \mathbb{N}_0$, the value functions are independent of k, *i.e.* $V^\pi(x) = V_k^\pi(x)$ and $Q^\pi(x,u) = Q_k^\pi(x,u)$. The state-value and action-value function can be expressed by each other:

$$V^\pi(x) = \int Q^\pi(x,u)\pi(u|x)du, \tag{2.36}$$

$$Q^\pi(x,u) = \int \big(r(x,u,x') + \gamma V^\pi(x')\big)p(x'|x,u)dx'. \tag{2.37}$$

Furthermore, the value functions can be expressed through themselves by substituting the above equations in each other. Then, the *Bellman equations* are obtained, which describe an important recursive relationships of the state-value and action-value functions:

$$\begin{aligned} V^\pi(x) &= E\left[R_{k+1} + \gamma V^\pi(X_{k+1}) | X_k = x\right] && (2.38)\\ &= \int \pi(u|x) \int p(x'|x,u)\big(r(x,u,x') + \gamma V^\pi(x')\big)dx'du, \\ Q^\pi(x,u) &= E\left[R_{k+1} + \gamma Q^\pi(X_{k+1}, U_{k+1}) | X_k = x, U_k = u\right] && (2.39)\\ &= \int p(x'|x,u) \int \pi(u'|x')\big(r(x,u,x') + \gamma Q^\pi(x',u')\big)du'dx'. \end{aligned}$$

Thus, the equations express the value functions as the expectation value of the sum of the reward and the value functions at the next time step.

An optimal policy π^* is defined as a policy that maximizes the expected return for all states:

$$V^*(x) = V^{\pi^*}(x) \geq V^\pi(x)\ x \in \mathcal{X}, \forall \pi, \tag{2.40}$$

with V^* the corresponding optimal state-value functions. It can be shown that at least one optimal policy exists that is better than or equal as good as all other policies for all states. An optimal policy according to (2.40) maximizes also the expected return in Equation (2.31) independently of the distribution over states at start $p(x_0)$:

$$J^\pi = \int V^\pi(x_0)p(x_0)dx_0 \leq \int V^*(x_0)p(x_0)dx_0 = J^{\pi^*} = J^* \;\forall\pi, \tag{2.41}$$

with J^* the notation for the maximum expected return. Analogously, Equation (2.37) and Equation (2.40) imply that an optimal policy maximizes the action-value function:

$$\begin{aligned} Q^\pi(x,u) &= \int \big(r(x,u,x') + \gamma V^\pi(x')\big)p(x'|x,u)dx' \qquad (2.42)\\ &\leq \int \big(r(x,u,x') + \gamma V^*(x')\big)p(x'|x,u)dx' \\ &= Q^{\pi^*}(x,u) = Q^*(x,u)\; x\in\mathcal{X}, u\in\mathcal{U}, \forall\pi, \end{aligned}$$

with Q^* the notation for the optimal action-value function. The optimal state-value function $V^*(x)$ and the optimal action-value function $Q^*(x,u)$ are related via:

$$V^*(x) = \max_{u\in\mathcal{U}} Q^*(x,u). \tag{2.43}$$

The right-hand side of the above equation is obviously the value function $V^{(\pi,\pi^*)}$ of a policy with $\pi(x) = \arg\max_u Q^*(x,u)$ in the current time step and then following π^*. According to Equation (2.36), the value function of an optimal policy is maximally equal as good as $V^{(\pi,\pi^*)}$:

$$V^*(x) \overset{(2.36)}{=} \int Q^*(x,u)\pi^*(u|x)du \leq \max_{u\in\mathcal{U}} Q^*(x,u) = V^{(\pi,\pi^*)}(x) \;\forall\pi. \tag{2.44}$$

Finally, the less or equal must be an equal because of the definition of an optimal policy in Equation (2.40).

The *Bellman optimality equations* describe a recursive relationship of the optimal state-value and action-value function and are obtained by applying equations (2.37) and (2.43) in the order specified above the equality symbol:

$$V^*(x) \overset{(2.43),(2.37)}{=} \max_{u\in\mathcal{U}} \int \big(r(x,u,x') + \gamma V^*(x')\big)p(x'|x,u)dx', \tag{2.45}$$

$$Q^*(x,u) \overset{(2.37),(2.43)}{=} \int \big(r(x,u,x') + \gamma \max_{u'\in\mathcal{U}} Q^*(x',u')\big)p(x'|x,u)dx'. \tag{2.46}$$

In comparison to the general Bellman equations in (2.38) and (2.39), the Bellman optimality equations are more compact and the value functions depend only on themselves and not on the policy. For finite state and action spaces, the Bellman optimality equations have unique solutions. Given this solution, an optimal policy is easily obtained, *e.g.* by $\pi^*(x) = \arg\max_u Q^*(x,u)$. However, typically the environments are complex, the dynamics $p(x'|x,u)$ unknown, and an optimal value function can not computed exactly or computational and memory costs become infeasibly large. Therefore, in the following methods for finding approximate solutions are presented.

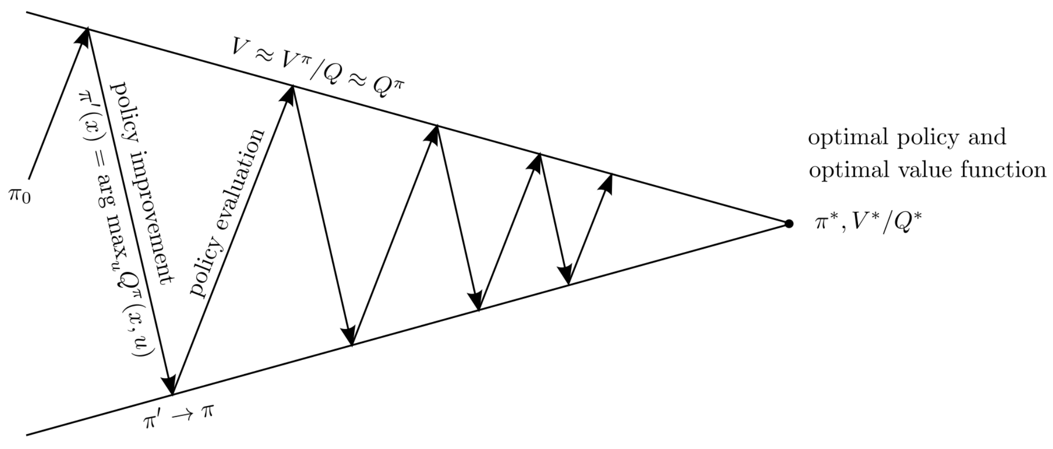


Figure 2.11: General policy iteration alternates between improving the policy greedily, the policy improvement step, and approximating the value function of the improved policy, the policy evaluation step. Typically, the policy evaluation is an iterative algorithm and the quality depends on the number of iterations. Instead of waiting for convergence, frequently policy evaluation involves only few iterations or a single iteration before the policy is updated again. If the policy improvement step no longer changes the policy and enough iterations are performed so that the approximated value function converged to the true value function, an optimal policy and an optimal value function is found. The figure has been adapted from Figure 4.7 in Sutton and Barto [2018, p.105].

General Policy Iteration

Many RL algorithms are based on a greedy learning scheme, which is called *general policy iteration* [Sutton and Barto, 2018] and visualized in Figure 2.11. In an alternating process, the policy is updated to improve the average return, then the improved policy is evaluated to update the estimation of the average return. The former is called *policy improvement* and the latter *policy evaluation*. Typically, policy evaluation requires that the agent interacts with the environment. An appropriate trade-off between exploration of unknown domains and exploitation of promising actions is crucial. In the long run and under certain conditions, general policy iteration converges to the optimal policy.

Policy improvement. Given the value function V^π/Q^π of a policy π, the following *greedy policy* π' is at least as good as π:

$$\begin{aligned} \pi'(x) &= \arg\max_{u\in\mathcal{U}} Q^\pi(x,u) \\ &\overset{(2.37)}{=} \arg\max_{u\in\mathcal{U}} \int \big(r(x,u,x') + \gamma V^\pi(x')\big) p(x'|x,u)dx'. \end{aligned} \tag{2.47}$$

This is a result of the *policy improvement theorem* and shows the importance of approximating value functions with policy evaluation. If it turns out that the greedy policy is unchanged, an optimal policy $\pi^* = \pi' = \pi$ is found. This can be seen by using Equation (2.47) with $\pi' = \pi$ in the Bellman Equations (2.38) and (2.39) to obtain the Bellman optimality equations.

Theorem 1 (Policy improvement theorem)**.** *Let π and π' be two policies so that the average return of π can be improved by using the policy π' in the current time step and then following π:*

$$\int Q^{\pi}(x,u)\pi'(u|x)du \geq V^{\pi}(x)\ \forall x \in \mathcal{X}. \tag{2.48}$$

Then, using the changed policy π' in all time steps yields at least the average return of π:

$$V^{\pi'}(x) \geq V^{\pi}(x)\ \forall x \in \mathcal{X}. \tag{2.49}$$

If there exists a state in Equation (2.48) *that inequality holds, the state-value function of π' is greater than the state-value function of π for at least this state.*

The idea in the proof of the policy improvement theorem is an iterative expansion of $Q^{\pi}(x,u)$ in Equation (2.48) with Equation (2.37) and a repeated application of Equation (2.48) [Sutton and Barto, 2018, p.95]. It becomes clear that the greedy policy π' in Equation (2.47) fulfills Equation (2.48) by writing the right-hand side in Equation (2.48) according to Equation (2.36).

Equation (2.47) shows the construction of an improved deterministic policy over a given policy. A similar statement exists for the construction of an improved stochastic policy. Given a value function Q^{π} of an ϵ-soft policy π (π fulfills $\pi(u|x) \geq \epsilon/|\mathcal{U}|\ \forall x \in \mathcal{X}$ with $|\mathcal{U}| = \int_{\mathcal{U}} du$), the following so-called *ϵ-greedy policy* π'_{ϵ} is at least as good as π:

$$u_{\epsilon} \sim \pi'_{\epsilon}(u|x) : \begin{cases} u_{\epsilon} = \arg\max_{u \in \mathcal{U}} Q^{\pi}(x,u) & \text{with probability } 1-\epsilon \\ u_{\epsilon} \sim p(U) = 1/|\mathcal{U}| & \text{with probability } \epsilon \end{cases} \tag{2.50}$$

The ϵ-greedy policy chooses a random action with a probability of ϵ and otherwise the action of the greedy policy *w.r.t.* π. It can be shown that the ϵ-greedy policy fulfills Equation (2.48) under the assumption that π is a ϵ-soft policy. This implies that the ϵ-greedy policy *w.r.t.* an ϵ-soft policy π improves π. As shown later, many RL algorithms approximate the value function only roughly in the policy evaluation step. Frequently, the value function is updated only locally at a certain state or state-action pair and the next state or state-action pair for the value function update is determined by the so-called *behavior policy*. For convergence, these algorithms typically require that the behavior policy explores the complete state-action space. This can be ensured by using the ϵ-greedy policy. If the behavior policy is also the policy that is optimized (*on-policy method*), the policy improvement theorem ensures that the ϵ-greedy policy improves in fact any previous ϵ-greedy policy. The value of ϵ determines the exploration-exploitation trade-off. Higher values of ϵ favor random actions (exploration) and lower values favor already explored and promising actions (exploitation).

Policy evaluation. In principle, the value functions of a policy π can be determined by solving the Bellman Equations (2.38) and (2.39). *E.g.* for a finite state and action space and a completely known environment model, the Bellman equation of the state-value function is a system of $|\mathcal{X}|$ linear equations with $|\mathcal{X}|$ unknowns. This system can be solved directly or with the *iterative policy evaluation* algorithm, which uses the Bellman equation (finite version of Equation 2.38) as update rule for all states:

$$V_{k+1}(x) = \sum_u \pi(u|x) \sum_{x'} p(x'|x,u)\big(r(x,u,x') + \gamma V_k(x')\big). \tag{2.51}$$

Iterative policy evaluation converges for $k \to \infty$ to the true value function V^π.

However, frequently the environment model is not known and/or solving for an exact solution is infeasible *w.r.t.* computational costs. Therefore, typically RL algorithms approximate the value functions. Value functions can be estimated from experience by interacting with the environment according to the policy and obtaining samples of the reward. For finite state/action spaces, the expectation values in the definitions of the value functions can be approximated with an average value over sampled returns. In these so-called *Monte Carlo methods*, one starts from a given state or state-action pair, performs actions according to the policy, and computes the discounted return from the obtained rewards.

Another approach is a combination of the sampling approach and the update rule of the iterative policy evaluation algorithm in Equation (2.51). In so-called *temporal difference learning*, Equation (2.51) is approximated with a sample $r + \gamma V_k(x')$ and this sample is weighted with a small constant α to update the current state-value estimate of the state x:

$$V_{k+1}(x) = V_k(x) + \alpha\big(r + \gamma V_k(x') - V_k(x)\big), \tag{2.52}$$

with x', r sampled from the environment $p(r, x'|x, u)$ and u sampled from the policy $\pi(u|x)$. Analogously, an update rule for the action-value function at state x and action u can be derived:

$$Q_{k+1}(x,u) = Q_k(x,u) + \alpha\big(r + \gamma Q_k(x',u') - Q_k(x,u)\big), \tag{2.53}$$

with x', r as in the case for the sate-value function and u' sampled from $\pi(u'|x')$. Contrary to applying Equations (2.52) and (2.53) to all states (and actions) in one iteration, the value functions are typically updated for the states (and actions) in the order they appear in the episode $(x,u), (x',u'), \ldots$ when following the policy π.

So far, the presented approaches, the Monte Carlo methods and the update rules, are applied at certain states or state-action pairs and the full value function is obtained by visiting each state or state-action pair frequently enough. This is not feasible for infinite state and action spaces. However, the concepts can be transferred to problems with infinite state and action spaces if *parametrized function approximators* are used for the value functions. In this case, the parametrized function approximators are fitted to targets that are defined by the Monte Carlo samples or the update rules. *E.g.* a class of DRL algorithms approximate the value functions with NNs.

2.3.4 Reinforcement Learning Algorithms

As explained in the previous section, many RL algorithms can be understood as some form of general policy iteration. General policy iteration alternates between policy improvement and policy evaluation and converges, under certain assumptions, to the optimal policy. RL algorithms differ mainly in the approach and accuracy of the policy evaluation. In the following, specific RL algorithms are summarized and common categorizations defined.

First, dynamic programming algorithms are considered. For finite state and action spaces and known environment models, the *policy iteration* and *value iteration* algorithms are applicable. Both algorithms leverage the update rule in Equation (2.51) for policy evaluation and Equation (2.47) for policy improvement. However, policy iteration performs the update rule until convergence, whereas value iteration executes only one iteration before updating the policy.

Definition 1 (Model-based and model-free methods)**.** *In model-based RL, either the environment model, i.e. $p(x'|x,u)$ and $r(x,u,x')$, is known or the agent learns a model in training. Contrary, model-free RL algorithms learn from interaction with the environment, i.e. from samples (x,u,r,x') of state transitions and rewards, without using a model of the environment.*

For unknown environment models, *Q-learning* is a popular version of general policy iteration. Q-learning leverages Equation (2.53) for policy evaluation and performs only one update step followed by a policy improvement step according to Equation (2.47). In this case, policy evaluation and policy improvement can be summarized in one step:

$$Q_{k+1}(x,u) = Q_k(x,u) + \alpha\big(r + \gamma \max_{u' \in \mathcal{U}} Q_k(x',u') - Q_k(x,u)\big). \tag{2.54}$$

For $k \longrightarrow \infty$, the greedy policy $\pi_k(x') = \arg\max_u Q_k(x',u)$ (the target policy) converges to the optimal policy according to the principle of general policy iteration. Q-learning is a so-called *off-policy* method and the interaction with the environment for generating the samples $\{(x,u,r,x')\}$ can be performed with a different policy (the behavior policy) than the target policy as long as all state-action pairs are continued to be updated. Typically, the ϵ-greedy policy in Equation (2.50) is used as behavior policy.

Definition 2 (On-policy and off-policy methods)**.** *In off-policy RL algorithms, the policy for the interaction with the environment (behavior policy) can differ from the policy that is optimized (target policy). Contrary, in on-policy methods the behavior and target policy are the same.*

In recent years, DRL algorithms have gained attention beyond the research community by catching up or even surpassing human experts in playing Atari Games [Mnih et al., 2015] and the board game Go [Silver et al., 2016]. DRL algorithms leverage NNs as function approximators. One of these breakthroughs is *deep Q-learning*, which approximates the action-value function with a deep NN, the so-called *Deep Q Network (DQN)* [Mnih et al., 2015]. In deep Q-learning, the update rule in Equation (2.54) of Q-learning is transferred to an update rule for the parameters θ of the DQN $Q_\theta(x,u)$. First, the DQN of iteration step k is fixed by using a second *target network* $Q_{\theta'}(x,u)$ with the same architecture as $Q_\theta(x,u)$ and by copying the parameters $\theta' = \theta$. For robustness, the target DQN is not updated after fitting to one target sample $r + \gamma \max_{u'} Q_{\theta'}(x',u')$ but rather after fitting C minibatches of samples. In practice, stochastic gradient descent steps are applied to the following loss function:

$$L(\theta) = \frac{1}{N}\sum_{i=1}^{N}\left(r_i + \gamma \max_{u'} Q_{\theta'}(x_i',u') - Q_\theta(x_i,u_i)\right)^2, \tag{2.55}$$

with $\mathcal{D} = \{(x_i,u_i,r_i,x_i')\}$ sampled from previous transitions of the behavior policy. After performing C stochastic gradient descent steps *w.r.t.* the loss function $L(\theta)$, the procedure repeats for the next time step $k+1$ and the parameters θ are copied to the target DQN $\theta' = \theta$. Typically, the ϵ-greedy policy is used as behavior policy and the set of previous transitions $\mathcal{D}$ is called *experience replay buffer*. The usage of the experience replay buffer is sample-efficient and only possible because deep Q-learning is an off-policy algorithm.

Deep Q-learning assumes finite action spaces and the $\max_u Q_{\theta'}(x_i',u')$ operation in Equation (2.55) is not feasible in continuous action spaces. Contrary, continuous action spaces are naturally covered by *policy gradient methods*. These methods approximate the policy $\pi(u|x)$ with a parametrized function approximator $\pi_\phi(u|x)$ with parameters ϕ and perform policy improvement with gradient ascent steps of the expected discounted return $J^{\pi_\phi} = J(\phi)$. The computation of the gradient of the expected return does not require the gradient of the potentially unknown state transition model $p(x'|x,u)$. This statement follows from the *policy gradient theorem* [Sutton et al., 1999].

Theorem 2 (Policy gradient theorem)**.** *The gradient of the expected discounted return $J^{\pi_\phi} = J(\phi)$ of a parametrized policy π_ϕ is given by the following expectation value [Sutton et al., 1999]:*

$$\begin{aligned}\nabla J(\phi) &= \int \sum_{k\in\mathcal{K}} \gamma^k p(X_k{=}x) \int \nabla_\phi \pi_\phi(u|x) Q^{\pi_\phi}(x,u)\,du\,dx, \\ &= E\left[\sum_{k\in\mathcal{K}} \gamma^k \nabla_\phi \log \pi_\phi(U_k|X_k) Q^{\pi_\phi}(X_k,U_k)\right].\end{aligned} \tag{2.56}$$

For deterministic parametrized policies $\pi_\phi(x)$, the gradient of the expected discounted return can be written as [Silver et al., 2014]:

$$\begin{aligned}\nabla J(\phi) &= \int \sum_{k\in\mathcal{K}} \gamma^k p(X_k{=}x) \nabla_\phi \pi_\phi(x) \nabla_u Q^{\pi_\phi}(x,u)_{|u=\pi_\phi(x)}\,dx, \\ &= E\left[\sum_{k\in\mathcal{K}} \gamma^k \nabla_\phi \pi_\phi(X_k) \nabla_u Q^{\pi_\phi}(X_k,u)|_{u=\pi_\phi(X_k)}\right].\end{aligned} \tag{2.57}$$

Given the action-value function Q^{π_ϕ} of the parametrized policy, the policy gradient theorem allows to approximate the gradient of the expected return by sampling roll-outs

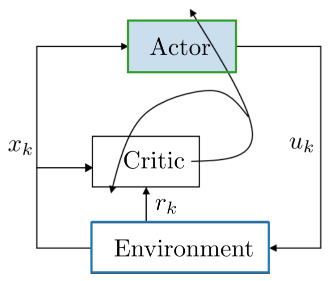


Figure 2.12: Actor-critic methods use different memory structures, *e.g.* parametrized function approximators, to model the policy and the value function. The memory structure of the policy is called actor and of the value function critic. In the DDPG and TD3 algorithm, the critic is updated with temporal difference learning and the actor according to an off-policy version of the deterministic policy gradient theorem. The variable x_k represents the state of the environment, r_k the reward, and u_k the action at time step k. The figure has been adapted from Figure 1 in Brosowsky et al. [2021b].

with the policy π_ϕ and building a sample mean. For the gradient of the expected return, different expressions have been derived. *E.g.* in the REINFORCE algorithm [Williams, 1992], the following expression is used:

$$\nabla J(\phi) = E\left[\sum_{k\in\mathcal{K}} \gamma^k \nabla_\phi \log \pi_\phi(U_k|X_k) G_k\right]. \tag{2.58}$$

In Equation (2.58), the action-value function of Equation (2.56) is substituted with a Monte Carlo estimate of the discounted return G_k. On the one hand, thereby the action-value function $Q^{\pi_\phi}(x, u)$ must not be estimated. On the other hand, the variance in the estimation of the gradient $\nabla_\phi J(\phi)$ is higher. The reduction of the variance in the gradient estimation is addressed by *actor-critic methods*.

Definition 3 (Value-based, policy gradient, and actor-critic methods)**.**

Value-based methods are RL algorithms that leverage memory structures, e.g. parametrized function approximators, to represent a value function, e.g. deep Q-learning [Mnih et al., 2015].

Policy gradient methods are RL algorithms that leverage parametrized function approximators to represent the policy and perform gradient ascent on the expected discounted return, e.g. *REINFORCE [Williams, 1992].*

Actor-critic methods are RL algorithms that leverage different memory structures, e.g. two parametrized function approximators, to represent a value function and the policy, e.g. Deep Deterministic Policy Gradient (DDPG) [Lillicrap et al., 2016].

Actor-critic methods model the policy and the value function with different memory structures, which are called actor and critic, respectively. This is visualized in Figure 2.12. *E.g.* the *DDPG* algorithm uses parametrized function approximators $\pi_\phi(x)$ and $Q_\theta(x, u)$ to model a deterministic policy and the action-value function. The action-value function is updated analogously to deep Q-learning by leveraging a separate target action-value function $Q_{\theta'}(x, u)$ and a separate target policy $\pi_{\phi'}$:

$$L(\theta) = \frac{1}{N} \sum_{i=1}^{N} \big(r_i + \gamma Q_{\theta'}(x'_i, \pi_{\phi'}(x'_i)) - Q_\theta(x_i, u_i)\big)^2. \tag{2.59}$$

The notation of the variables is analogously to Equation (2.55). Then, the policy is improved according to a variant of the deterministic policy gradient theorem for off-policy methods:

$$\nabla_\phi J(\phi) \approx \frac{1}{N} \sum_{i=1}^{N} \nabla_u Q_\theta(x, u)|_{x=x_i, u=\pi_\phi(x_i)} \nabla_\phi \pi_\phi(x_i). \tag{2.60}$$

Approximated gradient ascent steps are performed to improve the policy $\pi_\phi(x)$. Interestingly, the gradient ascent steps on the return can also be interpreted as improving the policy $\pi_\phi(x)$ towards the greedy policy in Equation (2.47). In the DDPG algorithm, the policy improvement is performed with gradient ascent according to Equation (2.60) and the policy evaluation with gradient descent according to Equation (2.59). Both steps are alternated until convergence. DDPG is an off-policy algorithm and uses a replay buffer known from deep Q-learning. A successor of DDPG is the TD3 algorithm [Fujimoto et al., 2018], which is explained in Section 4.3.3 and applied in Chapter 4.4 on a vehicle following controller.

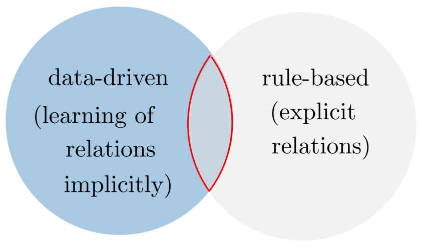
3 Sample-Specific Output Constraints for Neural Networks: ConstraintNet

Contents

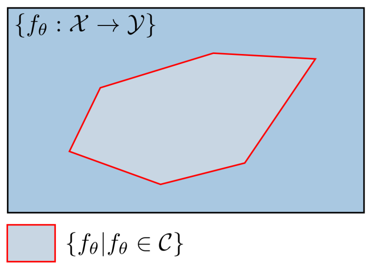
3.1 Motivation 50
3.2 Related Work 54
3.3 ConstraintNet 60
3.3.1 Sample-Specific Output Constraints for Neural Networks 60
3.3.2 Architecture and Construction 61
3.3.3 Constraint Guard Layer 63
3.3.4 Training 65
3.3.5 Supported Constraints and Generalizations 66
3.4 Facial Landmark Detection Experiments 68
3.4.1 Overview 68
3.4.2 Output Constraints 69
3.4.3 Training and Quantitative Results 72
3.4.4 Qualitative Results 75
3.5 Conclusion 79

For many tasks, NNs achieve a high performance on average. However, it is typically not ensured that the predictions are consistent with prior knowledge and safety requirements. Addressing this, Chapter 3.1 introduces sample-specific output constraints and proposes ConstraintNet—a novel and efficient NN architecture that constrains the output range in each forward pass independently to an externally specifiable geometry. In Chapter 3.2, related work is presented, which includes NNs with projection layers. Chapter 3.3 explains the construction of ConstraintNet and the modeling of different classes of constraints. In Chapter 3.4, ConstraintNet is evaluated in facial landmark detection experiments and several output constraints are modeled. Finally, Chapter 3.5 concludes the findings.

Parts of this chapter have previously appeared in the publication *Sample-Specific Output Constraints for Neural Networks* [Brosowsky et al., 2021a] and in the patents DE10 2019 119 739 A1, DE10 2020 127 051 A1, DE10 2021 100 765 A1, US 2021 0027150 A1, US 2022 0114416 A1 [Brosowsky, 2019, 2020, 2021a,b,c].


data-driven (learning of relations implicitly)
rule-based (explicit relations)

a) Combining data-driven and rule-based models


$\{f_\theta : \mathcal{X} \rightarrow \mathcal{Y}\}$
$\{f_\theta | f_\theta \in \mathcal{C}\}$

b) Constrained solution space

Figure 3.1: a) Data-driven models learn complex non-linear relations implicitly from data. In comparison, rule-based models are constructed by experts and frequently represented with explicit and interpretable expressions. Rule-based models are typically designed to ensure intended properties and safety guarantees. Imposing constraints on data-driven models combines the benefits of both approaches. b) Prior knowledge in the form of explicit rules can be leveraged to constrain the solution space of data-driven models to consistent, intended, and safe behavior. The boundary (red) between valid and invalid solutions is sharp as visualized or softened depending on whether the constraints are hard or soft.

3.1 Motivation

DNNs have become state-of-the-art in many high dimensional decision-making tasks by learning complex non-linear relationships implicitly from data. However, as discussed in Chapter 1 the opaque decision-making process, the so-called black box character, limits their application in safety-critical environments. A promising approach to improve the safety of and the trust in NNs is to reduce the black box character by incorporating *prior knowledge*. Frequently, prior knowledge is expressed by explicit relations that represent constraints on the model [Karpatne et al., 2017; Li and Srikumar, 2019; Pham et al., 2018]. These constraints enable the specification of a valid solution space with consistent, intended, and safe behavior (see Figure 3.1). Depending on whether the approach guarantees or only encourages constraint satisfaction, the constraints are called *hard* [Márquez-Neila et al., 2017] or *soft* [Karpatne et al., 2017], respectively. On the one hand, combining the implicit and data-driven approach of NNs with prior knowledge in the form of explicit relations is an ongoing challenge. On the other hand, such approaches promote consistency, interpretability, generalization, as well as reliability and safety.

Frequently, safety can be improved by deriving *output constraints* from expert knowledge and imposing them on NNs [Márquez-Neila et al., 2017; Dalal et al., 2018; Pham et al., 2018]. *E.g.* for safe DRL, Dalal et al. [2018] leverage output constraints on the policy's NN to confine agents to safe regions. Thereby, crashes with walls are avoided. Moreover, output constraints may be imposed on AVs' motion planning algorithms for safety reasons. The output of the planner directly influences the AV's longitudinal and lateral dynamics and the main safety requirement is the avoidance of collisions. The RSS model [Shalev-Shwartz et al., 2017] and the SFF [Nistér et al., 2019] (see Chapter 2.1.3) provide explicit rules that constrain control inputs of AVs and thereby formally guarantee collision avoidance. Consequently, collision avoidance of NN-based planners can be ensured by imposing the constraints of the RSS model or the SFF. One option to implement out-

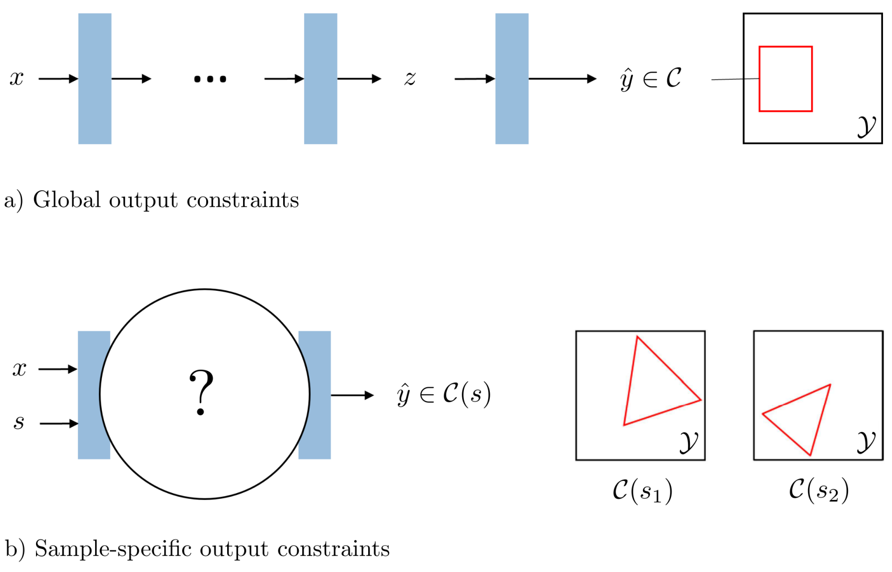

x
z
$\hat{y} \in \mathcal{C}$
$\mathcal{Y}$
a) Global output constraints
x
s
?
$\hat{y} \in \mathcal{C}(s)$
$\mathcal{Y}$
$\mathcal{Y}$
$\mathcal{C}(s_1)$
$\mathcal{C}(s_2)$
b) Sample-specific output constraints

Figure 3.2: a) Frequently, the output $\hat{y}$ of NNs is constrained to a problem-specific value range C by the final layer ϕ. For such global output constraints, the feasible region C is fixed by design of the final layer and constant *w.r.t.* the input. *E.g.* in classification, the softmax function restricts the output components to only valid parameters of the categorical distribution (see Figure 3.3). b) Contrary, sample-specific output constraints are characterized by restricting the output range in each forward pass independently to a different subdomain $C(s)$. The geometry of $C(s)$ is specified by an additional input s.

put constraints are post-processing steps like clipping and projections. However, such separate steps, which are not part of the actual learning task, are prone to be suboptimal and to degrade the performance and efficiency. Instead, this chapter strives for a holistic approach, which ensures constraint satisfaction by the model's construction. Embedding output constraints in the NN's architecture is promising for learning optimal constrained predictions.

It is common practice to constrain the output of NNs with an appropriate final layer to a fixed and problem-specific feasible output region C. The feasible region C is independent of the input and represents therefore a *global output constraint*. *E.g.* the softmax layer ensures positive and normalized output values (see Figure 3.3). However, in many safety-critical environments it is required to restrict the output for each input independently to a different range $C(s)$. In the following, output constraints are called sample-specific if the geometry of the feasible output range $C(s)$ can be specified with a vector description s in each forward pass independently. Figure 3.2 compares global and *sample-specific output constraints* for NNs. In this chapter, the novel NN architecture *ConstraintNet* is proposed. ConstraintNet embeds sample-specific output constraints in its architecture [Brosowsky, 2019; Brosowsky et al., 2021a]. Apart from safety-critical environments, sample-specific output constraints are applicable wherever a partition into valid and invalid outputs is given by an external source, *e.g.* by a human, map data, a rule-based model, or even a second NN. In the mentioned example with the NN-based motion planner, predicted trajectories may be further constrained depending on navigation instructions. In medical

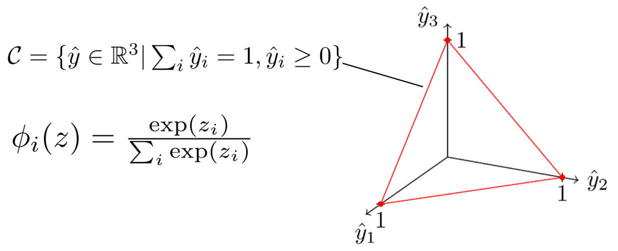

$\mathcal{C} = \{\hat{y} \in \mathbb{R}^3 | \sum_i \hat{y}_i = 1, \hat{y}_i \geq 0\}$
$\phi_i(z) = \frac{\exp(z_i)}{\sum_i \exp(z_i)}$
$\hat{y}_3$
1
$\hat{y}_2$
1
$\hat{y}_1$
1

Figure 3.3: The output of the softmax function ϕ is constrained to only valid parameters of the categorical distribution, *i.e.* the output vector's components are positive $\hat{y}_i \geq 0$ and add up to one $\sum_i \hat{y}_i = 1$. The feasible output space C corresponds to a convex polytope with vertices given by the standard basis and the softmax function parametrizes the interior of this polytope. The figure shows the 3D case.

image processing, a human expert may annotate a region, which confines the localization of an anatomical landmark.

Prior research regarding output constraints for NNs is known from safe DRL [Pham et al., 2018; Dalal et al., 2018] and proposes to apply a final projection layer. The input of the projection layer is an unconstrained action u and the output $u_\perp$ is the element within a state-specific safe set $C(x)$ with minimum distance to the input u:

$$u_\perp = \arg\min_{y \in C(x)} \frac{1}{2} \|y - u\|^2. \tag{3.1}$$

The projection can be considered as an optimization problem and solved with an arbitrary solver. The gradients for the backward pass are determinable by leveraging methods of the more generally studied differentiable projection layers [Amos and Kolter, 2017; Agrawal et al., 2019]. This is explained in more detail in Chapter 3.2 about related work.

Figure 3.4 visualizes the components of the proposed NN architecture ConstraintNet. Instead of performing a projection, a sample-specific parametrization $\phi(z, s)$ of the constrained region $C(s)$ is applied as final layer, the so-called *constraint guard layer*. Thereby, the complete interior of the constrained output range $C(s)$ is covered and almost no computational overhead is required. The input of the constraint guard layer is the output z of the previous layer and the vector description s of the chosen constraint. The vector description s is called *constraint parameter* and considered as an auxiliary input of ConstraintNet. The constraint parameter s specifies the geometry of the feasible output region $C(s)$ in each forward pass separately. Moreover, a *tensor description* $g(s)$ of the constraint parameter informs ConstraintNet about the chosen constraint. The tensor $g(s)$ is included by extending the input or by concatenation to the output of an intermediate layer. In this way, different output constraints, even for the same input x, are applicable. This is beneficial in many applications. *E.g.* in Chapter 3.4, the facial landmark detections are confined to an externally given bounding box around the face. In this example, ConstraintNet is capable of handling bounding boxes with varying shape and size. Altogether, ConstraintNet embeds a precisely described class of constraints, *e.g.* the class of convex polytopes with five vertices in three dimensions, in the NN architecture and an arbitrary constraint of this class can be picked in each forward pass via the additional input s.

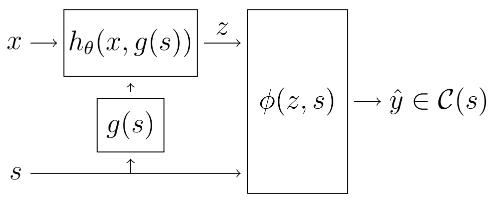


Figure 3.4: Components of ConstraintNet: First, an additional input is introduced for the constraint parameter s and specifies the geometry of the constrained output region $C(s)$. Second, a final layer ϕ maps the output of previous layers $z = h_\theta(x, g(s))$ on the constrained output region $C(s)$ depending on the constraint parameter s. Third, the input of the previous layers h_θ includes a tensor representation $g(s)$ of s. This enables ConstraintNet to deal with different constraints for the same input x. The figure is based on Brosowsky et al. [2021a].

A common element of safe architectures is a safety layer or supervisor. The safety layer monitors the output and intervenes if safety-critical behavior is detected [Shalev-Shwartz et al., 2017; Nistér et al., 2019] (see Figure 2.3). According to this concept, the projection-based approach observes the unconstrained output of the NN and performs a correction if the safe set is exceeded. Contrary, ConstraintNet considers the output constraint at an earlier stage in the architecture. The input z of the constraint guard layer should not be interpreted as an unconstrained output that needs to be monitored and eventually corrected. Contrary, $z = h(x, g(s))$ is generated by previous layers h that take the information about the output constraint via $g(s)$ into account. The intermediate variable z can be interpreted as the learned optimal constrained output in constraint-specific coordinates. The output in constraint-specific coordinates z is then transformed to an output $\hat{y}$ in global and constraint-independent coordinates with $\hat{y} = \phi(z, s)$. The trick is that the output range in constraint-specific coordinates $\mathcal{Z}$ is identical for all constraint parameters s and typically the unbounded $\mathbb{R}^n$. On the one hand, this means that for the same input x with label y different z must be learned depending on the chosen constraint s. On the other hand, the mapping between z and the final output $\hat{y}$ is identical for different inputs x and shared features can be learned. Finally, ConstraintNet is end-to-end trainable with almost no overhead in the forward pass and ensures hard output constraints by construction. Theoretically, ConstraintNet can be extended so that the constraint parameter s and the corresponding feasible region $C(s)$ is learned from data. An additional layer may predict the constraint parameter $s = r_\alpha(x)$ based on the input [Brosowsky, 2020, 2021c]. In another modification, the constraint guard layer may be applied to constrain the output of an intermediate layer instead of the final layer [Brosowsky, 2021a].

The contributions of this chapter are as follows:

- The novel NN architecture ConstraintNet is proposed, which embeds externally adjustable output constraints in the NNs architecture. The capability of imposing sample-specific output constraints allows to incorporate prior knowledge in a flexible way. The more prior knowledge is available, the more specific the constraints can be modeled and *vice versa*.
- For ensuring constraint satisfaction, a final layer in the form of an input-dependent parametrization of the constrained output space, the *constraint guard layer*, is

leveraged. Thereby, the complete interior of the constrained output space is covered and almost no additional computational costs are required.

- Multiple constraints for the same input are applicable. The output constraint is set in each forward pass independently by specifying a tensor description of the constraint. The tensor description informs ConstraintNet about the exact geometry of the constraint.
- ConstraintNet supports a broad class of constraints. For constraints in the form of convex polytopes, a compact parametrization based on the vertex representation is proposed. Furthermore, a general mathematical formalization for output constraints of arbitrary parametrizable geometries including non-convex polytopes and unbounded regions is provided and encourages the modeling of problem-specific constraints.

3.2 Related Work

This chapter provides an overview about approaches that leverage prior knowledge to constrain NNs. After an introduction to methods in general, particular methods that impose sample-specific output constraints on NNs are presented in detail. These specific constraints allow to confine predictions to safe sets and are of particular relevance for safety-critical environments.

Constrained Local Models (CLMs) [Cristinacce and Cootes, 2006] detect a set of points, *e.g.* facial landmarks on an image, by imposing a priori known constraints on data-driven detections of single points. The detection is performed in two steps. First, local models generate response maps, which assign costs to possible locations of single points. Second, a global shape model is fitted to the local detections and leverages knowledge about the spatial relations between the points. Thereby, CLMs are hybrid models and combine the advantages of rule-based and data-driven algorithms (see Figure 1.1). This can be illustrated through the example of facial landmark detection. On the one hand, the local appearances of the facial landmarks are diverse and a reliable detection must be robust against variations in occlusion, illumination, hairstyle, and accessories, among other characteristics. ML algorithms are state-of-the-art to provide local evidence for landmarks under these variations. In Zadeh et al. [2017], this step is performed with NNs. On the other hand, prior knowledge exists about the spatial arrangement of facial landmarks in the form of a global shape model. While certain rigid and non-rigid transformations are allowed, detections with large deviations to the shape model indicate wrong predictions that require corrections. The fitting of the global shape model is formulated as a classical optimization problem, *e.g.* with the Point Distribution Model [Cootes et al., 2001]. In CLMs, the shape model utilizes the output of the local models. However, the local models are independent of the global shape model. This drawback motivates holistic approaches and the embedding of constraints directly in NNs.

In this thesis, ConstraintNet is proposed to embed sample-specific output constraints in the NN architecture. In Chapter 3.4, ConstraintNet is evaluated on facial landmark detection tasks and relative positions of landmarks are enforced. Additionally, constraints in the form of bounding boxes with variable location and shape confine the landmark detections in each forward pass separately. Besides ConstraintNet [Brosowsky, 2019; Brosowsky et al., 2021a], many other methods incorporate constraints directly in NNs. The approaches address soft and hard constraints and are spread over a variety of applications. In the following, three categories of methods are distinguished depending on how the constraints are implemented. Methods of the first category add a loss term to penalize constraint violations. Thereby, NNs are encouraged to satisfy imposed constraints. However, these so-called soft constraints are not guaranteed and constraint violations may still occur. Furthermore, the weighting of the different loss terms is not obvious and the loss always represents a trade-off between constraint satisfaction and the actual objective. Karpatne et al. [2017] model lake temperature dynamics at different depths with NNs and derive constraints from physical relationships. Predicted temperatures are related to the water density and the water density increases with the depth. Inconsistencies are penalized with a so-called physics-based loss function. A second group of approaches modifies the optimizer to minimize the loss function under constraints. Márquez-Neila et al. [2017] impose equality constraints on a human pose estimation task and apply the method of Lagrange multipliers on a linearized objective in each training step. The equality constraints enforce symmetry conditions, *e.g.* equal length of the left and the right arm. The approach achieves similar performance as a baseline with soft constraints. Furthermore, the constraints are specific to the training data and constraint satisfaction on novel data is not ensured. A third category of approaches ensures constraint satisfaction by construction of the NN architecture. Li and Srikumar [2019] enforce logical statements between neurons by adding a manually designed distance function to the pre-activation scores. The distance function is designed to be smooth and differentiable and generates large values if a logical statement implies an activation of the neuron. In the following, further methods of the third category are shown and the approaches are related to ConstraintNet.

A common approach to constrain the output or an intermediate layer of NNs is the application of a parametrization $\phi(z)$ of the constrained subdomain $C = \{\phi(z) | z \in \mathcal{Z}\}$. $\mathcal{Z}$ is either the unbounded $\mathbb{R}^N$ or a region that is constrained by previous layers. *E.g.* the softmax layer constrains the output to only valid parameters of the categorical distribution in this way. Figure 3.3 visualizes the constrained output range of the softmax function with three dimensions. Parametrizations are also applicable to more complex constraints. Lezcano-Casado and Martínez-Rubio [2019] leverage the Lie group theory and the exponential map to parametrize the unitary and the special orthogonal group. The approach is applied to constrain the kernel matrices of RNNs. Thereby, the vanishing and exploding gradient problem is addressed, the training is stabilized, and the performance is improved. Cui et al. [2020] propose a final kinematic layer to generate the output of an NN-based trajectory prediction. The kinematic layer (see Figure 3.5) receives a sequence of future steering and acceleration predictions as input, clips these actions to allowed ranges, and transforms them to the corresponding trajectory under a given vehicle model. Thus, the kinematic layer can be considered as a parametrization of kinematically feasible trajectories. All previously presented approaches parametrize a globally fixed subspace C. However, for safety-critical environments sample-specific output constraints $C(s)$ are of particular relevance, which can be varied by the constraint parameter s. ConstraintNet implements

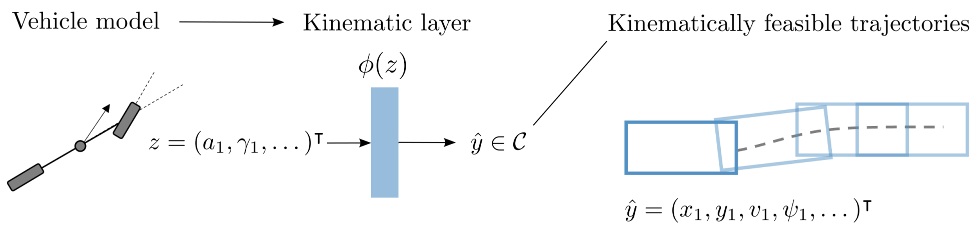

Vehicle model
Kinematic layer
Kinematically feasible trajectories
$\phi(z)$
$z = (a_1, \gamma_1, \dots)^\intercal$
$\hat{y} \in \mathcal{C}$
$\hat{y} = (x_1, y_1, v_1, \psi_1, \dots)^\intercal$

Figure 3.5: Cui et al. [2020] propose an NN with a final kinematic layer ϕ to generate only kinematically feasible trajectories $\hat{y} = (x_1, y_1, v_1, \psi_1, \dots)^\intercal$, with waypoints x_i, y_i, velocities v_i, and heading angles ψ_i. The layer computes the trajectories from series of acceleration and steering commands $z = (a_1, \gamma_1, \dots)^\intercal$ by leveraging a vehicle model.

sample-specific output constraints with the final constraint guard layer. The constraint guard layer is a parametrization that is adjusted by the constraint parameter s:

$$\{\phi(z, s) | z \in \mathcal{Z}\} = \phi(\mathcal{Z}, s) \subseteq C(s). \tag{3.2}$$

The image of the constraint guard layer $\phi(\mathcal{Z}, s)$ is typically the feasible region $C(s)$ or the interior of $C(s)$. The constraint parameter s, which specifies the geometry of the feasible set $C(s)$, can be chosen independently in each forward pass.

As mentioned in the motivation, there are approaches known from DRL [Pham et al., 2018; Dalal et al., 2018; Gros et al., 2020] that constrain the output of policy NNs to safe sets $C(x)$ depending on the current state x of the environment. Instead of a sample-specific parametrization, a projection is performed in the final layer to ensure the output constraint. Typically, the safe sets are derived from prior knowledge and computed with rule-based algorithms from the state x. For the safe set $C(x)$ and an unconstrained action u as input, the projection layer computes the projected action $u_\perp$ with minimum distance to u so that the constraint $u_\perp \in C$ is still satisfied. The projection step in Equation (3.1) can be rewritten with $z = u$ for the input and $\hat{y} = u_\perp$ for the output to highlight the applicability as a layer in NNs in general:

$$\hat{y} = \arg\min_{y \in C} \frac{1}{2} \|y - z\|^2. \tag{3.3}$$

If the input z of the projection is within C, the projection is the identity function. Otherwise, the output is the element in C with the minimum Euclidean distance to z. The latter case is visualized in Figure 3.6.

Pham et al. [2018] consider a 3D object-reaching task with DRL. State-specific safe sets are modeled to comply with physical constraints, *e.g.* constraints for collision avoidance and torque limits. The safe sets are defined via linear equality and inequality constraints $C(x) = \{y | G(x)y \leq h(x), A(x)y = b(x)\}$ and depend on the current state x of the robot. A projection is performed to impose these safe sets on the policy. The projection acts as a safety layer and is called OptLayer. To solve Equation (3.3), Pham et al. [2018] rewrite the

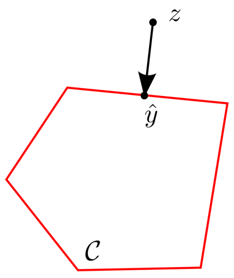


Figure 3.6: The projection layer maps an unconstrained input z on the element $\hat{y}$ within the feasible region C (red polygon) with the minimum Euclidean distance to z.

objective as a Quadratic Program (QP) with $Q = I$ the identity matrix and $q = z$ the input (the unconstrained action):

$$\arg\min_{y} \frac{1}{2} y^\intercal Q y - q^\intercal y, \tag{3.4}$$

$$\text{s.t. } G(x)y \leq h(x), \tag{3.5}$$

$$A(x)y = b(x). \tag{3.6}$$

For QPs, many solvers exist. However, the projection is supposed to be a layer of an NN. Thus, for the backward pass the derivations *w.r.t.* the input z must be determined as well. Both solving the optimization problem and computing the derivations are addressed by methods of the more general differentiable optimization layers [Amos and Kolter, 2017; Agrawal et al., 2019]. Amos and Kolter [2017] propose the NN architecture OptNet with a differentiable layer for solving QPs. OptNet even allows the parameters of the QP to be treated as learnable. The optimization is based on the dual-primal interior point method in Mattingley and Boyd [2012]. Agrawal et al. [2019] generalize the method to differentiable convex optimization layers. From a high level perspective, the method can be explained as follows. First, the convex optimization problem is mapped to a canonical cone program and the problem is solved with a conic solver. Next, the implicit function theorem is applied to the optimality conditions of the cone program. Thereby, the derivations *w.r.t.* the input and the parameters are computed. Finally, the solutions are mapped to the orginal problem. In Chapter 3.4, projection layers are applied on facial landmark detection tasks. The projection layers are implemented as differentiable optimization layers with the Python package cvxpylayers. The package cvxpylayers provides a domain-specific language for convex optimization layers [Agrawal et al., 2019]. Usually, differentiable optimization layers require higher computational costs than explicit layers like convolutional or linear layers [Amos and Kolter, 2017, Figure 1]. Dalal et al. [2018] reduce the runtime of projection layers in specific RL use cases by simplifying the constraints. They derive a closed-form solution by assuming that only one constraint is active at any time and linearizing the constraint *w.r.t.* the action. In the experiments, the constraints are designed to confine the RL agents to specific spatial regions. Note that the proposed simplifications are generally not applicable.

In the following, the fundamentals of the theory of differentiable optimization layers are explained. First, methods for solving constrained optimization problems are presented.

Second, the relevant gradients for the backpropagation algorithm are derived by utilizing the implicit function theorem [Dontchev and Rockafellar, 2009]. Explaining the details would go beyond the scope of this chapter and the reader is referred to the cited publications and books.

Optimization, Karush-Kuhn-Tucker Conditions, and Interior Point Methods

This section is based on Boyd and Vandenberghe [2009, Chapter 5 and 11]. The constrained optimization problem can be formulated as:

$$\begin{aligned} x^* = \arg\min_x \; & f(x), \\ \text{s.t. } & g(x) \leq 0, \\ & h(x) = 0. \end{aligned} \tag{3.7}$$

It is assumed that $x \in \mathbb{R}^N$ and that all functions $f : \mathbb{R}^N \to \mathbb{R}$, $g : \mathbb{R}^N \to \mathbb{R}^M$, $h : \mathbb{R}^N \to \mathbb{R}^P$ are continuously differentiable. Note, for an optimization layer the problem is specified by the input z, *i.e.* f, g, h are functions of x and z. For now, the dependency on z is skipped because it is not relevant for solving the optimization problem. The output of the optimization layer is the solution $\hat{y}(z) = x^*(z)$, which is assumed to be unique. The Lagrangian function of the problem (3.7) is defined by:

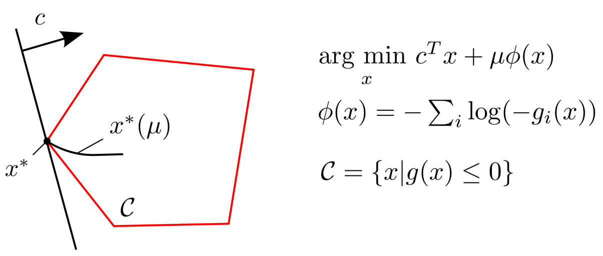
$$L(x, \nu, \lambda) = f(x) + \nu^T g(x) + \lambda^T h(x), \tag{3.8}$$

with $\nu \in \mathbb{R}^M$ and $\lambda \in \mathbb{R}^P$. For a solution x^* of the problem in Equation (3.7) and under certain regularity conditions, there exist ν^* and λ^* that fulfill the following so-called *Karush-Kuhn-Tucker (KKT) conditions*:

$$\begin{aligned} \nabla L(x^*, \nu^*, \lambda^*) = \nabla f(x^*) + \sum_{i=1}^{M} \nu_i^* \nabla g_i(x^*) + \sum_{j=1}^{P} \lambda_j^* \nabla h_j(x^*) &= 0, \\ g(x^*) &\leq 0, \\ h(x^*) &= 0, \\ \nu^* &\geq 0, \\ \nu_i^* g_i(x^*) &= 0 \; \forall i \in 1, \ldots, M. \end{aligned} \tag{3.9}$$

The KKT conditions are a generalization of the method of Lagrange multipliers. The latter allows only equality constraints. If the primal problem in Equation (3.7) is convex, the KKT conditions are also sufficient conditions.

Interior Point Methods (IPMs) solve convex optimization problems with a sequence of equality-constrained problems or modified KKT conditions that approximate the optimization problem with increasing accuracy. The sequence of solutions stays in the interior of the region that is defined by the inequality constraints and converges to the solution of the original problem. An equality-constrained problem can be obtained by adding a barrier function that penalizes solutions close to the boundary defined by the inequality



c
$x^*(\mu)$
x^*
$\mathcal{C}$
$\arg\min_x c^T x + \mu\phi(x)$
$\phi(x) = -\sum_i \log(-g_i(x))$
$\mathcal{C} = \{x | g(x) \leq 0\}$

Figure 3.7: The principle of the barrier method is shown for a linear program with inequality constraints $g(x) \leq 0$. A logarithmic barrier function $\phi(x)$ confines the solutions $x^*(\mu)$ to the interior of the feasible set C (red polygon). Iteratively, μ is decreased and the previous solution $x^*(\mu_i)$ is used as starting point for solving the current problem $x^*(\mu_{i+1})$ with the Newton method. Thereby, $x^*(\mu)$ approximates the true solution x^* with increasing accuracy. The figure has been adapted from Figure 11.2 in Boyd and Vandenberghe [2009, p. 566].

constraints. By choosing a logarithmic barrier function and a small scalar $\mu > 0$, the optimization problem in Equation (3.7) can be approximated with:

$$\begin{aligned} &\arg\min_x f(x) - \mu \sum_{i=1}^{M} \log(-g_i(x)), \\ &\text{s.t. } h(x) = 0. \end{aligned} \tag{3.10}$$

This equality-constrained problem can be solved with the method of Lagrange multiplier:

$$\begin{aligned} 0 &= \nabla f(x^*) - \mu \sum_{i=1}^{M} \frac{1}{g_i(x^*)} \nabla g_i(x^*) + \sum_{j=1}^{P} \lambda_j^* \nabla h_j(x^*), \\ 0 &= h(x^*). \end{aligned} \tag{3.11}$$

The so-called *modified or perturbed KKT conditions* are obtained by defining $\nu_i^*(\mu) = -\mu / g_i(x^*)$ and substituting the corresponding term in Equation (3.11). Thus, the modified KKT conditions are identical to the original ones in Equation (3.9) except that the term $\nu_i^* g_i(x^*) = 0$ is replaced with $\nu_i^*(\mu) g_i(x^*) = -\mu$. Unfortunately, directly solving the modified KKT conditions in Equation (3.11) for a small μ with the Newton method is unstable. However, this issue can be resolved by considering a sequence of problems $x^*(\mu)$ with decreasing values for μ. The problems are solved sequentially and the last solution is used as starting point for the next iteration. The *barrier method* and the *primal-dual IPM* are two important variants of IPMs. The barrier method solves Equation (3.11) directly with the Newton method in an inner loop and reduces the value of μ in an outer loop. For a linear program, the barrier method is visualized in Figure 3.7. Contrary, the primal-dual IPM determines the Newton step from the modified KKT conditions with the additional parameter ν and performs only one loop. In each iteration of the loop, μ is reduced and just one Newton step is performed. Furthermore, a line search is applied to ensure the satisfaction of the inequality constraints. The equality constraints must not necessarily be satisfied. Primal-dual IPMs are state-of-the-art in convex optimization and achieve typically faster convergence with higher accuracy than barrier methods. Compared to classical explicit layers, differentiable optimization layers are still computationally expensive because an iterative solving of linear systems in the Newton step is required.

Optimization as a Differentiable Layer and Implicit Function Theorem

For optimization layers, the optimization problem in Equation (3.7) is specified by the input z of the layer, *i.e.* f, g, h are functions of x and z, and the output is the primal optimal solution $\hat{y}(z) = x^*(z)$. Given a primal-dual optimal solution $q^*(z) = (x^{*\intercal}(z), \nu^{*\intercal}(z), \lambda^{*\intercal}(z))^\intercal$ of the optimization problem for an input z, the gradient $\frac{\partial \hat{y}}{\partial z}$ can be derived from the KKT equality conditions [Barratt, 2018] with the implicit function theorem [Dontchev and Rockafellar, 2009]. By denoting the equality conditions in Equation (3.9) with $\mathrm{KKT}_{\mathrm{eq}}$, the derivative $\frac{\partial q^*}{\partial z}$ can be written as:

$$\begin{aligned} 0 =& \mathrm{KKT}_{\mathrm{eq}}(q^*(z), z) \\ \Leftrightarrow 0 =& \frac{\partial \mathrm{KKT}_{\mathrm{eq}}}{\partial q^*}\frac{\partial q^*}{\partial z} + \frac{\partial \mathrm{KKT}_{\mathrm{eq}}}{\partial z} \\ \Leftrightarrow \frac{\partial q^*}{\partial z} =& -\left(\frac{\partial \mathrm{KKT}_{\mathrm{eq}}}{\partial q^*}\right)^{-1}\frac{\partial \mathrm{KKT}_{\mathrm{eq}}}{\partial z}. \end{aligned} \tag{3.12}$$

Here, the partial derivations denote the corresponding Jacobians $(\frac{\partial y}{\partial x})_{i,j} = \frac{\partial y_i}{\partial x_j}$. The final equation includes the desired expression for $\frac{\partial \hat{y}}{\partial z}$ as the first N rows of $\frac{\partial q^*}{\partial z}$.

3.3 ConstraintNet

This chapter starts with a mathematical definition of sample-specific output constraints for NNs in Section 3.3.1. ConstraintNet ensures sample-specific output constraints with a constraint guard layer and the full architecture is explained in Section 3.3.2. In Section 3.3.3, the constraint guard layer is modeled for constraints in the form of convex polytopes, sectors of a circle, and independent constraints on different output parts. The training of ConstraintNet requires a sampling of valid constraints and the algorithm is explained in detail in Section 3.3.4. Finally, in Section 3.3.5 supported classes of constraints and possible generalizations are presented.

3.3.1 Sample-Specific Output Constraints for Neural Networks

Common NNs $n_\theta : \mathcal{X} \rightarrow \mathcal{Y}$ are function approximators with learnable parameters $\theta \in \Theta$ and map inputs $x \in \mathcal{X}$ to elements $\hat{y}$ of a static output range $\mathcal{Y}$. In supervised learning (see 2.2.2), the parameters θ are deduced from an optimization problem over a data set $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$ with N data points x_i and labels y_i. The behavior of trained NNs is characterized by the data. However, properties or guarantees are typically difficult to prove and the NN is considered as a black box. Addressing this, in the following sample-specific output constraints for NNs are mathematically introduced. Sample-specific output constraints are supposed to implement and ensure desired properties and guarantees.

First, let us define an output constraint as a subset of the output domain $C \subset \mathcal{Y}$ and a class of output constraints as a parametrized set of them $\mathfrak{C} = \{C(s) \subset \mathcal{Y} : s \in \mathcal{S}\}$. $\mathcal{S}$ is a set of the so-called constraint parameters $s \in \mathcal{S}$. For NNs, sample-specific output constraints are defined as output constraints $\mathfrak{C}$ so that the constraint parameter s can be set separately in each forward pass to ensure $\hat{y} \in C(s)$. ConstraintNet is an NN $f_\theta : \mathcal{X} \times \mathcal{S} \rightarrow \mathcal{Y}$ with an additional input for the constraint parameter $s \in \mathcal{S}$ and the guarantee to predict within $C(s)$ by the design of the NN architecture, *i.e.* independently of the learned weights θ:

$$\forall \theta \in \Theta \; \forall s \in \mathcal{S} \; \forall x \in \mathcal{X} : f_\theta(x, s) \in C(s). \tag{3.13}$$

Sample-specific output constraints allow even multiple output constraints for the same input x. This capability has several benefits. The method is robust to variance in the output constraint. *E.g.* bounding boxes, which confine landmark detections, do not need to be centered perfectly. Furthermore, the output constraint may be relaxed or tightened depending on how much prior knowledge is available. *E.g.* if the feasible region is annotated by human experts, each individual may specify another region. Finally, sample-specific output constraints are able to direct the focus in ambiguous situations. *E.g.* for trajectory planning, the output constraints could encode navigation instructions like turn left or right. Another example is an human pose estimation and a bounding box may constrain the prediction to an individual person of a group.

3.3.2 Architecture and Construction

For the integration of sample-specific output constraints, ConstraintNet comprises three components. The components and the general construction approach are shown in Figure 3.4. The approach ensures that ConstraintNet satisfies Equation (3.13) and that backpropagation and gradient-based optimization algorithms remain applicable. Figure 3.8 visualizes a specific implementation of ConstraintNet with a CNN and output constraints in the form of triangles.

The first component is the introduction of the constraint parameter s as an additional input. The constraint parameter specifies the exact geometry of the output constraint $C(s)$. Second, the constraint guard layer $\phi : \mathcal{Z} \times \mathcal{S} \rightarrow \mathcal{Y}$ is applied as final layer. The layer parametrizes the constrained output region $C(s)$ depending on the constraint parameter s. The input $z \in \mathcal{Z}$ of the constraint guard layer is the output of previous layers. Given a class of constraints $\mathfrak{C} = \{C(s) \subset \mathcal{Y} : s \in \mathcal{S}\}$, ϕ must fulfill:

$$\forall s \in \mathcal{S} \; \forall z \in \mathcal{Z} : \phi(z, s) \in C(s). \tag{3.14}$$

Typically, the image $\phi(\mathcal{Z}, s)$ is equal to $C(s)$ or its interior. Furthermore, ϕ must be (piecewise) differentiable *w.r.t.* z so that backpropagation and gradient-based optimization algorithms are applicable. The constraint guard layer ϕ does not comprise learnable parameters and therefore the optimal constrained behavior is learned by the previous

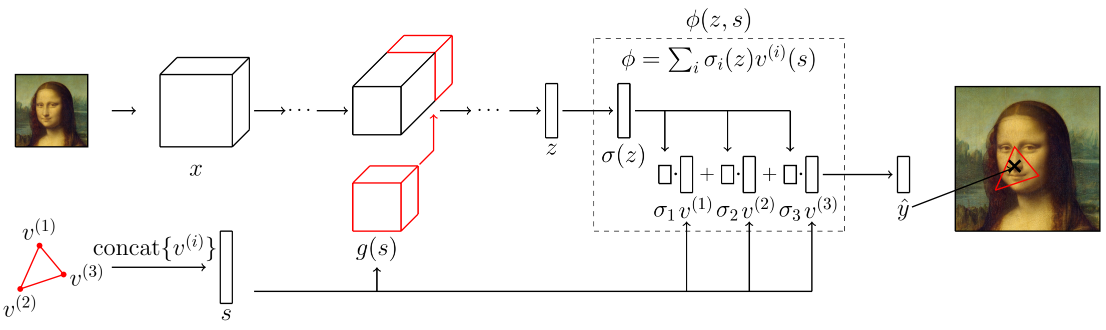

$\phi(z,s)$
$\phi = \sum_i \sigma_i(z) v^{(i)}(s)$
x
z
$\sigma(z)$
$\sigma_1 v^{(1)}$ $\sigma_2 v^{(2)}$ $\sigma_3 v^{(3)}$
$\hat{y}$
$v^{(1)}$
$v^{(2)}$
$v^{(3)}$
concat$\{v^{(i)}\}$
s
$g(s)$

Figure 3.8: Construction of ConstraintNet by extending a CNN. For illustration purposes, the detection of a nose landmark on an image x is shown with an output constraint in the form of a triangle, *i.e.* a convex polytope with three vertices $\{v^{(i)}(s)\}_{i=1}^{3}$. The constraint parameter s specifies the chosen triangle and consists of the six vertex x, y-coordinates. A tensor representation $g(s)$ of s is concatenated to the output of an intermediate convolutional layer and extends the input of the next layer. Instead of creating the final output for the nose landmark with a 2D dense layer, a 3D intermediate representation z is generated. The constraint guard layer ϕ applies a softmax function σ on z and weights the three vertices of the triangle with the softmax outputs. This guarantees a detection $\hat{y}$ within the specified triangle. The figure is based on Brosowsky et al. [2021a].

layers h_θ with parameters θ. In the ideal case, ConstraintNet predicts the same true output y for a data point x under different but valid constraints s. This behavior requires that h_θ depends on s in addition to x. Without this requirement, $z = h_\theta(x)$ would have the same value for fixed x, and ϕ would project this z for different but valid constraint parameters s to different outputs in general. Thus, the third modification informs the previous layers h_θ about the chosen constraint s. The constraint parameter s is transformed into an appropriate tensor description $g(s)$ and represents an additional input of h_θ, *i.e.* $h_\theta : \mathcal{X} \times g(\mathcal{S}) \rightarrow \mathcal{Z}$. For the construction of h_θ, it is proposed to concatenate $g(s)$ to the input x or to the output of an intermediate layer. For deep architectures, the latter is proposed because the first layers extract typically low level features of the input x. This recommendation assumes that the information about the output geometry $g(s)$ is optimally incorporated by layers that extract features with a higher level of abstraction.

To summarize, ConstraintNet is constructed by applying the layers h_θ and the constraint guard layer ϕ sequentially:

$$f_\theta(x,s) = \phi\big(h_\theta(x, g(s)), s\big). \tag{3.15}$$

The property of ϕ in Equation (3.14) implies that ConstraintNet predicts within the constrained output range $C(s)$ according to Equation (3.13). Furthermore, the constraint guard layer propagates gradients and backpropagation remains applicable.

In many image processing tasks, CNNs achieve a high performance and are a natural choice due to their translational equivariance. Thus, Figure 3.8 illustrates the construction of ConstraintNet with a CNN as backbone, *i.e.* the intermediate variable z is generated with a CNN $h_\theta(x, g(s))$. The detection of noses on face images is considered and the output constraints are assumed to be triangles randomly located around the nose, *i.e.* convex polytopes with three vertices. The constraints are specified by a constraint parameter s

consisting of the concatenated vertex coordinates. The constraint guard layer ϕ for convex polytopes is modeled in the next section and requires a 3D intermediate variable $z \in \mathbb{R}^3$ for triangles. The previous layers h_θ map the image data $x \in \mathcal{X}$ and the constraint parameter s on the 3D intermediate variable $z \in \mathbb{R}^3$. A CNN with output domain $\mathcal{Z} = \mathbb{R}^M$ can be realized by adding a dense layer with M output neurons and linear activations. The output of an intermediate convolutional layer and the tensor representation $g(s)$ of s are concatenated and represent the input of the next layer. Thereby, the dependency of h_θ on s is incorporated in a natural way. Empirically, a tensor description $g(s)$ with one channel per constraint parameter component is found as an appropriate representation. All entries within one channel of $g(s)$ are set to the same normalized value of the corresponding constraint parameter component.

3.3.3 Constraint Guard Layer

For a given class of constraints $\mathfrak{C}$, the key challenge in the construction of ConstraintNet is the identification of a corresponding constraint guard layer ϕ. Consequently, explicit expressions of ϕ for broadly applicable classes of constraints are of particular interest and presented in this section. The section starts with a compact constraint guard layer for the general class of convex polytopes with a fixed number of vertices. Furthermore, a concept is shown to constrain different output parts independently. For problem-specific constraints, the construction of tailored constraint guard layers is encouraged. As an example, a parametrization of sectors of a circle is shown. For other common geometric shapes with known parametrizations, the constraint guard layer can be derived analogously. The constraint guard layers in this section are evaluated in Chapter 3.4. Beyond that, Section 3.3.5 covers further supported constraint types and generalizations that are not evaluated in this work and serve as inspiration. Figure 3.9 visualizes several constraint classes and the corresponding constraint guard layers.

Convex Polytopes

A convex polytope $\mathcal{P}$ in $\mathbb{R}^N$ can be described by the convex hull of M vertices $\{v^{(i)}\}_{i=1}^M$ of dimension N:

$$\mathcal{P}\left(\{v^{(i)}\}_{i=1}^M\right) = \left\{ \sum_{i=1}^M p_i v^{(i)} : p_i \geq 0, \sum_{i=1}^M p_i = 1 \right\}. \tag{3.16}$$

The vertices $v^{(i)}(s)$ are assumed to be functions of the constraint parameter s so that the output constraints are defined as $C(s) = \mathcal{P}(\{v^{(i)}(s)\}_{i=1}^M)$. Convex polytopes can be written as the solution space of a finite number of linear inequality constraints $\mathcal{P}(\{v^{(i)}(s)\}_{i=1}^M) = \{y \in \mathbb{R}^N : Ay \leq b\}$ ($A \in \mathbb{R}^{K \times N}$, $b \in \mathbb{R}^K$) and *vice versa* as the convex hull of vertices if the solution space is bounded. The constraint guard layer for convex polytopes $\mathfrak{C} = \{C(s) : s \in \mathcal{S}\}$ with M vertices can be constructed as follows:

$$\phi(z, s) = \sum_{i=1}^M \sigma_i(z) v^{(i)}(s), \tag{3.17}$$

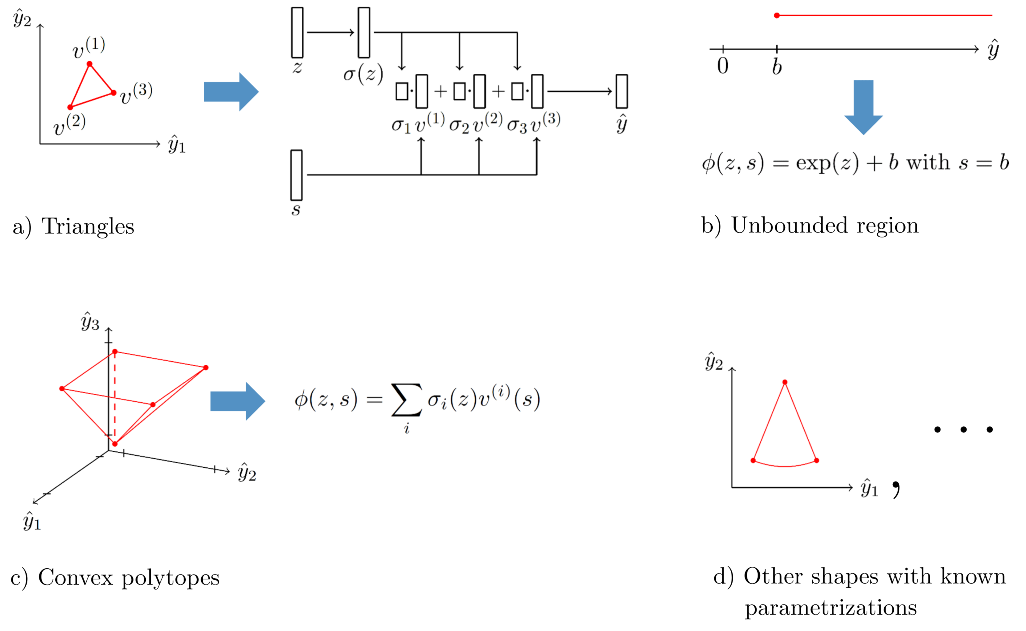


Figure 3.9: Visualization of the constraint guard layers for output constraints in the form of different geometric shapes. a) For triangles, a 3D softmax function σ generates weights for each vertex of the triangle and the output is the weighted sum of the vertices. Thereby, the output is ensured to be in the triangle's interior. b) The exponential function is leveraged to parametrize the unbounded region $[b, \infty)$. c) The constraint guard layer in Equation (3.17) generalizes a) to arbitrary convex polytopes with a fixed number of vertices. d) For other geometric shapes with known parametrizations, the constraint guard layer follows directly from the parametrization.

with $z \in \mathbb{R}^M$. In the formula, $\sigma_i(\cdot)$ denotes the ith component of the M-dimensional softmax function $\sigma : \mathbb{R}^M \to \mathbb{R}^M$. The required property of ϕ in Equation (3.14) follows directly from the properties $0 < \sigma_i(\cdot) < 1$ and $\sum_i \sigma_i(\cdot) = 1$ of the softmax function. To be precise, $\phi(z, s)$ covers the interior of the convex polytope because $\sigma_i(\cdot) \neq 1$. Thus, the boundary of the convex polytope is not reachable exactly but with arbitrary high accuracy. Furthermore, ϕ is differentiable *w.r.t.* z and thus a valid constraint guard layer.

Constraints on Output Parts

Frequently, it is desired to constrain different parts of a multidimensional output independently to a different subdomain. These constraints can easily be implemented by generating an intermediate variable $z^{(i)}$ for each output part and applying individual constraint guard layers $\phi^{(i)}$ in parallel.

Formally, an output with K parts $y^{(k)}$ ($k \in \{1, \ldots, K\}$) is considered:

$$y = (y^{(1)\top}, \ldots, y^{(K)\top})^\top \in \mathcal{Y} = \mathcal{Y}^{(1)} \times \cdots \times \mathcal{Y}^{(K)}. \tag{3.18}$$

Each output part $y^{(k)}$ should be constrained independently to an output constraint $C^{(k)}(s^{(k)})$ of a part-specific class of constraints $\mathfrak{C}^{(k)} = \{C^{(k)}(s^{(k)}) \subset \mathcal{Y}^{(k)} : s^{(k)} \in \mathcal{S}^{(k)}\}$. This is equivalent to constraining the overall output y to:

$$\begin{aligned} &\mathfrak{C} = \{C(s) \subset \mathcal{Y} : s \in \mathcal{S}^{(1)} \times \cdots \times \mathcal{S}^{(K)}\}, \\ &\text{with } C(s) = C^{(1)}(s^{(1)}) \times \cdots \times C^{(K)}(s^{(K)}), \end{aligned} \tag{3.19}$$

and $s = (s^{(1)}, \ldots, s^{(K)})^\intercal$. Furthermore, it is assumed that the constraint guard layers for the parts $\phi^{(k)}$ are given, *i.e.* for $\mathfrak{C}^{(k)}$. For this case, an overall constraint guard layer ϕ, *i.e.* for $\mathfrak{C}$, can be constructed by concatenating the constraint guard layers of the parts:

$$\phi(z, s) = \left(\phi^{(1)\intercal}(z^{(1)}, s^{(1)}), \ldots, \phi^{(K)\intercal}(z^{(K)}, s^{(K)})\right)^\intercal, \tag{3.20}$$

with $z = (z^{(1)\intercal}, \ldots, z^{(K)\intercal})^\intercal$. The validity of the property in Equation (3.14) for ϕ *w.r.t.* $\mathfrak{C}$ follows immediately from the validity of this property for $\phi^{(k)}$ *w.r.t.* $\mathfrak{C}^{(k)}$.

Sectors of a Circle

Let O be a sector of a circle with center position $(x_\mathrm{c}, y_\mathrm{c})^\intercal \in \mathbb{R}^2$ and radius $R > 0$. Furthermore, the sector is assumed to be symmetric *w.r.t.* the vertical line $x = x_\mathrm{c}$ and covers $\Psi \in (0, 2\pi]$ rad. Then, the sector of a circle can be described by the following set of points:

$$O(s) = \left\{r \cdot (\sin\varphi, \cos\varphi)^\intercal + (x_\mathrm{c}, y_\mathrm{c})^\intercal \in \mathbb{R}^2 : r \in [0, R], \varphi \in [-\Psi/2, +\Psi/2]\right\}, \tag{3.21}$$

with $s = (x_\mathrm{c}, y_\mathrm{c}, R, \Psi)^\intercal$. The following constraint guard layer satisfies Equation (3.14) for a class of these constraints $\mathfrak{C} = \{O(s) : s \in \mathcal{S}\}$:

$$\phi(z, s) = r(z_1) \cdot \left(\sin\varphi(z_2), \cos\varphi(z_2)\right)^\intercal + (x_\mathrm{c}, y_\mathrm{c})^\intercal, \tag{3.22}$$

with an intermediate variable $z = (z_1, z_2)^\intercal \in \mathbb{R}^2$, $r(z_1) = R \cdot \mathrm{sig}(z_1)$ and $\varphi(z_2) = \Psi \cdot (\mathrm{sig}(z_2) - 0.5)$. The sigmoid function $\mathrm{sig}(t) = 1/(1+\exp(-t))$ maps a real number to the interval $(0, 1)$. Finally, ϕ is differentiable *w.r.t.* z.

3.3.4 Training

In supervised learning, NNs are trained by utilizing a set of input-output pairs $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$. However, ConstraintNet has an additional input $s \in \mathcal{S}$ that is not unique. The constraint parameter s encodes the shape of the constrained output range $C(s)$, which should cover the true output $y \in C(s)$. Therefore, the constraint parameter s_i of an input-output pair (x_i, y_i) is restricted to a set of valid constraint parameters $\mathcal{S}_{y_i} = \{s \in \mathcal{S} : y_i \in C(s)\}$. In principle, also constraint parameters s_i with $y_i \notin C(s_i)$ may be justified to encourage a prediction within the feasible set with a minimum distance to the optimal output. However, it is proposed to sample s_i only from the valid set $\mathcal{S}_{y_i}$. Thereby, representative input-output triplets (x_i, s_i, y_i) are created. The sampling procedure enables

Algorithm 1 Training algorithm for ConstraintNet. The constraint parameter s_i for an input-output pair (x_i, y_i) is sampled from a set of valid parameters $\mathcal{S}_{y_i} = \{s \in \mathcal{S} : y_i \in C(s)\}$. This pseudocode is based on Brosowsky et al. [2021a].

```
procedure TRAIN($\{x_i, y_i\}_{i=1}^N$)
    $\theta \leftarrow$ random initialization
    for batch do
        $I_{\text{batch}} \leftarrow$ get_indices(batch)
        for $i \in I_{\text{batch}}$ do
            $s_i \leftarrow$ sample($\mathcal{S}_{y_i}$)
            $\hat{y}_i \leftarrow f_\theta(x_i, s_i)$                                ▷ ConstraintNet
        end for
        $L(\theta) \leftarrow \frac{1}{|I_{\text{batch}}|} \sum_{i \in I_{\text{batch}}} \ell(y_i, \hat{y}_i) + \lambda R(\theta)$
        $\theta \leftarrow$ update($\theta, \nabla_\theta L$)
    end for
    return $\theta$
end procedure
```

ConstraintNet to be trained with standard SGD as shown in Algorithm 1. Note that many input-output triplets can be generated for the same input-output pair (x_i, y_i) by sampling different constraint parameters s_i. Therefore, ConstraintNet is encouraged to learn an invariant prediction for the same input under different constraint parameters. Although inconsistent constraint parameters s with $y \notin C(s)$ are not used in training, ConstraintNet shows a desired generalization for invalid constraints (see Figure 3.16 in Section 3.4).

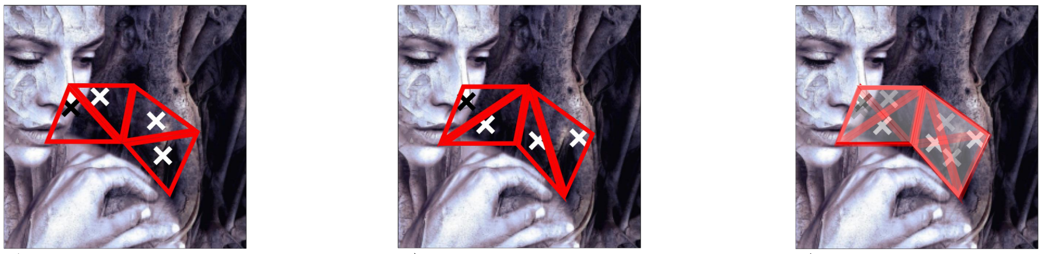
3.3.5 Supported Constraints and Generalizations

According to Equation (3.14), an output constraint C is feasible if a differentiable parametric equation ϕ exists for C. Consequently, the presented constraints are exemplary and the construction of constraint guard layers for problem-specific constraints is encouraged. In the following, general guidelines *w.r.t.* the construction of constraint guard layers are summarized:

a) All constraints of a set $\{C_j\}_{j \in J}$ can be ensured by parametrizing the intersection $C = \bigcap_{j \in J} C_j$. *E.g.* in the facial landmark detection experiment in Chapter 3.4, constraints in the form of relative relations between the landmarks and in the form of bounding boxes are ensured by using this technique (see Equation (3.27) and Figure 3.12).

b) At least one constraint of a set $\{C_j\}_{j \in J}$ can be ensured by performing a prediction $\hat{y}_j$ within each C_j. An additional output with a sigmoid activation function may be introduced to estimate the probability $p(y \in C_j | x)$ that the specified constrained region covers the ground truth. Finally, the output is the prediction $\hat{y}_j$ under the constraint C_j with the highest confidence p to cover the ground truth. This approach is applicable to sets $\{C_j\}_{j \in J}$ with different numbers of constraints.

c) Non-convex polytopes are feasible by partitioning the polytope into a triangle mesh. The triangle mesh represents a set of constraints and exactly one of the constraints



a) Triangle mesh 1 b) Triangle mesh 2 c) Overlayed meshes

Figure 3.10: a) and b) For a nose landmark detection, a constraint in the form of a non-convex polytope is partitioned into two different triangle meshes (red). For each triangle constraint, ConstraintNet performs a prediction (white and black crosses). c) The predictions within the two triangles that cover the true nose landmark (black crosses) are in close vicinity while the others are uncorrelated. Thus, the final prediction can be defined as the mean value of the two predictions in mesh 1 and 2 that are closest together.

must be satisfied. Thus, the non-convex polytope constraint can be ensured analogously to b) or as visualized in Figure 3.10.

d) For an output constraint in the form of separated sets, *e.g.* two separated triangles, the feasible region can be partitioned into connected spaces and b) is applicable.

e) ConstraintNet can be generalized to classification tasks by constraining the continuous logit or probability space. *E.g.* in a multi-label classification task with K classes, the class confidences $0 \leq p_k \leq 1$ may be generated with a sigmoid function $\text{sig} : \mathbb{R}^K \to (0,1)^K$ and shall be individually limited from above $p_k \leq c_k$ with a vector $c \in [0,1]^K$, which is assumed to be given by prior knowledge. In this case, a simple element-wise multiplication of the sigmoid function with the upper bounds $\hat{p} = \text{sig}(z) \odot c$ ensures constraint satisfaction.

f) ConstraintNet is also capable of dealing with unbounded regions. *E.g.* for predicting $\hat{y} \in \mathbb{R}$ above a threshold b, the constraint guard layer $\phi(z,s) = \exp(z) + b$ is proposed and the constraint parameter $s = b$ is the threshold (see Figure 3.9 b).

In principle, the concept of sample-specific parametrizations $\phi(z,s)$ is also applicable to constrain the output of intermediate layers instead of only the final layer [Brosowsky, 2021a]. As long as $\phi(z,s)$ is differentiable *w.r.t.* z, backpropagation is applicable. Sample-specific constraints on intermediate layers might be interesting if interpretable latent representations are supposed to be learned and should be in line with explicit relations, *e.g.* for consistency with rule-based models. A second interesting modification of ConstraintNet is learning the constraint parameter s from data instead of assuming that the parameter is externally given [Brosowsky, 2020, 2021c]. Additional layers $s = r_\alpha(x)$ with own learnable parameters α may be leveraged to determine the geometry of the feasible region from the input. The constraint parameter s may either be learned in a supervised manner with given target values or implicitly by considering s as latent variable of ConstraintNet. The latter requires that $\phi(z,s)$ is also differentiable *w.r.t.* s because the layers r_α are assumed to be part of ConstraintNet.

3.4 Facial Landmark Detection Experiments

3.4.1 Overview

In this section, ConstraintNet is applied to facial landmark detection tasks on images of the *Large-scale CelebFaces Attributes (CelebA) data set* [Liu et al., 2015]. Different classes of output constraints are evaluated and visualized in Figure 3.11. First, output constraints that restrict the landmarks to a bounding box around the face are considered. These constraints are extended to enforce additionally relative relation between the landmarks, *e.g.* that the eyes are above the nose. Moreover, for demonstration purposes triangle and sector of a circle constraints are applied and imposed on the prediction of the nose landmark. The performance is evaluated and compared to conventional NNs without constraints and NNs with a projection layer for constraint satisfaction. The code is available on github[1].

Data Set and Preprocessing

The experiments are performed on the Large-scale CelebFaces Attributes (CelebA) data set [Liu et al., 2015]. This data set consists of a training (162 771 images), a validation (19 867 images), and a test set (19 961 images) with in total 202 599 images, each picturing a face. In addition, *CelebA* provides five landmark annotations, among other attributes. This chapter focuses on predicting the nose, the left eye, and right eye landmark. For preprocessing, the images are rescaled, patches of size 224×224 px are cropped, and finally the patches are normalized.

Modified ResNet50 Architecture

ConstraintNet is constructed by applying the approach described in Section 3.3.2 and by leveraging ResNet50 as backbone. The construction is visualized in Figure 3.8. ResNet50 [He et al., 2016] is a state-of-the-art CNN architecture and is frequently used for classification and regression in computer vision [Lathuilière et al., 2020]. In regression, the output is typically generated by a final dense layer with linear activations. The modifications for ConstraintNet comprise the following steps: adapting the output dimension of the final dense layer with linear activations to match the required dimension of z, adding the constraint guard layer ϕ for the considered class of constraints $\mathfrak{C}$, and concatenating the representation $g(s)$ of the constraint parameter s to the output of an intermediate layer. The representation $g(s)$ is defined as a tensor and the channels $c \in \{1, \ldots, \dim(s)\}$ are identified with the components of the constraint parameter s. Furthermore, all entries within a channel are set to a rescaled value of the corresponding constraint parameter component s_c:

$$g_{c,w,h}(s) = \lambda_c \cdot s_c \; \forall w \in \{1, \ldots, W\}, h \in \{1, \ldots, H\}, \tag{3.23}$$

[1] `https://github.com/mbroso/constraintnet_facial_detect`

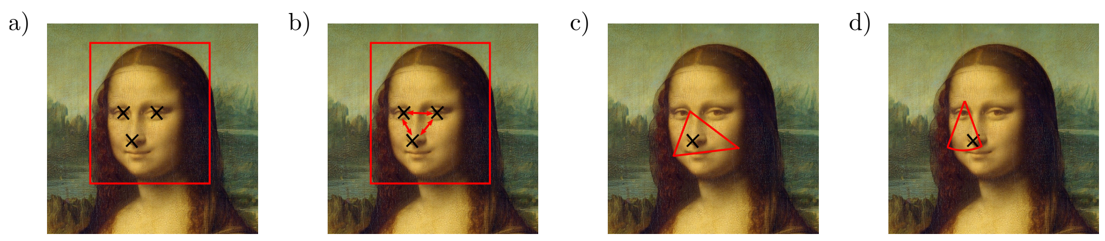


Figure 3.11: Visualization of four different output constraints that are imposed on facial landmark detection. a) Landmark detections for nose, left, and right eye are confined to a bounding box around the face. b) In addition to the bounding box constraints, the eyes are enforced to be above the nose and the left eye is restricted to the left of the right eye. c) and d) The prediction for the nose landmark is constrained c) to a domain in the form of a triangle or d) to a sector of a circle, respectively. This figure is based on the supplementary part of Brosowsky et al. [2021a].

with W and H for the tensor's width and height and each λ_c is a rescaling factor. If identical constraint parameter components s_c occur several times in s, they are encoded in the same channel of g to avoid unnecessary redundancy. The tensor $g(s)$ is concatenated to the output of the 22nd layer of ResNet50 with dimensions $W = H = 28$. This extended output represents the input for the 23rd layer. The 22nd layer is the last layer in the second of four building blocks of ResNet50 and followed by further 27 convolutional layers. The first layers of CNNs recognize image features with a low-level of abstraction, *e.g.* edges. The 22nd layer extracts already high-level abstractions of the image and therefore a joint processing with the constraint parameter representation $g(s)$ is argued to be reasonable. However, it would be interesting to analyze whether optimizing the layer for inserting $g(s)$ even improves performance. Furthermore, rescaling s_c to the scale of the values in the concatenated tensor is found empirically to be important for the convergence and performance of training. The specific values of the rescaling factors λ_c are chosen depending on the considered class of constraints and denoted for each experiment.

3.4.2 Output Constraints

Bounding Box Constraints and Relative Constraints

In the first experiments, the landmarks for the nose $(\hat{x}_\text{n}, \hat{y}_\text{n})^\intercal$, the left eye $(\hat{x}_\text{le}, \hat{y}_\text{le})^\intercal$, and the right eye $(\hat{x}_\text{re}, \hat{y}_\text{re})^\intercal$ are predicted. The NN's output is arranged according to:

$$\hat{y} = (\hat{x}_\text{n}, \hat{x}_\text{le}, \hat{x}_\text{re}, \hat{y}_\text{n}, \hat{y}_\text{le}, \hat{y}_\text{re})^\intercal . \tag{3.24}$$

These landmark detections are confined to a bounding box, which might be given by a face detector. The bounding box is specified by a left $l^{(x)}$, a right $u^{(x)}$, a top $l^{(y)}$, and a bottom $u^{(y)}$ boundary. Note that the y-axis starts at the top of the image and points downwards so that l denotes the lower and u the upper bound. Confining the landmark detections to a bounding box is equivalent to constrain the x-components $\hat{x}_i \in \{\hat{x}_\text{n}, \hat{x}_\text{le}, \hat{x}_\text{re}\}$ to the interval $[l^{(x)}, u^{(x)}]$ and the y-components $\hat{y}_i \in \{\hat{y}_\text{n}, \hat{y}_\text{le}, \hat{y}_\text{re}\}$ to the interval $[l^{(y)}, u^{(y)}]$. These intervals are One Dimensional (1D) convex polytopes with the interval boundaries

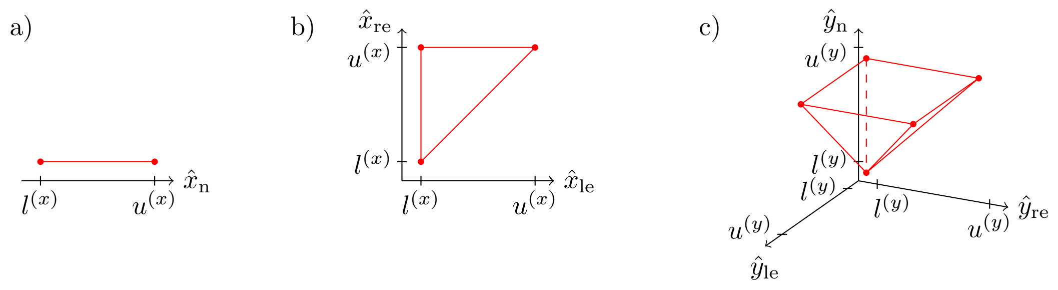


Figure 3.12: Combining the bounding box constraints with relative relations between landmarks is equivalent to constrain the output parts $\hat{y}^{(1)} = \hat{x}_{\mathrm{n}}$ to the line segment in a), $\hat{y}^{(2)} = (\hat{x}_{\mathrm{le}}, \hat{x}_{\mathrm{re}})^\intercal$ to the triangle in b), and $\hat{y}^{(3)} = (\hat{y}_{\mathrm{n}}, \hat{y}_{\mathrm{le}}, \hat{y}_{\mathrm{re}})^\intercal$ to the pyramid in c). This figure is based on Brosowsky et al. [2021a].

as vertices. Thus, the bounding box constraints for the x- and y-components can be written with the definition in Equation (3.16) as:

$$\begin{aligned} C_{\mathrm{bb}}^{(x)}(s^{(x)}) &= \mathcal{P}(\{l^{(x)}, u^{(x)}\}), \\ C_{\mathrm{bb}}^{(y)}(s^{(y)}) &= \mathcal{P}(\{l^{(y)}, u^{(y)}\}), \end{aligned} \tag{3.25}$$

with $s^{(x)} = (l^{(x)}, u^{(x)})^\intercal$ and $s^{(y)} = (l^{(y)}, u^{(y)})^\intercal$. The constraint guard layers for the x- and y-components are derived from Equation (3.17):

$$\begin{aligned} \phi_{\mathrm{bb}}^{(x)}(z^{(\hat{x}_i)}, s^{(x)}) &= \sigma_1(z^{(\hat{x}_i)})l^{(x)} + \sigma_2(z^{(\hat{x}_i)})u^{(x)}, \\ \phi_{\mathrm{bb}}^{(y)}(z^{(\hat{y}_i)}, s^{(y)}) &= \sigma_1(z^{(\hat{y}_i)})l^{(y)} + \sigma_2(z^{(\hat{y}_i)})u^{(y)}, \end{aligned} \tag{3.26}$$

with $z^{(\hat{x}_i)}, z^{(\hat{y}_i)} \in \mathbb{R}^2$ and σ the 2-dimensional softmax function. Note that each of the x- and y-components has an individual latent variable $z^{(\hat{x}_i)}$ or $z^{(\hat{y}_i)}$ but shares the constraint parameter $s^{(x)}$ or $s^{(y)}$. Finally and according to Equation (3.20), the overall constraint guard layer $\phi_{\mathrm{bb}}(z, s)$ is constructed from the constraint guard layers of the components and requires a 12-dimensional intermediate variable $z \in \mathbb{R}^{12}$. The constraint parameter representation $g(s)$ has four channels corresponding to the boundaries of the box and a rescaling factor of $\lambda_c = 0.01$ is chosen for each of the four channels. For bounding boxes that do not exceed the image boundaries, thereby the value range of $g_{c,w,h}(s)$ is reduced to $[0, 2.24]$.

For training of ConstraintNet, valid constraint parameters $s \in \mathcal{S}_y$ are randomly sampled (sample($\mathcal{S}_{y_i}$) in Algorithm 1). Valid constraint parameters correspond to bounding boxes that cover all considered facial landmarks. Thus, first the smallest rectangle that covers the landmarks nose, left eye, and right eye is determined. Next, four integers are sampled from the range $[20, 60]$ and used to extend each of the four rectangle boundaries independently. Finally, the sampled constraint parameter is given by the boundaries of the generated box $l^{(x)}, u^{(x)}, l^{(y)}, u^{(y)}$. For test, the constraint parameter is sampled with the same procedure as in training. This tests how well ConstraintNet performs under bounding boxes with high variability in size.

The bounding box constraints are extended by modeling relations between landmarks. As an example, the left eye is enforced to be in fact to the left of the right eye ($\hat{x}_{\text{le}} \leq \hat{x}_{\text{re}}$) and the both eye landmarks are restricted to be above the nose ($\hat{y}_{\text{le}}, \hat{y}_{\text{re}} \leq \hat{y}_{\text{n}}$). The bounding box constraints $\mathcal{C}_{\text{bb}}$ and these relative constraints $\mathcal{C}_{\text{rel}}$ are combined by parametrizing the intersection of both constraints $\mathcal{C}_{\cap} = \mathcal{C}_{\text{bb}} \cap \mathcal{C}_{\text{rel}}$. This intersection can be written as three independent constraints for the output parts $\hat{y}^{(1)} = \hat{x}_{\text{n}}$, $\hat{y}^{(2)} = (\hat{x}_{\text{le}}, \hat{x}_{\text{re}})^{\top}$, $\hat{y}^{(3)} = (\hat{y}_{\text{n}}, \hat{y}_{\text{le}}, \hat{y}_{\text{re}})^{\top}$:

$$\begin{aligned}
\mathcal{C}_{\cap}^{(1)}(s^{(x)}) &= \{\hat{x}_{\text{n}} \in \mathbb{R} : l^{(x)} \leq \hat{x}_{\text{n}} \leq u^{(x)}\}, \\
\mathcal{C}_{\cap}^{(2)}(s^{(x)}) &= \{(\hat{x}_{\text{le}}, \hat{x}_{\text{re}})^{\top} \in \mathbb{R}^2 : \hat{x}_{\text{le}} \leq \hat{x}_{\text{re}},\ l^{(x)} \leq \hat{x}_{\text{le}}, \hat{x}_{\text{re}} \leq u^{(x)}\}, \\
\mathcal{C}_{\cap}^{(3)}(s^{(y)}) &= \{(\hat{y}_{\text{n}}, \hat{y}_{\text{le}}, \hat{y}_{\text{re}})^{\top} \in \mathbb{R}^3 : \hat{y}_{\text{le}}, \hat{y}_{\text{re}} \leq \hat{y}_{\text{n}},\ l^{(y)} \leq \hat{y}_{\text{n}}, \hat{y}_{\text{le}}, \hat{y}_{\text{re}} \leq u^{(y)}\},
\end{aligned} \tag{3.27}$$

with unchanged constraint parameters $s^{(x)} = (l^{(x)}, u^{(x)})^{\top}$ and $s^{(y)} = (l^{(y)}, u^{(y)})^{\top}$. The constraints $\{\mathcal{C}_{\cap}^{(k)}\}_{k=1}^{3}$ are visualized in Figure 3.12: $\mathcal{C}_{\cap}^{(1)}$ is a line segment in 1D, $\mathcal{C}_{\cap}^{(2)}$ is a triangle in 2D, and $\mathcal{C}_{\cap}^{(3)}$ is a pyramid with 5 vertices in 3D. These geometric shapes are all convex polytopes and can be written in vertex representation according to Equation (3.16):

$$\begin{aligned}
\mathcal{C}_{\cap}^{(1)}(s^{(x)}) &= \mathcal{P}\big(\{v^{(i)}\}_{i=1}^{2}\big) && \text{with } v^{(1)} = l^{(x)}, v^{(2)} = u^{(x)}, \\
\mathcal{C}_{\cap}^{(2)}(s^{(x)}) &= \mathcal{P}\big(\{v^{(i)}\}_{i=1}^{3}\big) && \text{with } v^{(1)} = (l^{(x)}, l^{(x)})^{\top}, v^{(2)} = (l^{(x)}, u^{(x)})^{\top}, v^{(3)} = (u^{(x)}, u^{(x)})^{\top}, \\
\mathcal{C}_{\cap}^{(3)}(s^{(y)}) &= \mathcal{P}\big(\{v^{(i)}\}_{i=1}^{5}\big) && \text{with } v^{(1)} = (l^{(y)}, l^{(y)}, l^{(y)})^{\top}, v^{(2)} = (u^{(y)}, l^{(y)}, l^{(y)})^{\top}, \\
& && v^{(3)} = (u^{(y)}, l^{(y)}, u^{(y)})^{\top}, v^{(4)} = (u^{(y)}, u^{(y)}, u^{(y)})^{\top}, \\
& && v^{(5)} = (u^{(y)}, u^{(y)}, l^{(y)})^{\top}.
\end{aligned} \tag{3.28}$$

In the notation of the vertices, the index for the output part is skipped for readability. However, note that $v^{(i)}$ is a different vertex depending on the output part. Given the vertex representations, the constraint guard layers $\{\phi_{\cap}^{(k)}\}_{k=1}^{3}$ for the three parts are directly given by Equation (3.17). Note that $\phi_{\cap}^{(k)}$ requires an intermediate variable $z^{(k)}$ with dimension equal to the number of vertices of the corresponding polytope, *i.e.* $z^{(1)} \in \mathbb{R}^2$, $z^{(2)} \in \mathbb{R}^3$, and $z^{(3)} \in \mathbb{R}^5$. Finally and according to Equation (3.20), the overall constraint guard layer $\phi_{\cap}$ is given by applying the layers for the parts $\phi_{\cap}^{(k)}$ in parallel. $\phi_{\cap}$ depends on the intermediate variable $z = (z^{(1)\top}, z^{(2)\top}, z^{(3)\top})^{\top}$ with dimension $2+3+5=10$. In other words, for each output part a softmax layer is applied in parallel to generate the weights for the vertices of the convex polytopes in Figure 3.12. The first softmax is of dimension two, the second one of dimension three, and the third one of dimension five. Finally, each of the three output parts is generated by an average weighting of the vertices of the polytopes with the generated softmax probabilities. The constraint parameter representation $g(s)$ and the sampling of valid constraint parameters are identical to the previously explained experiment with only bounding box constraints. Under rotations of the image, the nose might be above the eyes whereas the constraints prohibit such a prediction. Thus, it should be noted that the introduced relations between the landmarks are considered as an example to illustrate the modeling of constraints.

Triangle and Sector of a Circle Constraints

In further experiments, only the nose landmark $(\hat{x}_\mathrm{n}, \hat{y}_\mathrm{n})^\top$ is detected and the detection is constrained to a triangle or a sector of a circle. These constraints are visualized in Figure 3.11 c) and d) and Figure 3.8 shows ConstraintNet and the constraint guard layer for triangle constraints.

A triangle is a convex polytope with three vertices. Therefore, the constraint guard layer is given by Equation (3.17) and requires a 3-dimensional intermediate variable $z \in \mathbb{R}^3$. The triangle constraint is specified with a 6-dimensional constraint parameter s that consists of the concatenated vertex coordinates. For training and test, triangles with random shape around the nose landmark are sampled with the following procedure. First, an equilateral triangle is created on an imaginary circle around the nose landmark with radius of 55.5 px. This corresponds to an area of 4000 px^2 covered by the equilateral triangle. Next, the triangle is set into a polar coordinate system with the point of origin at the center of the triangle. For randomization, for each triangle vertex a deviation for the radius coordinate is sampled from $[-30, 10]$ px and a deviation for the angle coordinate from $[-\pi/6,\ \pi/6]$ rad. These transformations result in a sampling procedure for triangles that cover the nose landmark. For the tensor representation $g(s)$ in Equation (3.23), a rescaling factor of $\lambda_c = 0.01$ is chosen for all six constraint parameter components.

A sector of a circle is defined in Equation (3.21) and the shape and position is specified via the constraint parameter $s = (x_\mathrm{c}, y_\mathrm{c}, R, \Psi)^\top$. The constraint guard layer for constraints in the form of sectors of a circle is given by Equation (3.22) and requires a 2-dimensional intermediate variable $z \in \mathbb{R}^2$. For training and test, a constraint parameter sampling procedure is defined so that the nose landmark $y = (x_\mathrm{n}, y_\mathrm{n})^\top$ is covered by the corresponding sector of a circle. The radius R is sampled from $[50, 100]$ px, Ψ from $[0.4, 1.0]$ rad, y_c from $[y_\mathrm{n} - R, y_\mathrm{n}]$, and x_c from $[x_\mathrm{n} - \delta x, x_n + \delta x]$ with $\delta x = \lfloor \tan(\Psi/2)(y_\mathrm{n} - y_\mathrm{c}) \rfloor$. For the tensor representation $g(s)$ in Equation (3.23), a rescaling factor of $\lambda_c = 0.01$ is chosen for the first three constraint parameter components x_c, y_c, and R and a rescaling factor of $\lambda_4 = 1.0$ for the fourth component Ψ .

3.4.3 Training and Quantitative Results

For a fair comparison, the baseline is created by substituting the constraint guard layer of ConstraintNet with a differentiable optimization layer that performs a projection [Agrawal et al., 2019]. The output of the projection layer is the element within the feasible region $C(s)$ with the shortest Euclidean distance to the input (see Equation (3.3)). Analogous to ConstraintNet, the tensor description of the constraint parameter $g(s)$ is concatenated to the output of the 22nd layer of the projection-based approach. Furthermore, the conventional ResNet50 without any constraint is evaluated as a second baseline.

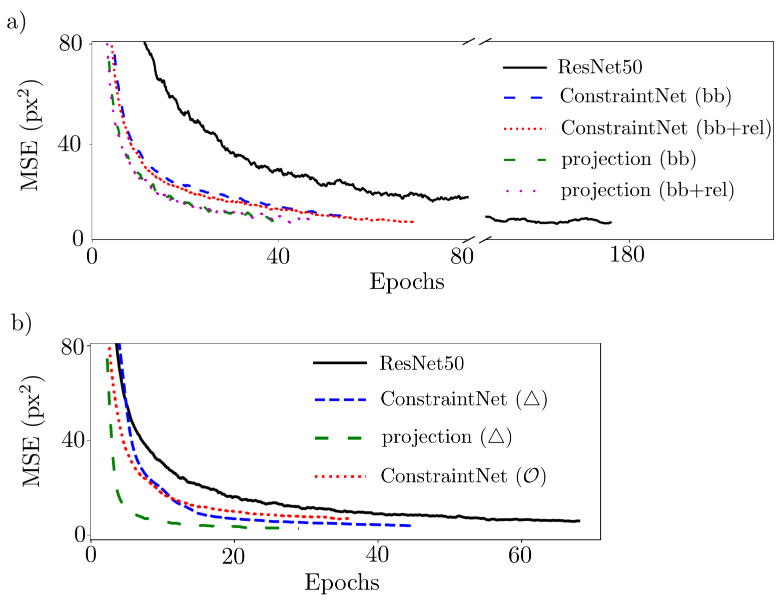


Figure 3.13: Learning curves of ConstraintNet, the projection-based approach and ResNet50. The plots show the MSE as a function of the training progress up to the epoch when overfitting begins. a) Learning curves for models with bounding box (bb) constraints and with additional relative relations between the landmarks (bb+rel). b) Learning curves for triangle (△) and sector of a circle constraints ($\mathcal{O}$). This figure is based on the supplementary part of Brosowsky et al. [2021a].

All models are trained with the Adam [Kingma and Ba, 2014] optimizer (see Section 2.2.2) and the MSE loss is used without a regularization term R ($\lambda = 0$):

$$L(\theta) = \frac{1}{N}\sum_{i=1}^{N} \|y_i - \hat{y}_i\|_2^2 + \lambda R(\theta). \tag{3.29}$$

Here, $\|\cdot\|_2$ is the Euclidean distance. For the experiments with bounding box and relative constraints, a hyperparameter optimization is performed by a search over a discrete range of learning rates and batch sizes. First, five trainings with learning rates $[50, 10, 2, 0.5, 0.1]\cdot 10^{-4}$ and a fixed batch size of 64 are evaluated. Then, the learning rate with the lowest MSE on the validation set is picked and additional five trainings with batch sizes $[8, 16, 32, 64, 128]$ are performed. For all models, a learning rate of $5\cdot 10^{-5}$ and a batch size of 64 are found as the optimal hyperparameters. For the triangle and sector of a circle constraints, these optimal values are reused without performing a separate hyperparameter optimization. In all experiments, training is stopped when overfitting begins and the weights with lowest validation score are used for the evaluation on the test set.

Figure 3.13 shows the learning curves of ConstraintNet and the projection-based approach for the different output constraints. For comparison, ResNet is trained for the same tasks without constraints and the learning curves are depicted as well. ConstraintNet and the projection-based approach are trained according to Algorithm 1 with randomly generated bounding boxes, triangles, and sectors of a circle. ConstraintNet and the projection-based approach converge significantly faster than the ResNet50 baseline. This can be explained

Method (Constraint)	Predicted landmarks	Training		Evaluation
		One epoch	Total (epochs)	
ConstraintNet (bb)	nose, left, and right eye	18 min 1 s	15 h 55 min (53)	(33.2 ± 0.2) s
projection (bb)		32 min 1 s	20 h 17 min (38)	(64.8 ± 0.5) s
ConstraintNet (bb+rel)		17 min 49 s	20 h 11 min (68)	(33.7 ± 0.4) s
projection (bb+rel)		31 min 45 s	24 h 52 min (47)	(67.5 ± 0.8) s
ResNet50 (None)		18 min 28 s	53 h 51 min (175)	(32.9 ± 0.2) s
ConstraintNet (△)	nose	17 min 32 s	12 h 52 min (44)	(34.2 ± 0.7) s
projection (△)		118 min 52 s	55 h 29 min (28)	(295.5 ± 0.8) s
ConstraintNet ($\mathcal{O}$)		18 min 25 s	10 h 44 min (35)	(33.4 ± 0.4) s
ResNet50 (None)		18 min 12 s	20 h 19 min (67)	(32.9 ± 0.2) s

Table 3.1: Runtimes for the training and the evaluation of ConstraintNet, the projection-based approach, and ResNet50 for facial landmark detection tasks. Results are shown for the detection of the nose, left eye, and right eye landmarks with constraints in the form of bounding boxes (bb), additional constraints to enforce relative relations (bb+rel) and no constraints (ResNet50). Furthermore, a detection of only the nose landmark is performed and runtimes are shown for triangle constraints (△), sector of a circle constraints ($\mathcal{O}$), and no constraints (ResNet50). Training is stopped when overfitting begins and the corresponding number of epochs is denoted in brackets (compare the learning curves in Figure 3.13). The evaluation runtime is measured for performing predictions on the complete test set (19 961 images) and computing the MSE metric. The mean values and the standard deviations of the runtimes are determined from 30 repetitions. Training and evaluation is performed with a batch size of 64 and the data loader is parallelized on 10 threads. The runtimes are measured on a Lambda Quad (CPU: 12x Intel Core i7-6850K 3.6 GHz, GPU: 4x Nvidia 12 GB Titan V, RAM: 128 GB) using one of the four dedicated graphics cards.

by the incorporated domain knowledge. The output constraints comprise knowledge about the rough location of specific facial landmarks. The projection-based approach requires fewer epochs but higher runtime than ConstraintNet. One explanation for the different epoch numbers is that the intermediate representation z of ConstraintNet depends not only on the input x but additionally on the constraint parameter s, *i.e.* an invariant prediction for the same input under varying constraints must be learned. Whereas this mechanism requires training the same input-output pairs under varying constraints, the design choice is argued as follows: (1) The dependency of the intermediate variable z on the constraint parameter s enforces ConstraintNet to incorporate the constraint parameter. (2) Diverse decision paths for the same sample must be learned, which may support regularization [Opitz and Shavlik, 1995]. (3) The effort to learn the additional relation is limited because the relation is identical between all samples. (4) In all performed experiments, an explicit order is defined on the vertices. Thereby, learning an invariant prediction under vertex permutations is avoided.

Table 3.1 shows the runtimes for the training and the evaluation of pure ResNet50, the projection-based approach, and ConstraintNet. The projection-based approach requires fewer epochs than ConstraintNet. However, the total runtime for training is higher due to a higher runtime of more than 30 min for a progress of one epoch (162771 images). In comparison, the runtime per epoch of ConstraintNet and ResNet50 is approximately 18 min. Performing the projection requires solving an optimization problem and the computational

Method (Constraint)	MSE($\hat{x}_\text{n}$)	MSE($\hat{y}_\text{n}$)	MSE($\hat{x}_\text{le}$)	MSE($\hat{y}_\text{le}$)	MSE($\hat{x}_\text{re}$)	MSE($\hat{y}_\text{re}$)	MSE($\hat{y}$)
ConstraintNet (bb)	2.17±0.04	**1.63±0.05**	1.88±0.05	1.92±0.06	2.06±0.06	1.57±0.04	11.23±0.16
Projection (bb)	1.84±0.06	1.71±0.10	2.00±0.10	2.03±0.08	2.93±0.09	1.48±0.06	11.99±0.35
ConstraintNet (bb+rel)	**1.43±0.05**	1.76±0.05	**1.80±0.04**	**1.62±0.06**	**1.69±0.04**	**1.39±0.03**	**9.70±0.10**
Projection (bb+rel)	1.62±0.06	1.95±0.09	1.78±0.06	1.74±0.08	1.91±0.05	1.43±0.06	10.44±0.25
ResNet50 (None)	3.28±0.76	2.40±0.34	3.78±0.80	2.20±0.22	3.82±0.78	1.93±0.27	17.42±2.89
ConstraintNet (△)	**1.44±0.03**	**1.59±0.04**	–	–	–	–	**3.03±0.05**
Projection (△)	1.63±0.04	1.66±0.06	–	–	–	–	3.30±0.08
ConstraintNet ($\mathcal{O}$)	2.17±0.05	4.15±0.14	–	–	–	–	6.33±0.15
ResNet50 (None)	4.24±0.53	3.78±0.37	–	–	–	–	8.02±0.67

Table 3.2: MSE metrics for facial landmark detection with ConstraintNet, ResNet50 with projection layer, and pure ResNet50 on the test set of the *CelebA* data set. Results are shown for constraints in the form of bounding boxes (bb), additional constraints to enforce a relative arrangement (bb+rel), triangle (△), and sector of a circle constraints ($\mathcal{O}$). The average covered area of the triangle constraints is (2345 ± 6) px^2 and the area of the sector of the circle constraints (2040 ± 8) px^2.

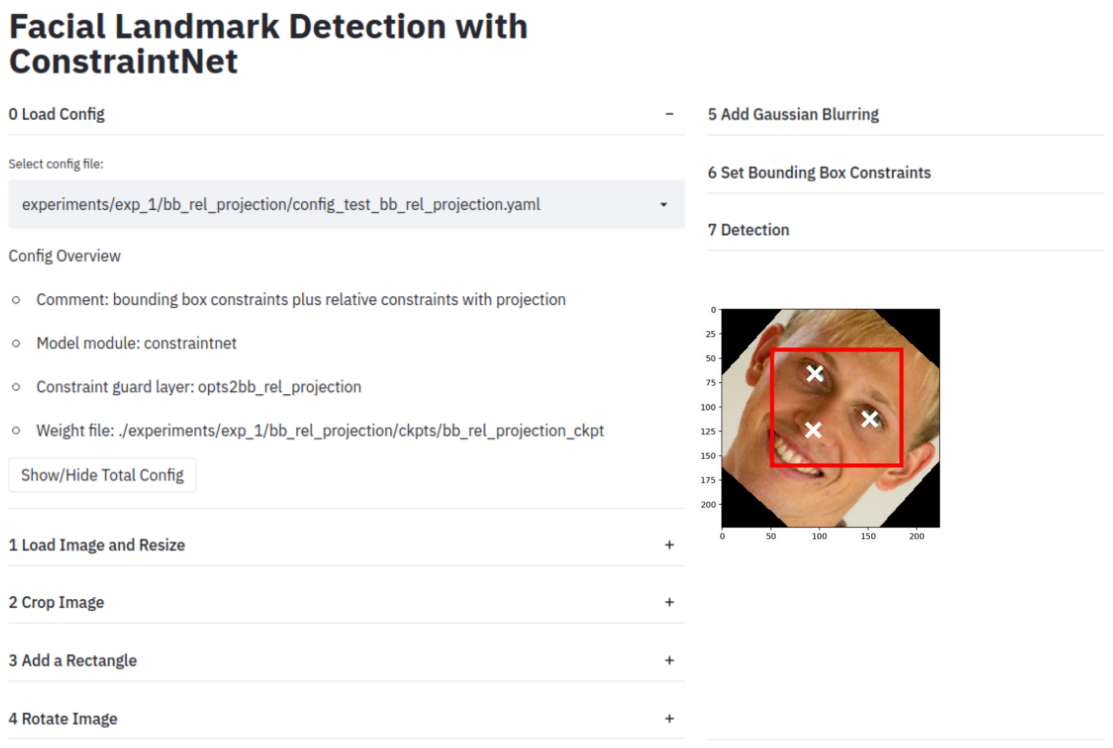
costs are higher than for the execution of the constraint guard layer. Analogously, for test ConstraintNet requires almost no additional runtime compared to ResNet50. Contrary, the projection-based approach requires about twice the time.

After training, the facial landmark predictions of ConstraintNet, the projection-based approach, and ResNet50 without constraints are evaluated on the test set. For the three models and the different types of constraints, Table 3.2 shows the obtained MSE metrics of the facial landmark predictions. For all considered constraints, ConstraintNet achieves lower MSEs metrics than the projection-based approach and the projection-based performs better than pure ResNet50. Consequently, the approaches with imposed output constraints successfully utilize the information encoded in the constraint. This is in line with the observation that the performance with additional relative constraints improves over just bounding box constraints. Furthermore, applying the constraint guard layer instead of the projection-layer seems to have a positive impact on the performance. One possible explanation is that the smooth transformation of the constraint guard layer from the latent space $\mathcal{Z}$ on the feasible region $C(s)$ is better suited for SGD than a projection on the boundary of $C(s)$. *E.g.* for the feasible region $C = \{y \geq 0\}$ in 1D, the projection layer is identical to the ReLU, which has non-informative gradients of zero for inputs $z < 0$.

3.4.4 Qualitative Results

For a qualitative evaluation of the facial landmark detection tasks, a graphical user interface has been developed with streamlit[2], which is visualized in Figure 3.14 and available on github[3]. The app allows to select ConstraintNet, the projection-based approach, or ResNet50 without output constraints as NN for the prediction. Furthermore, widgets are configured to load an image, to perform preprocessing and manipulation steps, to set the

[2] `https://streamlit.io`, accessed on 10/10/2022
[3] `https://github.com/mbroso/constraintnet_facial_detect`



Facial Landmark Detection with ConstraintNet

0 Load Config

Select config file:

experiments/exp_1/bb_rel_projection/config_test_bb_rel_projection.yaml

Config Overview

- Comment: bounding box constraints plus relative constraints with projection
- Model module: constraintnet
- Constraint guard layer: opts2bb_rel_projection
- Weight file: ./experiments/exp_1/bb_rel_projection/ckpts/bb_rel_projection_ckpt

Figure 3.14: For the facial landmark detection experiments, an interactive user interface has been developed with streamlit. The user can unfold the steps 0 to 6 and set the options for specifying the NN, loading an image, preprocessing and manipulation steps, and for setting the output constraint. Finally, in step 7 the corresponding facial landmark detection is shown. In the depicted screenshot, the NN with the projection layer and bounding box as well as relative output constraints is selected.

output constraint depending on the specified NN, and to perform and visualize the facial landmark detection. The graphical user interface is structured in step 0 to 6:

0. *Load config*: A trained NN is loaded by setting the configuration file of the training. An arbitrary NN of the experiments in Table 3.2, except the experiment with sector of a circle constraints, can be specified.
1. *Load image and resize*: An image can be loaded by specifying the path. Furthermore the image is resized so that the shorter dimension is of length 300 px.
2. *Crop image*: A 224 px × 224 px patch of the image can be specified for cropping.
3. *Add a rectangle* (optional): A part of the image can be covered with a colored rectangle of arbitrary size, shape, and color (as in Figure 3.15).
4. *Rotate image* (optional): The image can be rotated with an arbitrary angle.
5. *Add Gaussian blurring* (optional): The image is blurred with an adjustable standard deviation.
6. *Set output constraint*: If the projection-based approach or ConstraintNet is selected in step 0, the output constraint (bounding box or triangle) must be specified.

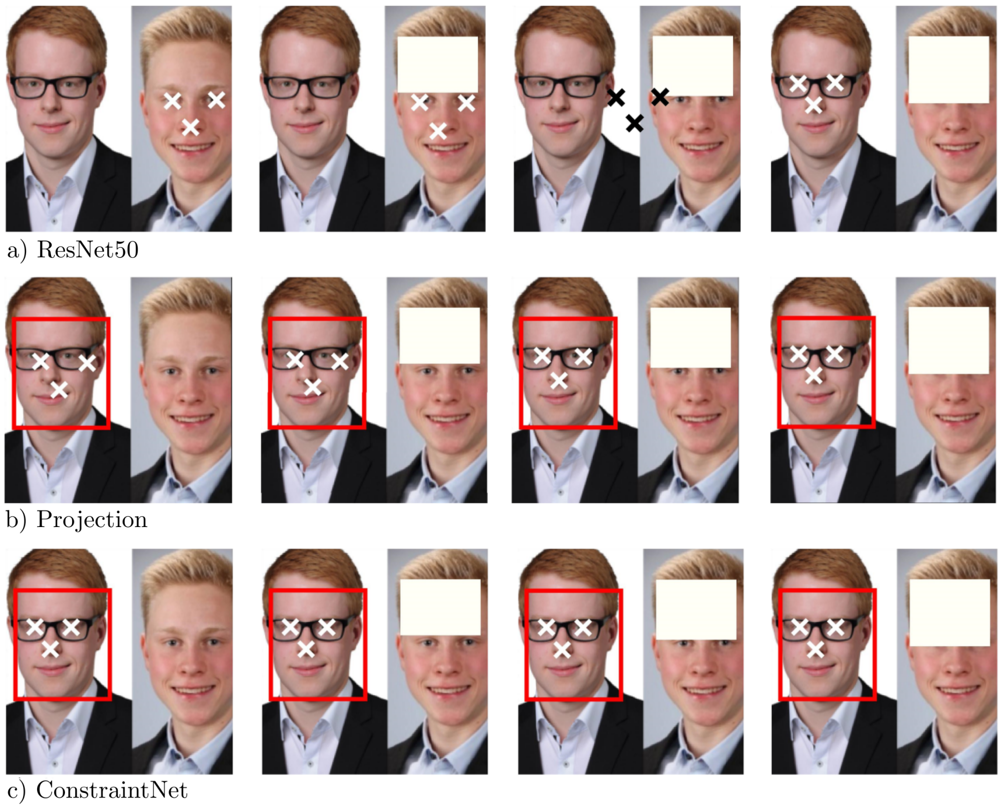


Figure 3.15: Merging two portraits generates an ambiguous setting for the facial landmark detection because the NNs are modeled for detections on images picturing only one face. a) ResNet50 without output constraints predicts the facial landmarks of the face on the right-hand side and interpolates between both faces if the preferred face is covered with a white rectangle. b) and c) Facial landmark detections with the projection-based approach and ConstraintNet with bounding box and relative constraints. The bounding box is set around the face on the left-hand side to resolve the ambiguity. b) The projection-based approach incorporates the information of the bounding box constraint. However, the landmark detections still tend to the face on the right-hand side. c) ConstraintNet predicts the facial landmarks correctly even on the image without the white rectangle. This indicates that ConstraintNet incorporates the information that is provided by the constraint deep in the decision process.

In Figure 3.15, predictions are performed on an image that is generated by merging two portraits. Thereby, the decision-making process is designed intentionally ambiguous because the NNs are trained for a facial landmark detection on images with only one face. In the first row, ResNet50 is applied without output constraints. On the original image, ResNet50 predicts the facial landmarks for the face on the right-hand side. The reason is probably that the face on the right-hand side is larger pictured or that the person wears no glasses. Then, the face on the right-hand side is gradually covered with a white rectangle. Interestingly, first the facial landmark detections are between the two faces and move gradually to the face on the left-hand side. In other applications, this interpolation may cause hazardous behavior. *E.g.* in an NN-based trajectory planner, a highway exit may cause an ambiguous situation with the options to stay on the highway or to take the exit. The interpolation of the optimal trajectories of both scenarios would lead to driving on the impassable area between the highway and exit or a crash in the worst-case. In the center and

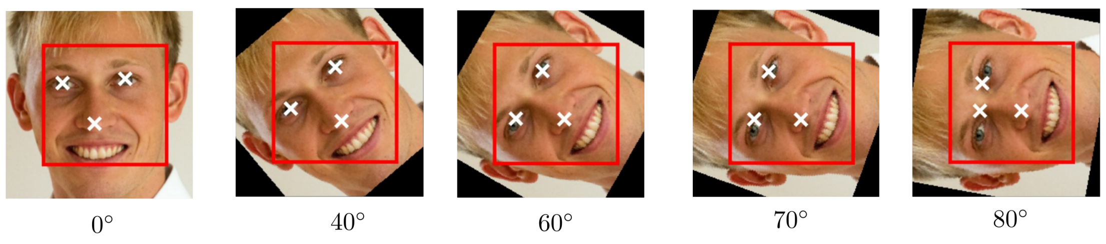


Figure 3.16: Facial landmark detection with ConstraintNet on a portrait with different rotation angles. ConstraintNet ensures bounding box constraints in combination with relative constraints, *i.e.* that the eyes are above the tip of the nose and the left eye is indeed to the left of the right eye. For a rotation of 70°, ConstraintNet detects the left eye and the tip of the nose with deviations from the correct position. The deviations are minimal under the constraint that the landmark for the left eye must be above the tip of the nose. For larger rotation angles, the prediction of the right eye moves to the symmetry axis of the face. This behavior can be explained by missing training data for such rotations and probably ConstraintNet learned that faces are typically symmetric.

bottom row of Figure 3.15, the facial landmark detections of the projection-based approach and of ConstraintNet are shown with bounding box and relative constraints. The bounding box is set to enforce the facial landmark detections to the face on the left-hand side. The projection-based approach incorporates the information of the bounding box constraint and detects the facial landmarks for the face on the left-hand side but with a significant offset in the direction of the second face. The offset vanishes only when the second face is covered to a large extent with the white rectangle. The projection-based approach is generated by substituting the constraint guard layer of ConstraintNet with a projection layer and is informed about the constraint via the constraint parameter representation $g(s)$ analogously to ConstraintNet. Contrary, ConstraintNet predicts the facial landmarks correctly even when the second face is not covered at all. Thus, ConstraintNet leverages the information that is encoded in the constraint strongly. This can be explained by the fact that ConstraintNet must learn the dependency on the tensor representation of the constraint $g(s)$ for correct predictions even under only slightly varying output constraints. Contrary, the projection-based approach does not depend on $g(s)$ necessarily because the projection layer is just the identity function if the prediction is within the feasible set.

Figure 3.16 shows facial landmark detections on a rotated portrait with different rotation angles. The detections are performed with ConstraintNet and bounding box and relative constraints. The relative constraints enforce that the eyes are above the nose and that the left eye is in fact to the left of the right eye. For a rotation of 60°, the left eye is on a similar level than the nose and the output constraint enforces that the landmark for the left eye is above the landmark for the tip of the nose. For a larger rotation of 70°, the left eye is below the tip of the nose. Although such rotations are not explicitly addressed in the training, ConstraintNet generalizes well. The landmark for the left eye is predicted slightly above and the landmark for the nose slightly below the correct position. Thereby, the relative constraint is ensured and the error of the prediction is minimized as far as possible. For a rotation of 80°, a similar behavior is observed. However, the prediction of the right eye is also closer to the symmetry axis of the face. A possible reason is that ConstraintNet learned that faces are usually symmetric.

3.5 Conclusion

In safety-critical environments of DASs, constraints are a common method for the implementation of safety requirements. *E.g.* for ensuring collision avoidance, Nistér et al. [2019] and Shalev-Shwartz et al. [2017] propose constraints on the control input of AVs. However, imposing constraints on NNs is challenging and existing methods are limited. Addressing this, the chapter has focused on the novel NN architecture ConstraintNet that embeds sample-specific output constraints in its architecture. Facial landmark detection experiments have been performed and different constraints have been modeled for illustration purposes. Safety-critical environments of DASs are considered in the next chapters. In addition to safety considerations, incorporating constraints in NNs is promising to reduce the black box character, to stabilize the training, to upgrade the performance, or to improve the data efficiency.

Contrary to prior approaches that perform a projection in the final layer, ConstraintNet applies an input-dependent parametrization of the constrained output space, the so-called constraint guard layer. Thereby, the complete interior of the constrained region is covered, the approach is applicable to DNNs, and the computational overhead is minimal. The constraint guard layer ensures constraint satisfaction by construction. As a second modification, an auxiliary input has been added. The auxiliary input specifies the geometry of the constraint and enables handling multiple constraints for the same sample. Finally, ConstraintNet is end-to-end trainable with almost no overhead in the forward and backward pass. For output constraints in the form of convex polytopes, a constraint guard layer based on the vertex representation has been proposed. The experiments on facial landmark detection tasks have shown improved performance and reduced runtimes over performing projections with differentiable optimization layers.

Moreover, a general mathematical formalization of the approach has been presented and is applicable to output constraints of arbitrary parameterizable geometries. In addition to this formalization, a set of guidelines and ideas for generalizations are provided. This includes concepts for feasible regions that are non-convex polytopes and concepts for restricting the output of an intermediate layer. The mathematical formalization and the generalizations aim to reach researchers of different domains and are intended to encourage the construction of problem-specific constraint guard layers. In particular, ConstraintNet is applicable to safety-critical environments of DASs. This is demonstrated in the next chapter on the example of a vehicle following controller.

4 Safe Reinforcement Learning with Constrained Neural Networks: Vehicle Following Controller

Contents

4.1 Motivation 82

4.2 Related Work 85

4.2.1 Safe Reinforcement Learning 85

4.2.2 Adaptive Cruise Control 87

4.3 Methods 88

4.3.1 Vehicle Following Controller 88

4.3.2 State-Specific Safe Sets 90

4.3.3 Twin Delayed Deep Deterministic Policy Gradient Algorithm . . 98

4.4 Experiments 100

4.4.1 Simulator and Reward Function 100

4.4.2 Training 103

4.4.3 Results 105

4.5 Conclusion 108

This chapter focuses on safe DRL with constrained NNs. On the example of a vehicle following controller, state-specific safe sets are modeled and imposed as output constraints on the control policy. A vehicle following controller is a core functionality of ACC, which is a common ADAS for longitudinal control in modern vehicles. Thus, the proposed approach is called Adaptive Cruise Control with State-Specific Safe Sets (ACC 4S). For the constrained policies, ConstraintNet and projection-based approaches are leveraged. First, Chapter 4.1 provides an overview and motivates the approach. In Chapter 4.2, related safe RL algorithms are presented. Moreover, classical and RL approaches for ACC are compared. In Chapter 4.3, the vehicle following controller is explained from a control theory perspective. Starting from the RSS model, the safe sets are derived and extended. Furthermore, the TD3 algorithm is explained, which has been chosen as DRL algorithm for training. In Chapter 4.4, the details of the experimental evaluation are presented including the simulator and reward function, the NN architectures for the policy, the evaluation metrics, and the final results. Finally, in Chapter 4.5 the findings are discussed.

Parts of this chapter have previously appeared in the publication *Safe Deep Reinforcement Learning for Adaptive Cruise Control by Imposing State-Specific Safe Sets* [Brosowsky et al., 2021b].

4.1 Motivation

In RL, an agent interacts with an environment in a time-discrete way and receives a reward per time step depending on its chosen action and the state of the environment. Frequently, the dynamic of the environment is implemented with a simulator. The objective is the maximization of the cumulative and discounted rewards. In recent years, DRL algorithms have gained attention beyond the research community by surpassing human experts in playing Atari Games [Mnih et al., 2015] and the board game Go [Silver et al., 2016]. In comparison to classical RL, DRL is characterized by leveraging NNs as function approximators.

A key characteristic of RL is self-taught learning. Contrary to supervised learning, the learning algorithm does not rely on a data set with expert labels. Instead, the optimal behavior is learned from experience of past actions, state transitions, and obtained rewards. Of particular importance are model-free RL algorithms, which do not require a mathematical model for the environment transitions but learn it implicitly from experience. On the one hand, the fundamental principle *learning from trial-and-error* is a strength and allows to potentially surpass the best experts in the considered task, even in the case of complex environments [Silver et al., 2016]. On the other hand, it makes RL prone to accidental damage in safety-critical environments. In DRL, safety is even more challenging due to the black box character of NNs. Safe RL addresses these difficulties and gains increasing attention. An important class of these approaches restricts the policy‘s output space to a state-dependent set of safe actions, the safe set $C(x)$. Typically, a final differentiable optimization layer [Amos and Kolter, 2017; Agrawal et al., 2019] is used to project an unconstrained action u on the element $u_\perp$ within the safe set with minimal distance to u [Pham et al., 2018; Dalal et al., 2018]. It is referred to Equation (3.1) in Chapter 3:

$$u_\perp = \arg\min_{y \in C(x)} \frac{1}{2} \|y - u\|^2. \tag{4.1}$$

In the previous chapter, ConstraintNet [Brosowsky et al., 2021a] has been proposed as an alternative approach to impose output constraints on NNs. ConstraintNet applies a sample-specific parametrization of the feasible region in the final layer, the constraint guard layer. Contrary to the projection-based approach, no optimization problem needs to be solved and the computational overhead is minimal. Further concepts and a short overview of safe RL are provided in Chapter 4.2.

For solving control tasks of ADs and ADASs, DRL algorithms with continuous action spaces are of particular interest. The TD3 algorithm [Fujimoto et al., 2018] is a state-of-the-art actor-critic model of this class and a further development of the commonly used DDPG algorithm [Lillicrap et al., 2016]. However, the described safety concerns limit the

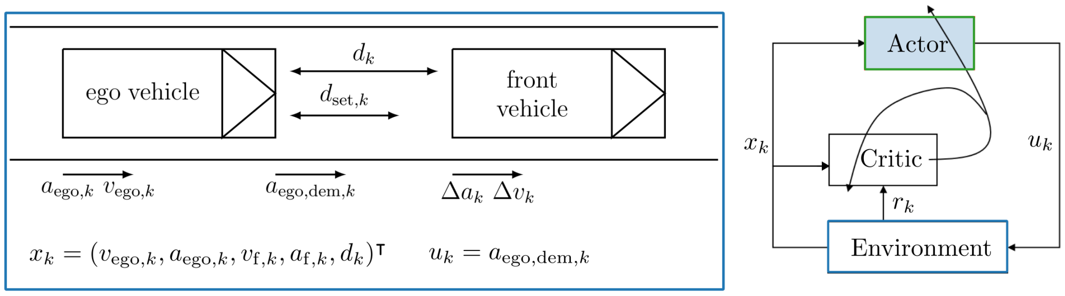

ego vehicle
d_k
$d_{\text{set},k}$
front vehicle
$\overrightarrow{a_{\text{ego},k}\ v_{\text{ego},k}}$
$\overrightarrow{a_{\text{ego,dem},k}}$
$\overrightarrow{\Delta a_k\ \Delta v_k}$
$x_k = (v_{\text{ego},k}, a_{\text{ego},k}, v_{\text{f},k}, a_{\text{f},k}, d_k)^\intercal$
$u_k = a_{\text{ego,dem},k}$
Actor
Critic
Environment
x_k
u_k
r_k

Figure 4.1: *Left*: The vehicle following controller is active if a vehicle ahead is detected. Sensors measure the acceleration $a_{\text{ego},k}$ and velocity $v_{\text{ego},k}$ of the ego vehicle, the distance d_k to the front vehicle, the relative velocity $\Delta v_k = v_{\text{f},k} - v_{\text{ego},k}$, and the relative acceleration $\Delta a_k = a_{\text{f},k} - a_{\text{ego},k}$ of the front vehicle *w.r.t.* the ego vehicle. The subscript k is for the current time step. The control input u_k is a demanded acceleration $a_{\text{ego,dem},k}$ with the goal to reach and maintain a velocity-dependent distance $d_{\text{set},k}(v_{\text{ego}})$ according to a set time gap. *Right:* An accurate, comfortable, and safe vehicle following control behavior is learned with the TD3 algorithm, which is an actor-critic algorithm for continuous action spaces. The following vehicle represents the actor and chooses an action $u_k = a_{\text{ego,dem},k}$ in the form of a demanded acceleration at time step k depending on the state x_k. Next, the environment generates the state and reward for the next time step. The figure is based on Brosowsky et al. [2021b].

application of DRL algorithms for DASs. Addressing this, the chapter focuses on learning a safe vehicle following controller with the TD3 algorithm and output constrained NNs.

A vehicle following controller represents one of the two core capabilities of ACC, which is a common ADAS for longitudinal control in modern vehicles. In the cruise control mode, ACC controls the speed according to a set value or a recognized speed limit. If a vehicle ahead is detected, the second mode is activated and the vehicle following controller keeps a velocity-dependent distance (see Figure 4.1). For a vehicle following controller, the main safety goal is the avoidance of rear-end collisions with the front vehicle. To address this, the RSS model [Shalev-Shwartz et al., 2017] (see Section 2.1.3) defines a safe distance to the front vehicle. If the safe distance is undershot, the following vehicle is supposed to initiate a braking maneuver and minimum conditions *w.r.t.* the deceleration are required. This maneuver is called the proper response. Thereby, the RSS model formally guarantees the avoidance of rear-end collisions even for unexpected full braking of the front vehicle. In Section 4.3.2, the proper response of the RSS model is extended with the goal to avoid undershooting the minimal safe distance. However, if the minimal safe distance is undershot, the response is according to RSS. Thereby, an upper bound $c_{\max,k}$ for the demanded acceleration of the vehicle following controller at time step k is obtained or in other words a lower bound for the demanded deceleration. Additional operational limits lead to a final state-specific interval $C(s_k) = [c_{\min,k}, c_{\max,k}]$ of safe actions. Safe driving is ensured by imposing these state-specific safe sets (4S) as output constraints on the policy, which gives the approach the name ACC 4S. Moreover, safe exploration is ensured by restricting the policies also during training.

For the experiments, ConstraintNet is constructed to embed the state-specific safe sets $C(s_k) = [c_{\min,k}, c_{\max,k}]$ in the NN architecture. Furthermore, ConstraintNet is compared with an unconstrained NN, an NN with an additional clipping to $[c_{\min,k}, c_{\max,k}]$ as post-

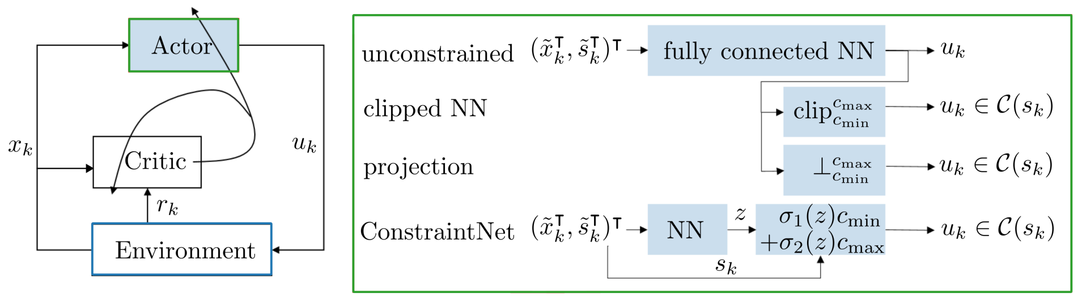

Actor
Critic
Environment
x_k
u_k
r_k
unconstrained
$(\tilde{x}_k^\intercal, \tilde{s}_k^\intercal)^\intercal$
fully connected NN
u_k
clipped NN
$\text{clip}_{c_\text{min}}^{c_\text{max}}$
$u_k \in \mathcal{C}(s_k)$
projection
$\perp_{c_\text{min}}^{c_\text{max}}$
$u_k \in \mathcal{C}(s_k)$
ConstraintNet
$(\tilde{x}_k^\intercal, \tilde{s}_k^\intercal)^\intercal$
NN
z
$\sigma_1(z)c_\text{min}$
$+\sigma_2(z)c_\text{max}$
s_k
$u_k \in \mathcal{C}(s_k)$

Figure 4.2: *Left*: Visualization of learning the control law for vehicle following with an actor-critic model like the TD3 algorithm. *Right*: In the original TD3 algorithm, the actor's policy is modeled with an unconstrained fully connected NN (unconstrained). Here, state-specific safe sets are imposed as constraints $C(s_k)$ on the demanded acceleration $u_k = a_{\text{ego,dem},k}$. Note, $C(s_k)$ is an interval $[c_{\min,k}, c_{\max,k}]$ and s_k is the vector $(c_{\min,k}, c_{\max,k})^\intercal$. Three different approaches are evaluated: 1) clipping as post-processing step (clipped NN), 2) clipping as a differentiable layer (projection), and 3) ConstraintNet. The input of all models is the state of movement x_k and the parameter for the safe set s_k after normalization $(\tilde{x}_k^\intercal, \tilde{s}_k^\intercal)^\intercal$. The figure is based on Brosowsky et al. [2021b].

processing (clipped NN), and clipping as part of the NN (projection-based approach). Note, for the safe set in form of a 1D interval $C(s_k) = [c_{\min,k}, c_{\max,k}]$ the projection in Equation (4.1) is simply the clipping procedure. The difference between the clipped NN and the projection-based approach is that the former considers clipping as part of the environment and the latter as part of the NN with an influence on the backward pass. The different approaches are visualized in Figure 4.2. Imposing state-specific safe sets on the policy has two advantages compared to adding a reward term for safe driving. First, the additional reward term would only encourage safe driving and does not guarantee it. Second, tuning the relative weights of the reward terms is always a trade-off between the different objectives. A higher weight on the reward term for safe driving may degrade the performance. Contrary, considering safety with constraints requires only balancing between comfort and distance accuracy.

The main contributions of this chapter are as follows:

- The RSS model is leveraged and extended to derive state-specific safe sets for a vehicle following scenario. The harsh interventions of the RSS model are transformed into a continuous braking constraint. For the case that the minimal safe distance is approached, the continuous constraint converges to the full braking request of the RSS model. Thereby, abrupt braking requests are smoothed without losing the collision avoidance guarantees of the RSS model.
- A safe vehicle following policy is learned with the TD3 algorithm by imposing the safe sets as hard output constraints on the policy. The output constraints are implemented with ConstraintNet, with clipping as post-processing, and with a projection layer. Furthermore, an unconstrained policy is evaluated by using a conventional NN.
- The effectiveness of the safe sets is validated by measuring a constant crash rate of zero for all constrained policies.

- The training behavior and the performance *w.r.t.* safety, distance accuracy, and comfort of the constrained and the unconstrained policies are evaluated. While all approaches achieve similar comfort metrics, ConstraintNet and the clipped NN achieve the smallest time gap error. Compared to the other approaches, ConstraintNet has the most stable training behavior and converges fastest.

4.2 Related Work

4.2.1 Safe Reinforcement Learning

In RL, the interaction between the agent and the environment is modeled with an MDP. Frequently, the maximum expected return is the only goal and thereby an appropriate definition of the objective. However, in safety-critical environments the maximization of the expected return is not sufficient. In line with the definition of safety in Section 2.1.1, safe RL aims to reduce the risk of the agent during training and/or after deployment. This general definition covers algorithms with different and more specific definitions of risk. Frequently, safe RL algorithms are supposed to solve tasks in real physical environments, *e.g.* to control a real robot arm or an AV, and risk is related to concrete hazards like a collision or torque limits [Pham et al., 2018]. Another common understanding of risk refers to the uncertainty of the return. A policy with a lower expected return may be preferred because the uncertainty of the return is lower. This phenomenon is known as risk aversion.

García et al. [2015] classify safe RL algorithms depending on whether 1) the objective and/or 2) the exploration process are modified. Approaches of the first category extend the objective to include some form of risk. This implies generally also a modification of the exploration process. For a clear separation, the second category covers approaches that modify only the exploration process and keep the maximization of the expected return as objective. 1) Approaches that modify the objective can be further clustered depending on the way how risk is included. a) The expected return is optimized under worst-case assumptions *w.r.t.* the uncertainty in the parameters of the MDP or the intrinsic uncertainty of the MDP [Heger, 1994]. However, frequently worst-case assumptions are too restrictive. b) Contrary, in risk-sensitive RL the objective includes a parameter to balance between risk avoidance and maximizing the expected return [Howard and Matheson, 1972]. c) A third way to modify the objective is the maximization of the expected return under additional constraints. Frequently, the problem is modeled with a Constrained Markov Decision Process (CMDP) [Altman, 1999]. 2) A common form to reduce the risk of the exploration process is the incorporation of expert knowledge instead of learning from scratch. In apprenticeship learning, demonstrations of experts are leveraged to initialize the RL algorithm. For value-based methods, Maire and Bulitko [2005] propose an algorithm to estimate an informative initial value function from demonstrations. For the approach ACC 4S in this chapter, state-specific safe sets are imposed as output constraints on the policy of a vehicle following controller. The safe sets are based on the RSS model [Shalev-Shwartz et al., 2017] and guarantee avoidance of rear-end collisions with the front vehicle. ACC 4S is part of category 1b) because the RSS model assumes reasonable worst-case assumptions *w.r.t.* the uncertainty of the vehicle dynamics. Furthermore, the safe sets are applied as output constraints on the policy. Thus, ACC 4S belongs also to category 1c) and can be interpreted as solving a CMDP with a restricted feasible set as shown in the following.

A CMDP $(\mathcal{X}, \mathcal{U}, p, r, \gamma, C, B)$ [Altman, 1999] extends a classical MDP $(\mathcal{X}, \mathcal{U}, p, r, \gamma)$ by a set of cost functions $C = \{C_i : \mathcal{X} \times \mathcal{U} \times \mathcal{X} \to \mathbb{R}\}$ and a set of cost limits $B = \{b_i \in \mathbb{R}\}$. For each cost function, a cost return $J^\pi_{C_i}$ is defined for a policy π analogously to the expected discounted return J^π in Equation (2.31):

$$J^\pi_{C_i} = E\left[\sum_{k\in\mathcal{K}} \gamma^k C_i(X_k, U_k, X_{k+1})\right]. \tag{4.2}$$

Furthermore, the cost functions and limits define a set of feasible policies:

$$\Pi_C = \{\pi : \forall i\, J^\pi_{C_i} \le b_i\}. \tag{4.3}$$

Finally, the objective of the CMDP is defined as finding a feasible policy with maximum expected return:

$$\pi^* = \underset{\pi\in\Pi_C}{\arg\max}\, J^\pi. \tag{4.4}$$

For the vehicle following controller, the feasible set Π_C should be a set of policies that ensure collision avoidance. A natural definition of a cost function is to define a cost only in the case of a crash:

$$C(x, u, x') = C(x) = \begin{cases} 0 & \text{for } d > 0, \\ 1 & \text{else,} \end{cases} \tag{4.5}$$

with d the distance between the ego and the front vehicle in state x. Then, the cost return is only $J^\pi_C = 0$ if collisions never occur ($d > 0$ for all time steps) under a policy π. Otherwise, the cost return is greater than zero $J^\pi_C > 0$. Consequently, Π_C is exactly the set of collision free policies for a cost limit of zero $b = 0$. However, solving a CMDP according to Equation (4.4) is difficult because the cost return of the policy in Equation (4.2) must be estimated to decide whether the policy is feasible. In model-free RL, the expectation value of the cost return is approximated from samples. Achiam et al. [2017] propose the on-policy method *constrained policy optimization* to solve a CMDP. While the approach can be generally applied and considers safe exploration, constraint violations still occur due to approximation errors. In comparison and for the specific use case of a collision-free vehicle following controller, ACC 4S leverages domain knowledge about the worst-case behavior to ensure constraint satisfaction without approximation errors. By denoting the output constrained policies of ACC 4S with Π_{4S}, the relation $\Pi_{4S} \subseteq \Pi_C$ holds because all policies of ACC 4S are formally ensured to be collision free. Thus, on the one hand imposing safe sets is more restrictive than necessary. On the other hand, collision avoidance is ensured without any approximation errors like in constrained policy optimization. Furthermore, ACC 4S allows the usage of sample-efficient off-policy methods (*e.g.* the TD3 algorithm).

The idea of identifying safe sets and restricting the actions to these sets with a safety layer has been proved as an efficient safe RL approach. Previous research [Pham et al., 2018; Dalal et al., 2018] has focused on projecting the action of an unconstrained policy $u = \pi$ on the safe set, *i.e.* the action $u_\perp$ within the safe set with the minimal Euclidean distance:

$$u_\perp = \arg\min_{y \in C(x)} \frac{1}{2} \|y - u\|^2. \tag{4.6}$$

There are DRL algorithms that perform the projection as a final differentiable layer in the policy NN [Dalal et al., 2018]. The projection can be considered as an optimization problem and the more generally studied differentiable optimization layers [Amos and Kolter, 2017; Agrawal et al., 2019] are applicable. For further explanation, it is referred to Section 3.2.

In this chapter, ConstraintNet is leveraged for safe RL and the output constraints are identified with safe action sets $C(s)$. Here, s is the parameter that describes the geometry of the safe set. Constraint satisfaction is achieved by parametrizing the safe set:

$$\phi(s, \cdot) : \begin{cases} \mathcal{Z} & \rightarrow C(s), \\ z & \mapsto \hat{y}, \end{cases} \tag{4.7}$$

and applying the parametric equation ϕ as final layer, which is called constraint guard layer. For the state-specific safe sets, the constraint parameter s is a function of the state x. The constraint guard layer is motivated as follows. First, the parametric equations are smooth over the total safe set with informative gradients. Contrary, the projection is the identity function if the input is within the safe set and otherwise a mapping on the boundary of the safe set. This reduces the information of the gradient, *e.g.* in 1D the gradient is zero if the input is not within the safe set. Second, the constraint guard layer is an explicit layer with almost no computational overhead.

4.2.2 Adaptive Cruise Control

ACC is an ADAS for longitudinal control and deployed in many modern vehicles. It comprises two working modes. The vehicle following controller gets activated if a vehicle in front is detected. Otherwise, the cruise control mode is active and keeps the velocity close to a set velocity or to the recognized speed limit. Different control algorithms have been proposed for ACC and consider safety, comfort, energy consumption, and traffic flow aspects [Darbha and Rajagopal, 1999; Tapani, 2012]. For simple controller and plant models, approaches from classical control theory can be effectively leveraged to meet the intended control behavior [Chamraz and Balogh, 2018; Canale and Malan, 2003]. Frequently, the cruise control is implemented as a simple P, PI, or PID controller [Rout et al., 2016]. Design methods help to achieve the desired control characteristics, *e.g.* pole-placement design [Rout et al., 2016]. However, for more complex systems the optimal tuning of the controller gains is challenging. Advanced approaches for ACC are based on MPC [Bauer and Gauterin, 2016; Sakhdari and Azad, 2018; Lin et al., 2021]. In MPC, the measured current state of the system is predicted with a plant model under a given control strategy up to a finite time horizon and a cost is assigned. In each time step, the control strategy is optimized *w.r.t.* the costs. Finally, the control input is obtained by the first step of the optimized control strategy. Except for model simplifications, this predictive approach is optimal and additional requirements can be incorporated as constraints, *e.g.* in Bauer and Gauterin [2016] constraints are created from map data for a predictive speed control. However, MPC requires a model and computational costs limit the model complexity, the dealing with uncertainties [Li and Gorges, 2019; Bradford and Imsland, 2018], and the length of the prediction horizon. RL is promising to overcome these shortcomings, whereas ensuring safety is challenging. Lin et al. [2021] and Desjardins and Chaib-draa [2011] leverage RL for ACC. They focus mainly on accuracy and comfort and less on safety. *E.g.* Desjardins and Chaib-draa [2011] address safety by penalizing small distances to the front vehicle in the reward function. Nevertheless, rear-end collisions may still occur. In this chapter, safe RL is leveraged and hard output constraints are imposed on the policy to learn a collision-free vehicle following controller.

4.3 Methods

4.3.1 Vehicle Following Controller

The vehicle following controller aims to reach and maintain a desired distance d_{set} to the front vehicle depending on the ego vehicle's velocity v_{ego} and a set time gap T_{set} (see Figure 4.1). For velocities of the ego vehicle v_{ego} below a threshold $\tilde{v}$, an offset distance d_0 is ramped in linearly and d_0 is reached when the vehicle stops:

$$d_{\text{set}}(v_{\text{ego}}) = \begin{cases} T_{\text{set}} \cdot v_{\text{ego}} & \text{for } v_{\text{ego}} > \tilde{v}, \\ T_{\text{set}} \cdot v_{\text{ego}} + d_0 \left(1 - v_{\text{ego}}/\tilde{v}\right) & \text{for } 0 \leq v_{\text{ego}} \leq \tilde{v}. \end{cases} \tag{4.8}$$

The set distance can be written as $d_{\text{set}} = T_{\text{set}}(v_{\text{ego}} + v_{\text{ego,corr}})$ by defining the correction term $v_{\text{ego,corr}}$:

$$v_{\text{ego,corr}}(v_{\text{ego}}) = \begin{cases} 0 & \text{for } v_{\text{ego}} > \tilde{v}, \\ d_0(1 - v_{\text{ego}}/\tilde{v})/T_{\text{set}} & \text{for } 0 \leq v_{\text{ego}} \leq \tilde{v}. \end{cases} \tag{4.9}$$

Then, a natural definition for the current time gap is:

$$T = d/(v_{\text{ego}} + v_{\text{ego,corr}}), \tag{4.10}$$

and $T = T_{\text{set}}$ corresponds to $d = d_{\text{set}}$. The chosen values for the parameters of the time gap policy are summarized in Table 4.1.

For the dynamics of the ego vehicle and for setting a demanded acceleration $a_{\text{ego,dem}}$, it is assumed that a low-level controller exists. Its closed-loop behavior is modeled with a first-order lag with a gain of one and a time constant τ [Ploeg et al., 2011; Li et al., 2017b]. The corresponding transfer function is given by $G_{\text{low}}(s) = 1/(1+\tau s)$. Separate

Parameter	Value	Description
T_{set}	2 s	Time gap of the desired distance.
d_0	3.2 m	Offset distance for standstill.
$\tilde{v}$	3.33 m s^{-1}	Below this velocity threshold the offset distance d_0 is ramped in linearly.
dt	0.1 s	Time step interval for discretization of control input/action.
$\tau_{\text{ego,a}}/\tau_{\text{ego,b}}$	0.7 s/0.3 s	Time constant for the ego vehicle (acceleration/deceleration).
τ_{f}	0.5 s	Time constant for the front vehicle.
p_{f}	0.000833	Probability for a new front vehicle (cut-in and -out scenario).

Table 4.1: Parameters of the time gap policy and the simulator. The table is based on Brosowsky et al. [2021b].

time constants $\tau_{\text{ego,a}}$ and $\tau_{\text{ego,b}}$ are used to take different response times for acceleration and braking into account:

$$\dot{a}_{\text{ego}} = \begin{cases} (a_{\text{ego,dem}} - a_{\text{ego}})/\tau_{\text{ego,a}} & \text{for } a_{\text{ego,dem}} \geq 0, \\ (a_{\text{ego,dem}} - a_{\text{ego}})/\tau_{\text{ego,b}} & \text{for } a_{\text{ego,dem}} < 0. \end{cases} \tag{4.11}$$

For the total model of the plant and the dynamics of the front vehicle, the time is discretized. Furthermore, the time steps are enumerated with the subscript k and the time interval between them is denoted with dt. The state of the system is described by:

$$x_k = (v_{\text{ego},k}, a_{\text{ego},k}, v_{\text{f},k}, a_{\text{f},k}, d_k)^\top, \tag{4.12}$$

with $v_{\text{ego},k}$, $a_{\text{ego},k}$ the velocity and acceleration of the ego vehicle, $v_{\text{f},k}$, $a_{\text{f},k}$ the velocity and acceleration of the front vehicle, and d_k the distance between the vehicles at time step k. The front vehicle‘s acceleration is modeled probabilistically $a_{\text{f},k+1} \sim p_k(a_\text{f})$ to simulate the unknown behavior of the front vehicle's driver. For a given control input $u_k = a_{\text{ego,dem},k}$ and a sampled acceleration of the front vehicle $a_{\text{f},k+1}$, the state of the next time step x_{k+1} is determined by using Equation (4.11) and fundamental kinematic relations. Consequently, the state-space representation for the system is given by:

$$x_{k+1} = Ax_k + Bu_k + Da_{\text{f},k+1}, \tag{4.13}$$

$$A = \begin{pmatrix} 1 & dt & 0 & 0 & 0 \\ 0 & 1 - dt/\tau_{\text{ego}} & 0 & 0 & 0 \\ 0 & 0 & 1 & dt & 0 \\ 0 & 0 & 0 & 0 & 0 \\ -dt & 0 & dt & 0 & 1 \end{pmatrix}, \tag{4.14}$$

$$B = (0, dt/\tau_{\text{ego}}, 0, 0, 0)^\top, \ D = (0, 0, 0, 1, 0)^\top, \tag{4.15}$$

$$\tau_{\text{ego}} = \begin{cases} \tau_{\text{ego,a}} \text{ for } u_k \geq 0, \\ \tau_{\text{ego,b}} \text{ for } u_k < 0, \end{cases} \tag{4.16}$$

$$a_{\text{f},k+1} \sim p_k(a_\text{f}). \tag{4.17}$$

Thus, the plant dynamics are described by a piecewise linear model with an additional probabilistic term for the front vehicle‘s acceleration. It is piecewise linear due to the different values for the time constant τ_{ego} depending on the sign of the control input. In Section 4.4.1, the simulator will be presented, which samples front vehicle trajectories, computes the dynamics of the vehicle following controller, and determines the reward per time step. The simulator uses slightly modified versions of the state transition Equations (4.13)-(4.17) to exclude backwards movement of the ego and front vehicle, and to include cut-in and -out scenarios. Furthermore, a crash ($d = 0$ m) is defined as a terminal state.

As shown in Figure 4.1, the vehicle following controller computes u based on sensor measurements $(v_\text{f}, a_\text{f}, \Delta v, \Delta a, d)^\intercal$, with $\Delta v = v_\text{f} - v_{\text{ego}}$ and $\Delta a = a_\text{f} - a_{\text{ego}}$. Note, the measured quantities contain full information about the state x in Equation (4.12). Therefore, the state is fully observable and the sensor measurements are denoted with x as well. Future states depend only on the state of the current time step and thus the control problem can be modeled as an MDP. The optimal behavior of an MDP is defined by introducing an additional reward function. For optimizing the policy, RL algorithms gain experience from state transitions $x_k, u_k \rightarrow x_{k+1}$ and corresponding rewards r_{k+1}. The behavior policy chooses actions u_k and the transitions are computed with a simulator (see Section 4.4.1). Contrary to model-free RL, in MPC dealing with uncertainty is non-trivial and typically simplifying assumptions are made, *e.g.* Lin et al. [2021] assume that the front vehicle continues with constant velocity and its acceleration is considered as disturbance.

4.3.2 State-Specific Safe Sets

The control input of the vehicle following controller is a demanded acceleration $u = a_{\text{ego,dem}}$ and the goal of this section is the derivation of an interval of safe control inputs:

$$a_{\text{ego,dem},k} \in C(s_k) = [c_{\min,k}, c_{\max,k}], \tag{4.18}$$

with k for the time step and s_k a tuple comprising the lower and upper bound $s_k = (c_{\min,k}, c_{\max,k})^\intercal$. Thus, the safe sets define a feasible band for the demanded acceleration as it is illustrated by the dashed lines in the top right plot of Figure 4.5. In the vehicle following scenario, the rear vehicle, hereafter referred to as ego vehicle, is responsible for collision avoidance and must keep a sufficient safe distance.

The distance d between the front vehicle and the ego vehicle is considered safe if the ego vehicle is able to avoid collisions even in a reasonable worst-case scenario. This is visualized in Figure 4.3. In the worst-case, the front vehicle suddenly brakes fully with $a_\text{f} = -a_{\text{f,min}}$ and the ego vehicle requires a response time ρ before braking fully with $a_{\text{ego}} = -a_{\text{ego,min}}$. It is assumed that the ego vehicle brakes less strongly than the front vehicle $a_{\text{ego,min}} < a_{\text{f,min}}$. In the response time, a maximal acceleration of the ego vehicle of $a_{\text{ego}} = a_{\text{ego,max}}$ is assumed. Further, both vehicles do not drive backwards, *i.e.* the deceleration is set to zero $a_{\text{ego}}(t_{\text{ego,stop}}) = a_\text{f}(t_{\text{f,stop}}) = 0$ when the vehicles stop $v_{\text{ego}}(t_{\text{ego,stop}}) = v_\text{f}(t_{\text{f,stop}}) = 0$.

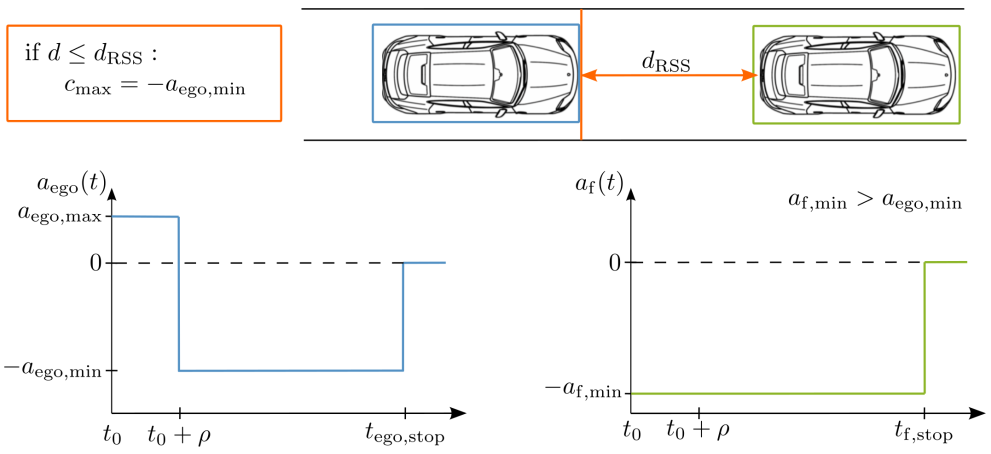


Figure 4.3: For safe vehicle following, the RSS model proposes to keep a minimal safe distance d_{RSS} (orange) to the vehicle in front. The safe distance is defined so that even in a reasonable worst-case scenario the ego vehicle is still able to avoid a crash by following a so-called proper response [Shalev-Shwartz et al., 2017]. In the worst-case, the front vehicle (green) suddenly brakes maximally with $a_{\mathrm{f}} = -a_{\mathrm{f,min}}$. The acceleration profile is plotted in the bottom right. As soon as the minimal safe distance $d \leq d_{\mathrm{RSS}}$ is undershot the ego vehicle (blue) is supposed to request full braking, so that after a response time ρ the deceleration is at least $a_{\mathrm{ego}} = -a_{\mathrm{ego,min}}$. The acceleration profile is visualized in the bottom left. During the response time, the ego vehicle may accelerate maximally with $a_{\mathrm{ego,max}}$. It is assumed that both vehicles do not drive backwards, *i.e.* the accelerations are set to zero at stopping time $t_{\mathrm{ego/f,stop}}$. In the picture of safe action sets, the maximum demanded acceleration of the ego vehicle is bounded from above with $c_{\max} = -a_{\mathrm{ego,min}}$ for $d \leq d_{\mathrm{RSS}}$ and the response time is the allowed time to set the demanded acceleration.

The RSS model formalizes a proper response of the ego vehicle by deriving an explicit expression for the minimal safe distance $d_{\mathrm{RSS}}(v_{\mathrm{ego}}, v_{\mathrm{f}})$ depending on the velocities of both vehicles. If the minimal safe distance $d \leq d_{\mathrm{RSS}}$ is undershot, the ego vehicle has to initiate the described braking maneuver. This response is required until the distance to the front vehicle is again larger than the safe distance. The proper response may be implemented by introducing an upper bound $c_{\max}$ for the ego vehicle's demanded acceleration and setting the bound to:

$$c_{\max} = -a_{\mathrm{ego,min}} \text{ for } d \leq d_{\mathrm{RSS}}. \tag{4.19}$$

This bound assumes a low-level controller that is able to set the demanded acceleration within the response time. However, undershooting the safe distance followed by a harsh full braking should be prevented as well. Therefore, in the following an upper bound is proposed also for $d > d_{\mathrm{RSS}}$ that continuously converges towards full braking $c_{\max} \longrightarrow -a_{\mathrm{ego,min}}$ for $d \longrightarrow d_{RSS}$. Thereby, the Equation (4.19) for the upper bound is extended. Finally, a lower bound $c_{\min,k}$ is defined according to standards for the maximum deceleration and completes the definition of the safe interval.

Property 1. *For the acceleration profiles of the ego and the front vehicle according to the RSS model (visualized in Figure 4.3), the minimal distance between the vehicles occurs either in the beginning t_0 or when both vehicles have come to a stop $t_{\mathrm{stop}} = \max\{t_{\mathrm{ego,stop}}, t_{\mathrm{f,stop}}\}$.*

Parameter	Value	Description
$a_{\mathrm{f,min}}$	$8\,\mathrm{m\,s^{-2}}$	Maximum deceleration of front vehicle (worst-case).
ρ	$0.5\,\mathrm{s}$	Response time of ego vehicle.
$a_{\mathrm{ego,min}}$	$7\,\mathrm{m\,s^{-2}}$	Minimum deceleration of ego vehicle after response time and $d < d_{\mathrm{RSS}}$.
$a_{\mathrm{ego,max}}$	$3\,\mathrm{m\,s^{-2}}$	Maximum acceleration of the ego vehicle within the response time.
j_{max}	$6\,\mathrm{ms^{-3}}$	Maximum jerk in definition for upper bound c_{max}.

Table 4.2: Parameters for the RSS-based safe sets. The table is based on Brosowsky et al. [2021b].

Proof. Three different cases can be considered. 1) If the ego vehicle is faster than the front vehicle at t_0, the front vehicle stops first and $v_{\mathrm{ego}}(t) > v_{\mathrm{f}}(t)$ holds for all $t \in [t_0, t_{\mathrm{stop}}]$ due to the stronger braking of the front vehicle. Consequently, the distance decreases monotonically and the minimum is reached when both vehicles have come to a stop t_{stop}. If the front vehicle is faster than the ego vehicle, the distance monotonically increases until both vehicles have the same velocity. At the time point of same velocity, either 2) both vehicles have come to a stop or 3) from now on the ego vehicle is faster and the distance monotonically decreases analogously to 1). Thus, in case 2) the minimal distance occurs at start time point t_0 and in case 3) either at start time point t_0 or at stop time point t_{stop}. □

If the stopping distance of the ego vehicle $s_{\mathrm{ego}} = x_{\mathrm{ego}}(t_{\mathrm{ego,stop}}) - x_{\mathrm{ego}}(t_0)$ is larger than of the front vehicle $s_{\mathrm{f}} = x_{\mathrm{f}}(t_{\mathrm{f,stop}}) - x_{\mathrm{f}}(t_0)$, the distance at stop is smaller than in the beginning. With Property 1 it follows that the minimal distance is reached at t_{stop} and the minimal safe distance is $s_{\mathrm{ego}}(v_{\mathrm{ego}}) - s_{\mathrm{f}}(v_{\mathrm{f}})$. Otherwise, the minimal distance occurs in the beginning t_0 and a distance of zero $d = 0$ is safe because the distance increases even in the worst-case. Consequently, the minimal safe distance can be written compactly as:

$$d_{\mathrm{RSS}}(v_{\mathrm{ego}}, v_{\mathrm{f}}) = \max\{0, s_{\mathrm{ego}}(v_{\mathrm{ego}}) - s_{\mathrm{f}}(v_{\mathrm{f}})\}, \tag{4.20}$$

with $s_{\mathrm{ego/f}} = x_{\mathrm{ego/f}}(t_{\mathrm{stop}}) - x_{\mathrm{ego/f}}(t_0)$ the stopping distance in the worst-case scenario. The following analytical expressions for $s_{\mathrm{ego/f}}$ arise from twice integration of the acceleration profiles in Figure 4.3:

$$\begin{aligned} s_{\mathrm{ego}}(v_{\mathrm{ego}}) &= v_{\mathrm{ego}}\rho + \frac{1}{2}a_{\mathrm{ego,max}}\rho^2 + \frac{(v_{\mathrm{ego}} + a_{\mathrm{ego,max}}\rho)^2}{2a_{\mathrm{ego,min}}}, \\ s_{\mathrm{f}}(v_{\mathrm{f}}) &= \frac{v_{\mathrm{f}}^2}{2a_{\mathrm{f,min}}}. \end{aligned} \tag{4.21}$$

The worst-case assumption is modeled by setting the response time ρ, the maximum acceleration of the ego vehicle within the response time $a_{\mathrm{ego,max}}$, the minimum braking of the ego vehicle after the response time $a_{\mathrm{ego,min}}$, and the maximum braking of the front vehicle $a_{\mathrm{f,min}}$. For the experiments in Chapter 4.4, the parameters are set according to Table 4.2 and represent a trade-off between safety and pragmatic distances between the vehicles.

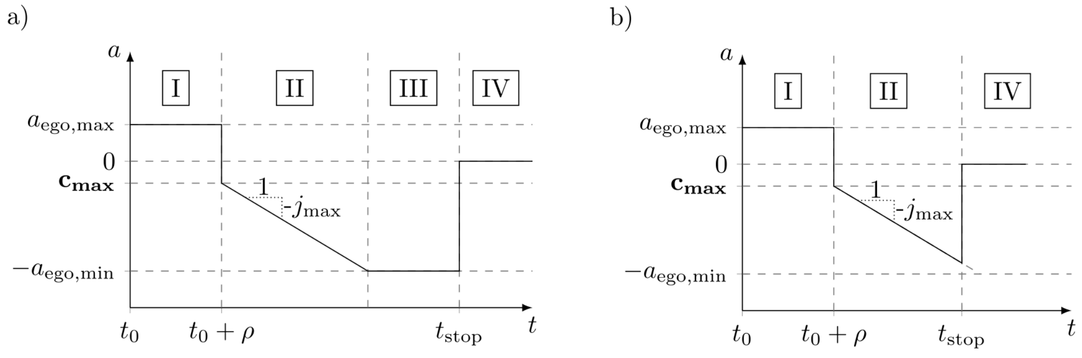


Figure 4.4: For distances above the minimal safe distance $d > d_{\mathrm{RSS}}$, a parameter that relaxes the ego vehicle's proper response of the RSS model is proposed as the upper bound c_{max} for the demanded acceleration. a) With this modification, the deceleration profile of the ego vehicle is divided into I) response time, II) jerk-limited braking, III) full braking and IV) stop phase. b) If the ego vehicle stops before full braking is reached, phase III) is skipped. Figure a) is based on Brosowsky et al. [2021b].

Equation (4.19) defines the upper bound c_{max} of the safe set for $d \leq d_{\mathrm{RSS}}$ according to the RSS model and the minimal safe distance d_{RSS} is given by Equations (4.20) and (4.21). Now, the upper bound c_{max} is extended for $d > d_{\mathrm{RSS}}$ so that a continuous pass over at distance d_{RSS} is achieved. Thereby, sudden braking maneuvers of the ego vehicle when the minimal safe distance d_{RSS} is undershot are replaced by smoother transitions and the safety guarantees of the RSS model remain valid. For a continuous upper bound, c_{max} is defined for $d > d_{\mathrm{RSS}}$ as a parameter that relaxes the full braking phase of the ego vehicle's proper response as visualized in Figure 4.4. Instead of full braking with $-a_{\mathrm{ego,min}}$ after the response time, a deceleration ramp with limited jerk $-j_{\mathrm{max}}$ renders a linear transition from the upper bound $c_{\mathrm{max}} \geq -a_{\mathrm{ego,min}}$ to full braking. The value of c_{max} is determined such that this modified ego acceleration profile just prevents a crash in the case that the front vehicle brakes fully $a_{\mathrm{f}} = -a_{\mathrm{f,min}}$. Analogously to Property 1 and by requiring $c_{\mathrm{max}} \geq -a_{\mathrm{ego,min}}$, it follows that the minimum distance is reached either in the beginning t_0 or when both vehicles stop t_{stop}. In a just prevented crash, the minimum distance is zero and must be reached at t_{stop} because $d(t_0) > d_{\mathrm{RSS}} \geq 0$ has been assumed in the beginning. Consequently, the following equation must be solved for c_{max}:

$$\begin{gathered} \hat{s}_{\mathrm{ego}}(v_{\mathrm{ego}}, c_{\mathrm{max}}) - s_{\mathrm{f}}(v_{\mathrm{f}}) = d, \\ \text{with } c_{\mathrm{max}} \geq -a_{\mathrm{ego,min}} \text{ and } d > d_{\mathrm{RSS}}, \end{gathered} \tag{4.22}$$

and $\hat{s}_{\mathrm{ego}}$ the stopping distance of the modified ego acceleration profile in Figure 4.4. In comparison to the notation s_{ego} without hat for $d \leq d_{\mathrm{RSS}}$, the notation with the hat $\hat{s}_{\mathrm{ego}}$ is used for $d > d_{\mathrm{RSS}}$. The upper bound can be interpreted as follows. If the front vehicle brakes fully, the ego vehicle must reach an acceleration below the upper bound c_{max} within the response time and then a jerk-limited braking is sufficient to avoid a crash. The upper bound converges by construction and as intended to full braking $c_{\mathrm{max}} \to -a_{\mathrm{ego,min}}$ for $d \to d_{\mathrm{RSS}} > 0$. Note, for $c_{\mathrm{max}} = -a_{\mathrm{ego,min}}$ the proper response of the RSS model is obtained. In a post-processing step, the upper bound is clipped to the limit $a_{\mathrm{ego,max}}$ if $c_{\mathrm{max}} > a_{\mathrm{ego,max}}$.

For solving Equation (4.22), the ego vehicle's stopping distance $\hat{s}_{\text{ego}}$ is written as a twice integral of the modified acceleration profile of the ego vehicle $\hat{a}_{\text{ego}}(t, c_{\max})$ over time:

$$\begin{aligned}\hat{v}_{\text{ego}}(t, v_{\text{ego}}, c_{\max}) &= v_{\text{ego}} + \int_{t_0}^{t} \hat{a}_{\text{ego}}(\tilde{t}, c_{\max})\, d\tilde{t}, \\ \hat{s}_{\text{ego}}(v_{\text{ego}}, c_{\max}) &= \int_{t_0}^{t_{\text{stop}}} \hat{v}_{\text{ego}}(t, v_{\text{ego}}, c_{\max})\, dt.\end{aligned} \tag{4.23}$$

The acceleration profile $\hat{a}_{\text{ego}}(\tilde{t}, c_{\max})$ consists of a I) response, II) jerk-limited braking, III) full braking, and IV) stop phase as visualized in Figure 4.4 a). If the ego vehicle stops already in the jerk-limited braking phase before full braking $-a_{\text{ego,min}}$ is reached, phase III) is skipped as visualized in Figure 4.4 b). For solving Equation (4.22), in the following several properties are derived.

Property 2. *In Figure 4.4, the two possible acceleration profiles $\hat{a}_{\text{ego}}(t, v_{\text{ego}}, c_{\max})$ are visualized. Let us call the profile in a) full braking regime and in b) jerk-limited braking regime. For an ego vehicle with velocity v_{ego} and upper bound $c_{\max}$, the regime can be determined by:*

$$regime = \begin{cases} a)\ if\ v'_{\text{ego}} > v_{\text{case}}(c_{\max}), \\ b)\ if\ v'_{\text{ego}} \leq v_{\text{case}}(c_{\max}), \end{cases} \tag{4.24}$$

with $v'_{\text{ego}} = v_{\text{ego}} + a_{\text{ego,max}}\rho$ the ego vehicle's velocity after the response phase and the velocity threshold $v_{\text{case}}(c_{\max}) = \frac{a^2_{\text{ego,min}} - c^2_{\max}}{2 j_{\max}} \geq \frac{a^2_{\text{ego,min}}}{2 j_{\max}}$. Note, for $v_{\text{ego}} > \frac{a^2_{\text{ego,min}}}{2 j_{\max}} - a_{\text{ego,max}}\rho =$ $2.58\,\mathrm{m\,s^{-1}}$ (parameters according to Table 4.2) the ego vehicle is always in the full braking regime independently of $c_{\max}$.

Proof. The corresponding regime can be determined by computing the velocity of the ego vehicle after the response phase followed by jerk-limited braking until $-a_{\text{ego,min}}$ is reached. If the final velocity is greater zero, further braking with $-a_{\text{ego,min}}$ is required and the case a) is identified. Otherwise, the vehicle must have come to a stop previously according to case b). For the jerk-limited braking, the acceleration of the ego vehicle at time $t' = t - (t_0 + \rho)$ can be written as:

$$\hat{a}_{\text{ego}}(t') = c_{\max} - j_{\max} t', \tag{4.25}$$

and full braking is reached at $t'_c = (a_{\text{ego,min}} + c_{\max})/j_{\max}$. The velocity at time t' is obtained by integration:

$$\hat{v}_{\text{ego}}(t') = v'_{\text{ego}} + c_{\max} t' - \frac{1}{2} j_{\max} t'^2, \tag{4.26}$$

with $v'_{\text{ego}} = v_{\text{ego}} + a_{\text{ego,max}}\rho$ the ego vehicle's velocity after the response phase. Substituting t' with t'_c results in the following expression for the velocity at the time when full braking is reached:

$$\hat{v}_{\text{ego}}(t'_c) = v'_{\text{ego}} + \frac{c^2_{\max} - a^2_{\text{ego,min}}}{2 j_{\max}}. \tag{4.27}$$

As explained, regime a) an b) can be assigned depending on the sign of $\hat{v}_{\text{ego}}(t'_c)$:

$$\text{regime } = \begin{cases} \text{a) if } \hat{v}_{\text{ego}}(t'_c) > 0 \Leftrightarrow v'_{\text{ego}} > v_{\text{case}}(c_{\max}), \\ \text{b) if } \hat{v}_{\text{ego}}(t'_c) \leq 0 \Leftrightarrow v'_{\text{ego}} \leq v_{\text{case}}(c_{\max}), \end{cases} \tag{4.28}$$

with the velocity threshold $v_{\text{case}}(c_{\max}) = \frac{a^2_{\text{ego,min}} - c^2_{\max}}{2 j_{\max}}$. □

Property 3. *The stopping distance $\hat{s}_{\text{ego}}$ is a strictly monotonically increasing function of $c_{\max}$.*

Proof. For two upper bounds $c_{\max,1} < c_{\max,2}$, the velocity profile $\hat{v}_{\text{ego}}(t, c_{\max,2})$ of $c_{\max,2}$ is greater equal than the velocity profile $\hat{v}_{\text{ego}}(t, c_{\max,1})$ of $c_{\max,1}$ for a point-wise comparison at $t \geq t_0$:

$$\begin{aligned} c_{\max,1} < c_{\max,2} \text{ and } \mathcal{T} = \{t \geq t_0 : \hat{v}_{\text{ego}}(t, c_{\max,1}) \geq 0 \wedge \hat{v}_{\text{ego}}(t, c_{\max,2}) \geq 0\} \\ \Rightarrow \forall t \in \mathcal{T} : \hat{a}_{\text{ego}}(t, c_{\max,1}) \leq \hat{a}_{\text{ego}}(t, c_{\max,2}) \\ \Rightarrow \forall t \in \mathcal{T} : \hat{v}_{\text{ego}}(t, c_{\max,1}) \leq \hat{v}_{\text{ego}}(t, c_{\max,2}) \\ \Rightarrow t_{\text{ego,stop}}(c_{\max,1}) \leq t_{\text{ego,stop}}(c_{\max,2}) \\ \Rightarrow \forall t \geq t_0 : \hat{v}_{\text{ego}}(t, c_{\max,1}) \leq \hat{v}_{\text{ego}}(t, c_{\max,2}). \end{aligned} \tag{4.29}$$

This implies immediately $\hat{s}_{\text{ego}}(v_{\text{ego}}, c_{\max,1}) \leq \hat{s}_{\text{ego}}(v_{\text{ego}}, c_{\max,2})$. Strict monotony follows from the fact that there is always a small time interval in the beginning of the jerk-limited braking phase with $\hat{a}_{\text{ego}}(t, c_{\max,1}) < \hat{a}_{\text{ego}}(t, c_{\max,2})$. □

Property 4. *The stopping distance $\hat{s}_{\text{ego}}$ is unbounded from above, i.e. $\hat{s}_{\text{ego}}(c_{\max}) \to \infty$ for $c_{\max} \to \infty$.*

Proof. According to Property 2, $c_{\max} \to \infty$ implies that the acceleration profile is in the full braking regime:

$$\begin{aligned} v_{\text{case}}(c_{\max}) \to -\infty \text{ for } c_{\max} \to \infty \\ \Rightarrow v'_{\text{ego}} > v_{\text{case}}(c_{\max}) \Rightarrow \text{ full braking regime.} \end{aligned} \tag{4.30}$$

The braking distance $\hat{s}_{\text{ego,II}}$ of phase II) (jerk-limited braking) can be computed by integration of Equation (4.26) and evaluation when full braking is reached at $t'_c = (a_{\text{ego,min}} + c_{\max})/j_{\max}$:

$$\begin{aligned} \hat{s}_{\text{ego,II}}(t') &= v'_{\text{ego}} t' + \frac{1}{2} c_{\max} t'^2 - \frac{1}{6} j_{\max} t'^3, \\ \hat{s}_{\text{ego,II}}(t'_c) &= v'_{\text{ego}} \frac{(a_{\text{ego,min}} + c_{\max})}{j_{\max}} + \frac{1}{2} c_{\max} \frac{(a_{\text{ego,min}} + c_{\max})^2}{j^2_{\max}} - \frac{1}{6} \frac{(a_{\text{ego,min}} + c_{\max})^3}{j^2_{\max}} \\ &= \frac{c^3_{\max}}{3 j^2_{\max}} + \mathcal{O}(c^2_{\max}). \end{aligned} \tag{4.31}$$

Thus, the braking distance of phase II) is unbounded $\hat{s}_{\text{ego,II}}(c_{\max}) \rightarrow \infty$. This implies that the full braking distance is unbounded $\hat{s}_{\text{ego}}(c_{\max}) \rightarrow \infty$ because the distances of the other braking phases are greater or equal to zero. □

Property 5. *There exists a unique upper bound $c_{\max}$ that solves Equation* (4.22).

Proof. The Equation (4.20) for the minimal safe distance can be rewritten by using $s_{\text{ego}}(v_{\text{ego}}) = \hat{s}_{\text{ego}}(v_{\text{ego}}, c_{\max} = -a_{\text{ego,min}})$. For $d > d_{\text{RSS}}$, this leads to the following inequalities:

$$\begin{aligned} &d_{\text{RSS}} = \max\{0, s_{\text{ego}}(v_{\text{ego}}) - s_{\text{f}}(v_{\text{f}})\} \\ \Rightarrow &d_{\text{RSS}} \geq s_{\text{ego}}(v_{\text{ego}}) - s_{\text{f}}(v_{\text{f}}) = \hat{s}_{\text{ego}}(v_{\text{ego}}, -a_{\text{ego,min}}) - s_{\text{f}}(v_{\text{f}}) \\ \Rightarrow &d > \hat{s}_{\text{ego}}(v_{\text{ego}}, -a_{\text{ego,min}}) - s_{\text{f}}(v_{\text{f}}) \\ \Rightarrow &\exists!\, c_{\max} > -a_{\text{ego,min}} : \; d = \hat{s}_{\text{ego}}(v_{\text{ego}}, c_{\max}) - s_{\text{f}}(v_{\text{f}}). \end{aligned} \tag{4.32}$$

The last implication leverages that $\hat{s}_{\text{ego}}$ is a strictly monotonically increasing function of $c_{\max}$ (Property 3) and not bounded from above (Property 4). □

In the following, Properties 2 to 5 are leveraged to solve Equation (4.22) for the upper bound $c_{\max}$. According to Property 5, there exists a unique solution.

1. If $v_{\text{ego}} > \frac{a^2_{\text{ego,min}}}{2 j_{\max}} - a_{\text{ego,max}}\rho = 2.58\,\text{m}\,\text{s}^{-1}$ holds (parameters according to Table 4.2), Property 2 ensures that full braking is reached. Thus, step 2 can be skipped and $c_{\max}$ is determined in step 3. Otherwise, *i.e.* for $v_{\text{ego}} \leq 2.58\,\text{m}\,\text{s}^{-1}$, step 2 is required.
2. Assuming that the solution is in the jerk-limited braking regime, Equation (4.22) can be solved under the relaxed condition that the jerk-limited braking phase is not bounded from below by $-a_{\text{ego,min}}$. If the solution $c_{\max}$ of the relaxed problem satisfies $v'_{\text{ego}} \leq v_{\text{case}}(c_{\max})$, Property 2 ensures that full braking is indeed not reached and the upper bound is found. Otherwise, the solution of Equation (4.22) must be in the full braking regime and step 3 must be performed.
3. Equation (4.22) is solved for $c_{\max}$ under the knowledge that full braking is reached.

For step 2, Equation (4.22) is solved in the jerk-limited braking regime under the relaxed condition that the acceleration is not bounded from below by $-a_{\text{ego,min}}$. First, the integrals for the velocity and the distance in Equation (4.23) are performed. For the response phase, the following expression is found:

$$\hat{s}_{\text{ego,I}}(v_{\text{ego}}) = v_{\text{ego}}\rho + \frac{1}{2}a_{\text{ego,max}}\rho^2. \tag{4.33}$$

For the jerk-limited braking phase, the acceleration, velocity, and distance are given by:

$$\begin{aligned}\hat{a}_{\text{ego}}(t') &= c_{\max} - j_{\max} t', \\ \hat{v}_{\text{ego}}(t') &= v'_{\text{ego}} + c_{\max} t' - \frac{1}{2} j_{\max} t'^2, \\ \hat{s}_{\text{ego,II}}(t') &= v'_{\text{ego}} t' + \frac{1}{2} c_{\max} t'^2 - \frac{1}{6} j_{\max} t'^3,\end{aligned} \tag{4.34}$$

with $t' = t - (t_0 + \rho)$ and $v'_{\text{ego}} = v_{\text{ego}} + a_{\max}\rho$. The distance until the ego vehicles stops is derived as follows:

$$\begin{aligned}&\hat{v}_{\text{ego}}(t') = 0, \\ \Rightarrow & t'_{\text{ego,stop}} = \frac{c_{\max}}{j_{\max}} + \sqrt{\frac{c^2_{\max}}{j^2_{\max}} + \frac{2v'_{\text{ego}}}{j_{\max}}}, \\ \Rightarrow & \hat{s}_{\text{ego,II}}(v_{\text{ego}}, c_{\max}) = v'_{\text{ego}} t'_{\text{ego,stop}} + \frac{1}{2} c_{\max} t'^2_{\text{ego,stop}} - \frac{1}{6} j_{\max} t'^3_{\text{ego,stop}}.\end{aligned} \tag{4.35}$$

Finally, $c_{\max}$ is determined numerically by solving for the roots of:

$$\hat{s}_{\text{ego,I}}(v_{\text{ego}}) + \hat{s}_{\text{ego,II}}(v_{\text{ego}}, c_{\max}) - s_{\text{f}}(v_{\text{f}}) - d = 0. \tag{4.36}$$

For step 3, Equation (4.22) is solved under the knowledge that full braking is reached. First, the integrals in Equation (4.23) are performed for phase I), II) and III) of the full braking regime. The distance of phase I) is given by Equation (4.33) and the distance of phase II) by Equation (4.31). In phase III), the ego vehicle brakes with $-a_{\text{ego,min}}$ and the braking distance until stopping is:

$$\hat{s}_{\text{ego,III}}(v_{\text{ego}}, c_{\max}) = \frac{\hat{v}^2_{\text{ego}}(t'_c)}{2a_{\text{ego,min}}}, \tag{4.37}$$

$$\hat{v}_{\text{ego}}(t'_c) = v'_{\text{ego}} + \frac{c^2_{\max} - a^2_{\text{ego,min}}}{2j_{\max}}, \tag{4.38}$$

with $\hat{v}_{\text{ego}}(t'_c)$ the ego vehicle's initial velocity in phase III) from Equation (4.27). Finally, the upper bound $c_{\max}$ is determined numerically by solving for the roots of:

$$\hat{s}_{\text{ego,I}}(v_{\text{ego}}) + \hat{s}_{\text{ego,II}}(v_{\text{ego}}, c_{\max}) + \hat{s}_{\text{ego,III}}(v_{\text{ego}}, c_{\max}) - s_{\text{f}}(v_{\text{f}}) - d = 0. \tag{4.39}$$

The left-hand side is a polynomial in $c_{\max}$ of degree four and the roots can be solved with standard numerical solvers.

The lower bound $c_{\min}$ of the state-specific safe set C describes the maximum deceleration. The maximum deceleration is defined according to ISO15622 [2018] and depends on the current velocity of the ego vehicle v_{ego}:

$$c_{\min} = \begin{cases} -5\,\mathrm{m\,s^{-2}} & \text{for } v_{\text{ego}} < 5\,\mathrm{m\,s^{-1}}, \\ -5\,\mathrm{m\,s^{-2}} + (v_{\text{ego}} - 5\,\mathrm{m\,s^{-1}})/10\,\mathrm{s} & \text{for } 5\,\mathrm{m\,s^{-1}} \leq v_{\text{ego}} \leq 20\,\mathrm{m\,s^{-1}}, \\ -3.5\,\mathrm{m\,s^{-2}} & \text{for } v_{\text{ego}} > 20\,\mathrm{m\,s^{-1}}. \end{cases} \tag{4.40}$$

If the upper bound $c_{\max}$ undershoots the lower bound $c_{\min}$, an emergency braking mode is entered and the lower bound is overwritten with the upper bound. Thus, in the emergency braking mode the ego vehicle's demanded acceleration is explicitly given by $a_{\text{ego,dem}} = c_{\max} = c_{\min}$.

4.3.3 Twin Delayed Deep Deterministic Policy Gradient Algorithm

The TD3 algorithm [Fujimoto et al., 2018] is a state-of-the-art RL algorithm for continuous action spaces and a further development of the commonly used and successful DDPG algorithm [Lillicrap et al., 2016]. According to Definitions 1, 2, 3 (see Section 2.3.4), the TD3 and the DDPG algorithm are model-free off-policy actor-critic DRL algorithms. Both algorithms save the interactions with the environment in an experience replay buffer and the same sample may be used multiple times in training. This efficient reuse of samples is only possible because the algorithms are off-policy methods. The TD3 algorithm modifies the DDPG algorithm to address three weak points. First, the overestimation of the action-value function Q^{π} in the DDPG algorithm is tackled by approximating Q^{π} with two NNs and taking the minimum of them. The modification represents the *twin* in TD3. Second, the policy is less frequently updated than the action-value function to stabilize training. This explains the word *delayed* in TD3. Third, overfitting of the deterministic policy to narrow peaks in the action-value function is addressed by regularizing the action-value function. In the following, the common concepts and differences of the DDPG and TD3 algorithm are explained on a technical level. Algorithm 2 shows the pseudo code of the TD3 algorithm.

Both algorithms, TD3 and DDPG, parametrize a deterministic policy by using an NN π_ϕ with parameters ϕ. For policy improvement, a gradient ascent step of the expected discounted return $J(\phi)$ is performed (line 13 in Algorithm 2). The gradient $\nabla J(\phi)$ is estimated with the deterministic policy gradient theorem [Silver et al., 2014]:

$$\nabla J(\phi) \approx \frac{1}{N} \sum_i \nabla_u Q^{\pi_\phi}(x,u)|_{x=x_i, u=\pi_\phi(x_i)} \nabla_\phi \pi_\phi(x_i), \tag{4.41}$$

$$\approx \frac{1}{N} \sum_i \nabla_u Q_\theta(x,u)|_{x=x_i, u=\pi_\phi(x_i)} \nabla_\phi \pi_\phi(x_i). \tag{4.42}$$

In the second equation, the action-value function Q^{π_ϕ} of the current policy π_ϕ is approximated with the NN Q_θ with parameters θ.

Algorithm 2 TD3. The pseudo code has been adapted from Algorithm 1 in Fujimoto et al. [2018].

```
 1: Initialize critic networks $Q_{\theta_1}, Q_{\theta_2}$, and policy $\pi_\phi$ with random parameters $\theta_1, \theta_2, \phi$
 2: Initialize target networks $\theta'_1 \leftarrow \theta_1, \theta'_2 \leftarrow \theta_2, \phi' \leftarrow \phi$
 3: Initialize replay buffer $\mathcal{D}$
 4: for $k = 1$ to $K$ do
 5:     Select action $u \sim \pi_\phi(x) + \epsilon$ with exploration noise $\epsilon \sim \mathcal{N}(0, \sigma)$
 6:     Observe reward $r$ and next state $x'$, and save the transition $(x, u, r, x')$ in $\mathcal{D}$
 7:     Sample a mini-batch of $N$ transitions $\{(x_i, u_i, r_i, x'_i)\}$ from the replay buffer $\mathcal{D}$
 8:     Determine targets:
 9:     $u'_i = \pi_{\phi'}(x'_i) + \epsilon$ and $\epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
10:     $y_i = r_i + \gamma \min_{j=1,2} Q_{\theta'_j}(x'_i, u'_i)$
11:     Update critics $\theta_j$ by gradient descent step w.r.t. $L(\theta_j) = 1/N \sum_i (y_i - Q_{\theta_j}(x_i, u_i))^2$
12:     if $k \mod K_d$ then
13:         Update policy $\pi_\phi$ by gradient ascent step w.r.t. $J(\phi)$:
14:         $\nabla J(\phi) \approx 1/N \sum_i \nabla_u Q_{\theta_1}(x, u)|_{x=x_i, u=\pi_\phi(x_i)} \nabla_\phi \pi_\phi(x_i)$
15:         Update target networks:
16:         $\theta'_j \leftarrow \alpha\theta_j + (1-\alpha)\theta'_j$
17:         $\phi' \leftarrow \alpha\phi + (1-\alpha)\phi'$
18:     end if
19: end for
```

For policy evaluation, the DDPG algorithm approximates an update rule based on the Bellman equation by fitting the action-value function towards sampled targets:

$$Q_\theta(x_i, u_i) \leftarrow y_i = r_i + \gamma Q_{\theta'}(x'_i, \pi_{\phi'}(x'_i)), \tag{4.43}$$

with r_i the obtained reward of the transition (x_i, u_i, x'_i), $Q_{\theta'}$ a previous approximation of the action-value function, and $\pi_{\phi'}$ a previous approximation of the policy. In comparison to the DDPG algorithm, the TD3 algorithm takes additionally the minimum of the two action-value functions (see line 10 in Algorithm 2). The fitting is performed with gradient descent steps of the following loss function (see line 11 in Algorithm 2):

$$L(\theta) = \frac{1}{N} \sum_{i=1}^{N} \big(y_i - Q_\theta(x_i, u_i)\big)^2. \tag{4.44}$$

For the targets y, the so-called target networks $Q_{\theta'}$ and $\pi_{\phi'}$ are used, which are copies of their counterparts Q_θ and π_ϕ with own parameters. The target networks are introduced to adjust the granularity of updates, *i.e.* the number of fitted targets per Bellman update. Either, the parameters of the target networks are set periodically to the parameters of the current networks or a delayed update is implemented with the update rule $\theta' \leftarrow \alpha\theta + (1-\alpha)\theta'$ and $\phi' \leftarrow \alpha\phi + (1-\alpha)\phi'$.

Fujimoto et al. [2018] show that the gradient ascent update $\pi_{\text{up}} = \pi_{\phi+\eta\nabla J}$ of the current policy π_ϕ with an approximated action-value function Q_θ as in Equation 4.42 instead of the unknown true action-value function Q^{π_ϕ} leads to an overestimation of the action-value function. Formally, for the expectation values of the value estimates the inequality $E[Q_\theta(x, \pi_{\text{up}}(x))] \geq E[Q^{\pi_\phi}(x, \pi_{\text{up}}(x))]$ holds under reasonable assumptions. This approximation bias influences the targets y and the Bellman updates. Finally, the overestimation of the action-value function slows down convergence. The TD3 algorithm reduces this overestimation by learning two action-value functions with different weight initializations θ_1, θ_2 (θ'_1 and θ'_2 for target networks) and by taking the minimum:

$$y_i = r_i + \gamma \min_{j=1,2} Q_{\theta'_j}(x'_i, \pi_{\phi'}(x'_i)). \tag{4.45}$$

This approach is called clipped double Q-learning in Fujimoto et al. [2018]. A second modification reduces the bias in the action-value function by a less frequent update of the policy and target networks. Thereby, the error of the value estimate *w.r.t.* the current policy is reduced by more policy evaluation steps. The target networks and the policy are updated only after K_d steps instead of every step (line 12-17 in Algorithm 2). In a third modification, the action-value function is regularized to reduce overfitting of the policy to narrow peaks. Smoothing is implemented by adding small random noise ϵ to the actions of the target policy (line 9 and 10 in Algorithm 2):

$$y_i = r_i + \gamma \min_{j=1,2} Q_{\theta'_j}(x'_i, \pi_{\phi'}(x'_i) + \epsilon) \text{ with } \epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c). \tag{4.46}$$

This smoothing exploits the fact that similar actions should have similar values.

In Fujimoto et al. [2018], the DDPG and the TD3 algorithm are compared on a number of continuous control problems. In the experiments, the TD3 algorithm outperforms the DDPG algorithm in most of the tasks significantly in performance.

4.4 Experiments

4.4.1 Simulator and Reward Function

Simulator

Figure 4.5 shows the graphical user interface of the simulator. For a given policy π and a random initial state x_0, the dynamics of the vehicle following behavior are computed and visualized. A 3D animation is rendered to get a qualitative feedback for the dynamics and plots show the relevant quantities over time.

The simulator generates a variety of vehicle following scenarios: front vehicle with constant and varying speed, stop-and-go traffic, lane changes of the leading vehicle, and cutting-in vehicles. First, the initial state $x_0 = (v_{\text{ego},0}, a_{\text{ego},0}, v_{\text{f},0}, a_{\text{f},0}, d_0)^\intercal$ is sampled randomly under the constraint that $v_{\text{ego/f},0} \geq 0$ and $d_0 > d_{RSS}(v_{\text{ego}}, v_{\text{f}})$ is satisfied. The state variables are defined in Equation (4.12) and d_{RSS} is the minimal safe distance according to the RSS model and Equation (4.20). Next, random acceleration profiles of the front vehicle are generated corresponding to driving with constant and varying speed and stop-and-go traffic. Instead of sampling the acceleration immediately, a demanded acceleration is generated and a first-order lag applied. The transfer function of the first-order lag is given by $G(s) = 1/(1+\tau_{\text{f}} s)$ and the same time constant τ_{f} for braking and acceleration is used. This kind of smoothing corresponds to usual driving with limited jerk. In principle, it would

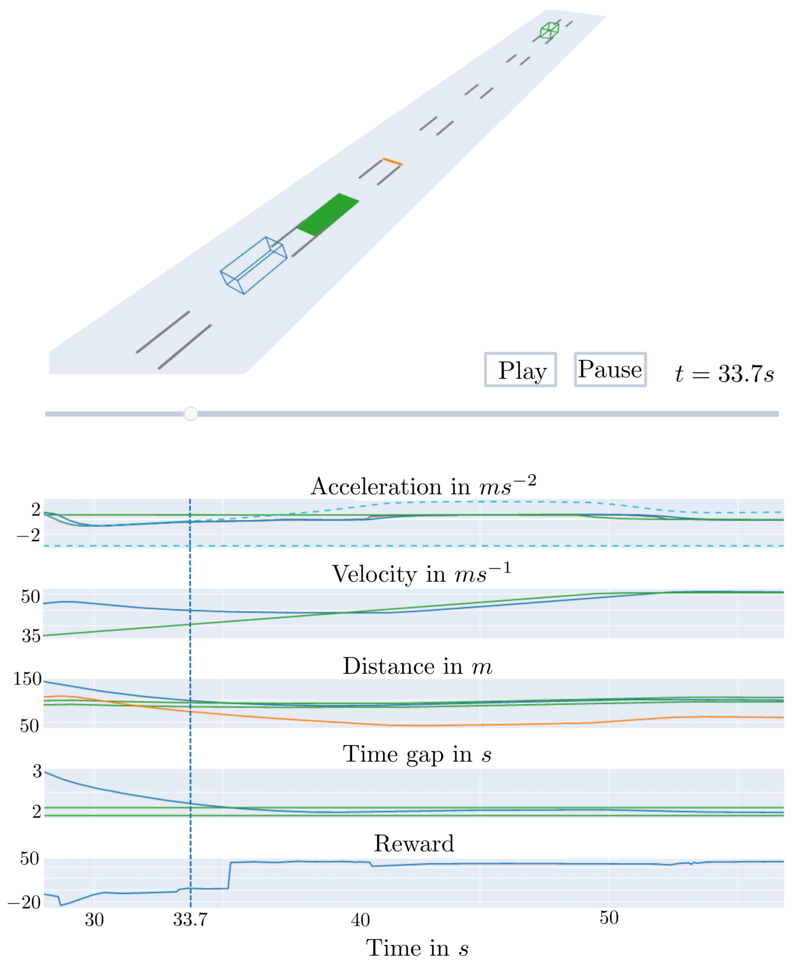


Figure 4.5: Simulator: A variety of trajectories are randomly generated for the front vehicle and the behavior of the vehicle following policy π is simulated. In the depicted scenario, the ego vehicle approaches the accelerating front vehicle. *Top*: Screenshot of the 3D animation. The green filled rectangle represents the optimal distance according to the set time gap and the highest reward and the orange line the minimal safe distance. *Bottom*: The plots show relevant quantities over time. Blue lines visualize quantities that are related to the ego vehicle, gray lines the demanded acceleration of the ego vehicle, green lines quantities that are related to the front vehicle or the optimal values, the orange line is for the minimal safe distance, and the dashed lines represent the constraints for the demanded acceleration. The figure is based on Brosowsky et al. [2021b].

also be possible to use recorded data for the trajectories of the front vehicle. For given front vehicle's acceleration profiles $a_\mathrm{f}(k)$, the state transitions are described by Equations (4.13) to (4.17) apart from the following small modifications. First, it is assumed that the ego vehicle's low-level controller for setting the demanded acceleration prevents reversing. Additionally, the ODD is restricted to only forward driving front vehicles. Technically, driving in forward direction is ensured by setting the velocity and acceleration of the front and ego vehicle to zero $v_{\mathrm{ego/f}} = 0\,\mathrm{m\,s^{-1}}$ and $a_{\mathrm{ego/f}} = 0\,\mathrm{m\,s^{-2}}$ if the demanded acceleration would lead to driving backwards. Second, for the simulation of cut-in and -out scenarios the appearance of a new front vehicle is modeled with a probability p_f. The probability $p_\mathrm{f} = dt/120\,\mathrm{s}$ is chosen so that a cut-in or -out scenario occurs on average every 2 min. For a cutting-in or -out vehicle, a new distance $d > d_{\mathrm{RSS}}(v_{\mathrm{ego}}, v_\mathrm{f})$ and velocity of the front

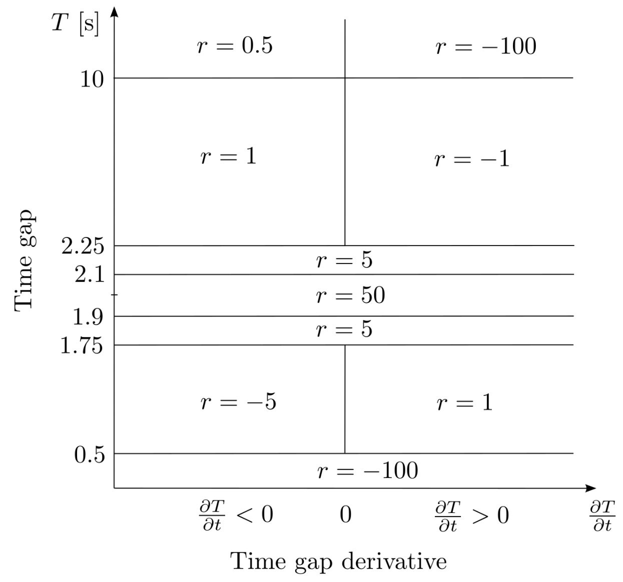


Figure 4.6: The reward function assigns high rewards r if the time gap T is close to the set time gap $T_{\text{set}} = 2$ s and if the time gap derivation $\partial T/\partial t$ indicates a reduction of the time gap deviation. The figure has been adapted from Desjardins and Chaib-draa [2011].

vehicle $v_{\text{f}} \geq 0$ is sampled. Finally, a terminal state is defined for a crash $d = 0$ m. The chosen values for the parameters of the simulator are summarized in Table 4.1.

Reward Function

The reward function is designed for safe, accurate, and comfortable driving. For safe and accurate vehicle following behavior, the reward function in Desjardins and Chaib-draa [2011] is reused. In a second step, a comfort term is added. Desjardins and Chaib-draa [2011] propose a discrete reward function that depends on the time gap T and its derivation in time. The time gap is computed from the current state x_k according to Equation (4.10) and the time gap derivation from two subsequent time gaps:

$$\begin{aligned} T_k &= d_k/(v_{\text{ego},k} + v_{\text{ego,corr},k}), \qquad (4.47) \\ \left(\frac{\partial T}{\partial t}\right)_k &= (T_{k+1} - T_k)/dt. \end{aligned}$$

High rewards are assigned to distances close to the set time gap and to time gap derivations that correspond to a decrease in the time gap error. For the choice of $T_{\text{set}} = 2$ s, Figure 4.6 visualizes the chosen values of the reward function [Desjardins and Chaib-draa, 2011]. The highest reward is assigned if the time gap is in a narrow band $1.9\,\text{s} \leq T \leq 2.1$ s around the set time gap. These optimal time gaps correspond to distances in the green filled rectangle of Figure 4.5.

Parameter	Value	Description
K_{total}	1×10^6	Total number of time steps in training.
K_{start}	1×10^4	Number of initial time steps with random actions to fill experience replay buffer.
K_{eval}	5×10^3	Number of time steps between evaluation of learned behavior.
K_{d}	2	Number of time steps between policy improvement steps (delayed update).
N_{buffer}	1×10^6	Size of replay buffer in number of transitions.
N_{neurons}	256	Number of neurons in hidden layers (actor, critic).
N_{batch}	256	Batch size used for training.
l	3×10^{-4}	Learning rate for Adam optimizer [Kingma and Ba, 2014] for actor and critic.
γ	0.99	Discount factor.
α	5×10^{-3}	Factor for soft-update of target-networks.
σ	$0.35\,\mathrm{m\,s^{-2}}$	Standard deviation of exploration noise.
$\widetilde{\sigma}$	$0.7\,\mathrm{m\,s^{-2}}$	Standard deviation of target policy noise.
c	$1.75\,\mathrm{m\,s^{-2}}$	Maximum target policy noise.

Table 4.3: Parameters of the TD3 algorithm.

Next, the reward function is extended to favor comfortable driving. An additional reward term r_{comfort} penalizes high values of the demanded acceleration and jerk:

$$r_{\text{comfort}}(x_k, u_k, x_{k+1}) = -w_a \cdot |a_{\text{ego,dem},k}| - w_j \cdot |a_{\text{ego},k+1} - a_{\text{ego},k}|/dt, \tag{4.48}$$

with weighting factors $w_a, w_j > 0$ for the absolute values of the acceleration and jerk. In Engel and Babuska [2014], absolute values show reduced steady-state errors compared to quadratic ones. In Section 4.4.3, different values of the weighting factors w_a, w_j are evaluated.

4.4.2 Training

The policies are optimized with the TD3 algorithm [Fujimoto et al., 2018], which is a DRL algorithm for continuous action spaces. The TD3 algorithm approximates the action-value function and a deterministic policy with separate NNs, the Q-network $Q_\theta(x, u)$ and the policy network $\pi_\phi(x)$, respectively. The parameters of the NNs are denoted with θ and ϕ. While the TD3 algorithm is explained in detail in Section 4.3.3, the learning scheme can be summarized shortly with the principle of general policy improvement. The policy network $\pi_\phi(x)$ is updated to improve the average return by leveraging the Q-network $Q_\theta(x, s)$ and *vice versa* the Q-network is updated to be consistent with the improved policy. Contrary to the common DDPG algorithm, a second Q-network and a second target Q-network with different initializations are added. The idea is to reduce the overestimation of the action-value function by using the minimum of both target Q-networks for the training updates of the current Q-networks. A second modification is the delayed update of the policy network in only every second time step ($K_d = 2$). This stabilizes the policy evaluation. In Table 4.3, the parameters of the TD3 algorithm and the chosen values are shown.

Contrary to the original TD3 algorithm, state-specific safe sets $C(s) = [c_{\min}, c_{\max}]$ are imposed as hard output constraints on the policy network π_ϕ as visualized in Figure 4.2. Safe exploration is ensured by restricting the output of the behavior policy to the safe set. The state-specific safe sets are derived in Section 4.3.2 and define an interval of safe actions with lower bound $c_{\min}$ and upper bound $c_{\max}$. $C(s)$ denotes the interval $[c_{\min}, c_{\max}]$ and s is the constraint parameter, *i.e.* the vector $(c_{\min}, c_{\max})^\intercal$ that defines the safe set. State-specific constraints are characterized by the fact that the constraint parameter $s(x)$ is a function of the current state x. If in each time step k the demanded acceleration of the ego vehicle $u_k = a_{\text{ego,dem},k}$ is within the safe set $u_k \in C(s_k)$, rear-end collisions are avoided by guarantee. In the original implementation of the TD3 algorithm[1], the policy network is a conventional unconstrained NN with three fully connected layers. For imposing the safe sets as hard output constraints, clipping is applied as post-processing (clipped NN), clipping is implemented with a differentiable layer (projection), and ConstraintNet is leveraged. ConstraintNet ensures constraint satisfaction by applying the following constraint guard layer:

$$\hat{y}(s, z) = \sigma_1(z)c_{\min} + \sigma_2(z)c_{\min}, \tag{4.49}$$

with $z \in \mathbb{R}^2$ the unconstrained input, σ the two-dimensional softmax function, and $s = (c_{\min}, c_{\max})^\intercal$ the constraint parameter. Thus, the constraint guard layer computes one weight for the upper and lower bound and generates the output by taking the weighted average of the bounds. Furthermore, the input of ConstraintNet is extended by the constraint parameter s to consider the dependency of the intermediate variable z on s. For the sake of fairness, the input of all policies is defined as the normalized state and constraint parameter $(\tilde{x}^\intercal, \tilde{s}^\intercal)^\intercal$.

All policies are trained with the TD3 algorithm for $K_{\text{total}} = 10^6$ training steps. This corresponds to almost 28 h of driving in the simulation with a discretization of $dt = 0.1$ s between subsequent time steps. Furthermore, the maximum duration of an episode is set to 5 min. For the comfort term of the reward function, the weights $w_\text{a} = 6\,\text{s}^2\text{m}^{-1}$ for penalizing large absolute values of the demanded acceleration and $w_\text{j} = 14\,\text{s}^3\text{m}^{-1}$ for penalizing large absolute values of the jerk are chosen. Empirically, these values are found to take comfort sufficiently into account. Furthermore, the standard deviation of the exploration noise is chosen as $\sigma = 0.35\,\text{ms}^{-2}$ and the network size of the policy is specified with $N_{\text{neurons}} = 256$ neurons per layer. Additionally, ConstraintNet is evaluated for different combinations of the comfort weights w_a and w_j, different values of the exploration noise, and different number of neurons per layer. Except for these variations, the standard parameter choice of the TD3 algorithm and the simulator is summarized in Table 4.1 and Table 4.3.

For the standard training configuration, Figure 4.7 shows the average episode return over training time for the different policies. ConstraintNet, the projection-based approach, and the clipped unconstrained policy converge faster and reach a higher average return than the unconstrained policy. ConstraintNet learns most stable and has the steepest learning curve. The projection-based approach has the second fastest convergence, followed by the clipped NN. The unconstrained NN converges slowest. This behavior can be explained by the importance of gradients for learning. ConstraintNet embeds the output constraints in the NN architecture and the gradient of the constraint guard layer (see Equation 4.49)

[1] `https://github.com/sfujim/TD3`, accessed on 10/10/2022

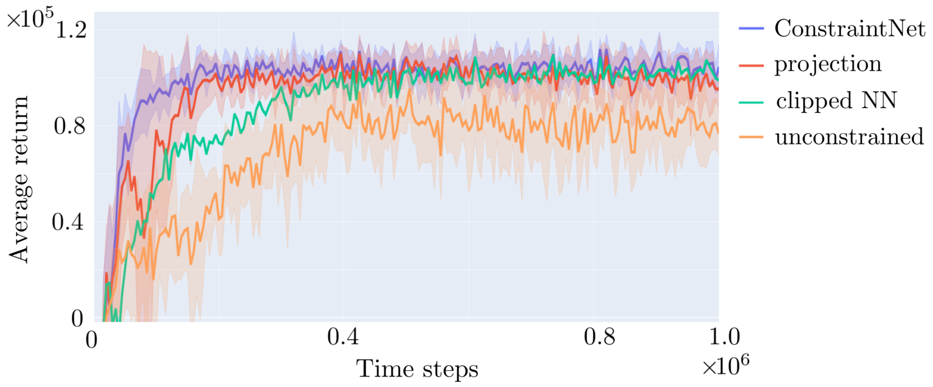


Figure 4.7: Average not discounted episode return over training time for ConstraintNet, the projection-based, the clipped, and the unconstrained policy. Mean and standard deviation are determined over six agents trained with different initializations. ConstraintNet learns fastest and is most stable. The projection-based approach converges slightly slower than ConstraintNet followed by the clipped unconstrained policy. The unconstrained policy requires more time steps for convergence and reaches a lower average return than the policies with imposed safe sets. The figure is based on Brosowsky et al. [2021b].

w.r.t. its input z is a continuous function. The projection-based approach applies clipping as a final layer and has gradients of zero outside of the safe interval. One explanation is that gradients of zero are less informative and less optimal for training. Finally, the policy with clipping as post-processing step and the unconstrained NN converge slowest and their gradients are independent of the safety constraints.

Figure 4.8 shows the crash rate over training time. Only unconstrained policies are plotted since the safe sets ensure collision avoidance completely. Thus, the constrained policies have a constant crash rate of zero. For the unconstrained policy with a comfort term in the reward function, the crash rate decreases slowly from over 60% to roughly 10% after full training. For comparison, the crash rate of an unconstrained policy that is trained without a comfort term in the reward function ($w_\mathrm{a} = w_\mathrm{j} = 0$) is shown. Without the comfort term, the unconstrained policy improves collision avoidance and achieves a crash rate of almost zero. However, crashes still occur rarely. This demonstrates the drawback of soft constraints. If the reward function is supposed to enforce constraints, a trade-off between performance and constraint satisfaction must be chosen. In the experiments, reducing collisions comes at the expense of reduced comfort. Contrary, imposing the safe sets ensures safe training without any collisions and a comfort term is included as well. Moreover, the safe sets improve training performance and stability.

4.4.3 Results

Evaluation Metrics

Several metrics are defined to quantify safety, the accuracy, and the comfort. For each agent, the metrics are computed over 100 episodes with a duration of maximally 5 min. For each training configuration, six agents with different random initializations of the NN

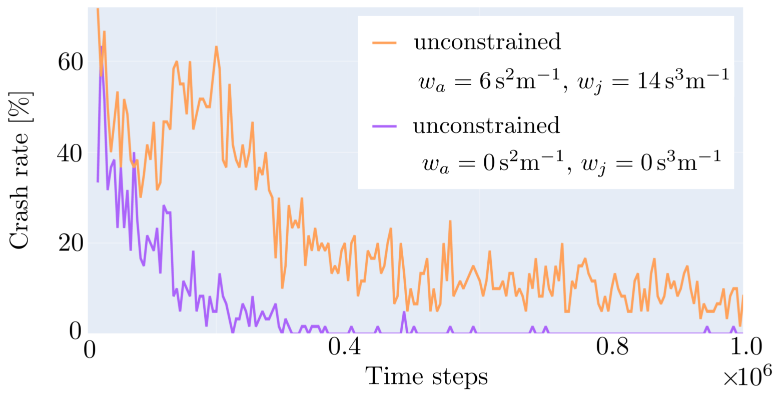


Figure 4.8: Crash rate over training time of the original unconstrained policy. In the reward function, comfortable driving is considered by choosing $w_\mathrm{a} = 6\,\mathrm{s^2m^{-1}}$ as weight for costs on the demanded acceleration and $w_\mathrm{j} = 14\,\mathrm{s^3m^{-1}}$ as weight for costs on the jerk (orange line). Since the crash rate decreases poorly, a second unconstrained policy without the comfort terms is trained, *i.e.* $w_\mathrm{a} = 0\,\mathrm{s^2m^{-1}}$ and $w_\mathrm{j} = 0\,\mathrm{s^3m^{-1}}$ (purple line). In this case, the crash rate drops fast. However, still crashes occur rarely. For the policies with imposed safe sets, the crash rate is zero over the total training time. The figure is based on Brosowsky et al. [2021b].

weights are trained. After training, the mean and the standard deviation of each metric are determined.

The *crash rate* is defined as the number of episodes τ that terminate with a crash ($d = 0\,\mathrm{m}$) divided by the total number of evaluated episodes:

$$M_\mathrm{CrashRate} = \frac{\#\{\tau | d_k = 0\}}{\#\{\tau\}}. \tag{4.50}$$

The *mean time gap error* measures the tracking accuracy:

$$M_{|\delta T|} = \langle |T_\mathrm{set} - T| \rangle, \tag{4.51}$$

with T the actual and T_set the set time gap. For evaluation of the comfort, the *mean absolute acceleration* and the *mean absolute jerk* of the ego vehicle are computed:

$$M_{|a_\mathrm{ego}|} = \langle |a_\mathrm{ego}| \rangle, \tag{4.52}$$
$$M_{|j_\mathrm{ego}|} = \langle |j_\mathrm{ego}| \rangle. \tag{4.53}$$

Comparison of Approaches and the Impact of Exploration Noise, Network Size, and Reward Weights

In this section, the unconstrained policy and the three policies with imposed state-specific safe sets are evaluated, *i.e.* the clipped NN, the projection-based approach, and ConstraintNet. Furthermore, the impact of the weights for comfort, of the exploration noise, and of the network size on ConstraintNet is analyzed. The approaches are compared *w.r.t.* evaluation metrics for safety, accuracy, and comfort and Table 4.4 shows the results. The

Policy	w_a $[\mathrm{s^2m^{-1}}]$	w_j $[\mathrm{s^3m^{-1}}]$	σ (exploration) $[\mathrm{ms^{-2}}]$	N_{neurons}	$M_{\lvert\delta T\rvert}$ [s]	$M_{\lvert a_{\text{ego}}\rvert}$ $[\mathrm{ms^{-2}}]$	$M_{\lvert j_{\text{ego}}\rvert}$ $[\mathrm{ms^{-3}}]$	$M_{\text{CrashRate}}$ [%]
unconstrained	6	14	0.35	256	1.29 ± 0.49	**0.32 ± 0.05**	**0.07 ± 0.01**	7 ± 11
clipped NN	6	14	0.35	256	**0.52 ± 0.20**	0.35 ± 0.02	0.09 ± 0.01	0.0
projection	6	14	0.35	256	0.82 ± 0.49	0.33 ± 0.02	0.08 ± 0.01	0.0
ConstraintNet	6	14	0.35	256	**0.52 ± 0.16**	0.35 ± 0.01	0.08 ± 0.01	0.0
ConstraintNet	6	14	0.1	256	0.86 ± 0.36	0.33 ± 0.02	0.10 ± 0.03	0.0
ConstraintNet	6	14	0.35	256	**0.52 ± 0.16**	0.35 ± 0.01	0.08 ± 0.01	0.0
ConstraintNet	6	14	0.7	256	1.15 ± 0.31	**0.32 ± 0.01**	**0.08 ± 0.01**	0.0
ConstraintNet	6	14	1.4	256	1.07 ± 0.35	0.35 ± 0.02	0.09 ± 0.01	0.0
ConstraintNet	6	14	0.35	64	2.16 ± 1.09	**0.32 ± 0.07**	0.14 ± 0.09	0.0
ConstraintNet	6	14	0.35	256	0.52 ± 0.16	0.35 ± 0.01	**0.08 ± 0.01**	0.0
ConstraintNet	6	14	0.35	512	**0.44 ± 0.06**	0.35 ± 0.01	**0.08 ± 0.01**	0.0
ConstraintNet	0	0	0.35	256	**0.26 ± 0.04**	0.44 ± 0.02	0.73 ± 0.17	0.0
ConstraintNet	0	14	0.35	256	0.34 ± 0.06	0.39 ± 0.02	0.10 ± 0.01	0.0
ConstraintNet	0	28	0.35	256	0.47 ± 0.13	0.37 ± 0.02	0.08 ± 0.01	0.0
ConstraintNet	6	0	0.35	256	0.48 ± 0.07	0.36 ± 0.01	0.19 ± 0.03	0.0
ConstraintNet	6	14	0.35	256	0.52 ± 0.16	0.35 ± 0.01	0.08 ± 0.01	0.0
ConstraintNet	6	28	0.35	256	0.74 ± 0.48	0.36 ± 0.03	0.07 ± 0.01	0.0
ConstraintNet	12.5	0	0.35	256	1.18 ± 0.24	0.30 ± 0.01	0.14 ± 0.02	0.0
ConstraintNet	12.5	14	0.35	256	2.17 ± 0.50	**0.27 ± 0.01**	0.06 ± 0.01	0.0
ConstraintNet	12.5	28	0.35	256	1.98 ± 1.00	0.29 ± 0.03	**0.05 ± 0.01**	0.0

Table 4.4: Results of the different policies and of the ConstraintNet-based policy depending on exploration noise, network size, and reward weights.

hyperparameters for the training are set according to Table 4.1 and Table 4.3 and the comfort weights, the exploration noise, and the network size is specified in the columns of Table 4.4.

The first block of Table 4.4 shows the evaluation metrics for the four approaches. While the comfort metrics are similar between the approaches, the time gap accuracy varies. ConstraintNet and the clipped NN achieve the best balance between time gap error and comfort. However, ConstraintNet converges in training fastest as shown in the previous section. The time gap error of the unconstrained NN is more than twice as high compared to ConstraintNet and crashes with the front vehicle occur on average in seven of the 100 evaluation episodes. Moreover, the formal guarantee of the safe sets *w.r.t.* collision avoidance is verified. All policies with imposed safe sets show a constant crash rate of zero.

For further analysis, ConstraintNet is trained with different hyperparameters for the standard deviation of the exploration noise ($\sigma = 0.1/0.35/0.7/1.4\,\mathrm{ms^{-2}}$), the number of neurons per layer ($N_{\text{neurons}} = 64/256/512$), and the weights of the comfort term ($w_a = 0/6/12.5\,\mathrm{s^2m^{-1}}$ and $w_j = 0/14/28\,\mathrm{s^3m^{-1}}$). All other hyperparameters are not modified. The evaluation metrics of the different experiments are shown in the lower three blocks of Table 4.4 and are interpreted as follows. Regarding the exploration noise, the best performance is achieved for $\sigma = 0.35\,\mathrm{ms}^{-2}$. The result indicates that this setting leads to an optimal exploration-exploitation trade-off. In the next block, different NN sizes are evaluated. A reduced number of neurons per layer from 256 to 64 increases the mean time gap error significantly to roughly 2 s. Probably, the capacity of the NN with 64 neurons per layer is not sufficient to deal with the task complexity. Contrary, doubling the number of neurons per layer from 256 to 512 improves the mean time gap error further without downgrading the comfort metrics. In the last block of Table 4.4, different weights for the comfort term in the reward function are compared. The set of evaluated comfort weights covers the full range from no comfort to a suitable trade-off between comfort and time gap accuracy to a high comfort at the expense of a low time gap accuracy. The results show an expected behavior. Increasing the costs for the demanded acceleration with the weight w_a reduces the mean absolute acceleration. Analogously, increasing the costs for the jerk with the weight w_j reduces the mean absolute jerk. However, improved comfort metrics correspond to a higher mean time gap error. For this trade-off between time gap accuracy and comfort, the weights $w_\mathrm{a} = 6\,\mathrm{s}^2\mathrm{m}^{-1}$ and $w_\mathrm{j} = 14\,\mathrm{s}^3\mathrm{m}^{-1}$ are considered as optimal.

4.5 Conclusion

Deep reinforcement learning is a promising candidate for solving continuous control tasks of driving automation systems. However, safety concerns limit the application in safety-critical systems. To address this, the TD3 algorithm has been applied for learning a vehicle following controller and state-specific safe sets have been imposed as constraints on the control input. For the safe sets, the responsibility-sensitive safety model has been leveraged. The model proposes an emergency braking request if a minimal safe distance is undershot. For distances close to the minimal safe distance, the model has been continuously extended. Thereby, collision avoidance is still ensured by the responsibility-sensitive safety model. However, the extended continuous upper bound intervenes already before the minimal safe distance is undershot and thereby harsh braking requests are reduced. To impose the safe sets on the policy's neural network, three approaches have been evaluated, *i.e.* ConstraintNet, clipping as post-processing, and a projection-based approach that performs clipping as a final layer. Consistent with the theory of the safe sets, all three approaches have reduced the crash rate to zero. Moreover, ConstraintNet and the clipped unconstrained policy have shown the best performance with respect to balancing the mean time gap error and the two comfort metrics. The most stable training behavior and the fastest convergence has been achieved with ConstraintNet. This suggests an efficient embedding of constraints. In conclusion, the effectiveness and practicability of constraints on the control input has been demonstrated successfully for the safety assurance of driving policies.

5 Behavior Prediction for Safe Driving with Constrained Neural Networks: Joint Vehicle Trajectory and Cut-In Prediction

Contents

5.1 Motivation 110

5.2 Related Work 113

5.3 Methods 116

5.3.1 Data Set and Automatic Scenario Detection 116

5.3.2 Model Input 123

5.3.3 Encoder-Decoder Architecture 125

5.3.4 Soft and Hard Output Constraints 128

5.3.5 Loss 129

5.4 Experiments 131

5.4.1 Evaluation Metrics 131

5.4.2 Preprocessing 134

5.4.3 Training 138

5.4.4 Results 139

5.5 Conclusion 148

Defensive and responsible driving like keeping sufficient large distances to preceding vehicles is crucial for road safety. However, in practice not all road-users behave accordingly. *E.g.* on highways, vehicles on the neighboring lane may cut-in recklessly. Thus, Chapter 5.1 motivates the necessity of an early and reliable behavior prediction of surrounding vehicles for safe ADSs and ADASs to mitigate the risk. Chapter 5.2 provides an overview of known and related maneuver and trajectory prediction models. In Chapter 5.3, the model in Deo and Trivedi [2018b] is extended and a joint vehicle trajectory and cut-in prediction [Brosowsky et al., 2021c] is proposed for real-world application. LSTMs are leveraged to predict the parameters of a bimodal distribution over future trajectories and soft and hard output constraints are evaluated to improve mode separation. In Chapter 5.4, experiments are performed on a large and diverse data set, which is created from measurements of a vehicle fleet. Chapter 5.5 concludes that the results show great potential to improve the safety and the comfort of DASs.

Parts of this chapter have previously appeared in the publication *Joint Vehicle Trajectory and Cut-In Prediction on Highways using Output Constrained Neural Networks* [Brosowsky et al., 2021c].

5.1 Motivation

A performant behavior prediction of surrounding vehicles is crucial for the safety, the comfort, and the efficiency of DASs. In AVs, motion prediction enables an optimized trajectory planning similar to a human driver. Thus, behavior prediction has become an elementary part of an AV's software stack [Bansal et al., 2019; Deo and Trivedi, 2018b]. Attentive drivers observe the traffic, recognize temporal patterns, anticipate other road-user's future behavior, and adapt the driving policy appropriately. For ADASs, behavior prediction is crucial for an intelligent and predictive control behavior. *E.g.* ACC controls a vehicle's longitudinal dynamics and sets a desired velocity-dependent distance to the front vehicle. Anticipating other vehicles' lane changes [Schmüdderich et al., 2015; Wirthmüller et al., 2020] enables an early reaction to the relevant vehicle. Thus, uncomfortable or even hazardous braking maneuvers can be avoided.

In principle, a behavior prediction can be implemented on the level of exact trajectories or on the coarser level of semantically interpretable maneuvers. Trajectories comprise exact information about space and time by definition. However, motion prediction is inherently uncertain and a probabilistic view is required. Two main factors of uncertainty are missing knowledge about the driving policies of surrounding vehicles and an only partially observable environment. Nevertheless, vehicle motion shows common and repeating patterns in the form of semantically interpretable maneuvers, *e.g.* lane changes, following objects, or overtake maneuvers.

Based on these observations, in this chapter a bimodal trajectory prediction is proposed and the modes are semantically associated with a cut-in ($m = \mathrm{c}$) and a passing ($m = \mathrm{p}$) maneuver, respectively. The model builds on and modifies the approach of Deo and Trivedi [2018b]. For each vehicle of interest, the motion prediction is performed separately. According to the ODD, specific rules are applied and decide whether a vehicle is relevant for prediction. In the following, the object under consideration is called target vehicle. Furthermore, a cut-in maneuver of the target vehicle is defined as a future lane change from a neighboring lane to the ego vehicle's lane, hereafter referred to as ego lane, so that the target vehicle becomes the Closest In-Path Vehicle (CIPV). In comparison, passing maneuvers are defined as the complement of cut-ins, *i.e.* continued lane-keeping, lane changes to a different than the ego lane, or lane changes to the ego lane in front of the CIPV. For the future trajectory, the variable $y_k = (y_{k,1}, \dots, y_{k,L})$ is introduced with k the time step of prediction and $y_{k,l}$ the 2D coordinates of the target vehicle at time step $k+l$ in the reference frame of the target vehicle. To support readability, the subscript is skipped occasionally. The trajectories are predicted $T_L = 3$ s in the future with a step size of $dt = 0.1$ s between subsequent time steps. The model output is a distribution over the target vehicle's future trajectory and maneuver

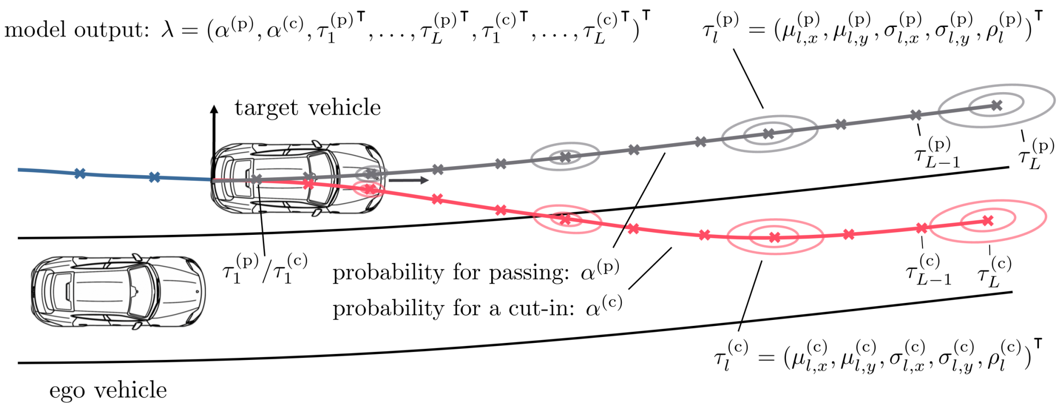

model output: $\lambda = (\alpha^{(p)}, \alpha^{(c)}, \tau_1^{(p)\intercal}, \dots, \tau_L^{(p)\intercal}, \tau_1^{(c)\intercal}, \dots, \tau_L^{(c)\intercal})^\intercal$
$\tau_l^{(p)} = (\mu_{l,x}^{(p)}, \mu_{l,y}^{(p)}, \sigma_{l,x}^{(p)}, \sigma_{l,y}^{(p)}, \rho_l^{(p)})^\intercal$
target vehicle
$\tau_{L-1}^{(p)}$
$\tau_L^{(p)}$
$\tau_1^{(p)}/\tau_1^{(c)}$
probability for passing: $\alpha^{(p)}$
probability for a cut-in: $\alpha^{(c)}$
$\tau_{L-1}^{(c)}$
$\tau_L^{(c)}$
$\tau_l^{(c)} = (\mu_{l,x}^{(c)}, \mu_{l,y}^{(c)}, \sigma_{l,x}^{(c)}, \sigma_{l,y}^{(c)}, \rho_l^{(c)})^\intercal$
ego vehicle

Figure 5.1: The output of the model $\lambda = f_\theta(x)$ are parameters of a bimodal distribution $p(y, m|\lambda)$ over the target vehicle's future trajectory y with two modes $m = \text{p}/\text{c}$. The modes are semantically associated with a cut-in (m=c, red) and a passing maneuver (m=p, gray) with probabilities $\alpha^{(c)}$ and $\alpha^{(p)}$, respectively. The probability distribution over the future trajectory for a given mode m is modeled as product of individual bivariate Gaussians. Each Gaussian models the spatial uncertainty of a waypoint at time step $l = 1, \dots, L$. The parameters of the bivariate Gaussians consist of the mean values and the standard deviations in x- and y-direction, and the correlation coefficient $\tau_l^{(m)} = (\mu_{l,x}^{(m)}, \mu_{l,y}^{(m)}, \sigma_{l,x}^{(m)}, \sigma_{l,y}^{(m)}, \rho_l^{(m)})^\intercal$. Thereby, different uncertainties in x- and y-direction and increasing uncertainties for waypoints further in the future are considered. For illustration purposes, the uncertainties of only every third waypoint are indicated.

$p(y, m|x, \theta)$, with x the input and θ the model parameters. As visualized in Figure 5.1, the model predicts the parameters $\lambda = f_\theta(x)$ of the bimodal distribution $p(y, m|\lambda)$. From this perspective, the actual driven future trajectory $y = y_k$ and maneuver $m = m_k$ of the target vehicle are realizations of the corresponding random variables. Bayes theorem allows to write the likelihood as:

$$p(y, m|x, \theta) = p(y|m, x, \theta)p(m|x, \theta), \tag{5.1}$$

with $x = x_k = (x_{k,-H}, \dots, x_{k,0})$ a sequence of input features, *i.e.* $x_{k,-h}$ is a feature vector *w.r.t.* time step k-h. In the most general form, the discrete distribution $p(m|x, \theta)$ is a categorical distribution:

$$p(m|x, \theta) = p\big(m|\alpha = f_\theta^{(\alpha)}(x)\big), \tag{5.2}$$

with parameters $\alpha = (\alpha^{(p)}, \alpha^{(c)})^\intercal$ ($\sum_m \alpha^{(m)} = 1$) that are predicted by the model $f_\theta^{(\alpha)}$. Furthermore, the trajectory distribution for a given mode $p(y|m, x, \theta)$ is modeled to be within a family of parametric distributions:

$$p(y|m, x, \theta) = p\big(y|\tau^{(m)} = f_\theta^{(m)}(x)\big), \tag{5.3}$$

with parameters $\tau^{(m)}$ predicted by the model $f_\theta^{(m)}(x)$. Typically, $p(y|\tau^{(m)})$ is modeled as a product of 2D Gaussian distributions [Alahi et al., 2016; Deo and Trivedi, 2018b]:

$$p(y|\tau^{(m)}) = \prod_{l=1}^{L} \mathcal{N}(y_l|\tau_l^{(m)}), \tag{5.4}$$

with $\tau_l^{(m)} = (\mu_{l,x}^{(m)}, \mu_{l,y}^{(m)}, \sigma_{l,x}^{(m)}, \sigma_{l,y}^{(m)}, \rho_l^{(m)})^\top$ comprising the mean values, variances, and the correlation coefficient. This heteroscedastic model allows to predict individual uncertainties for the longitudinal and lateral components and increasing uncertainties for waypoints in the more distant future. However, this comes at the expense of higher model and training complexity. In principle, instead of $\mathcal{N}(y_l|\tau_l^{(m)})$ different parametric distributions can be chosen, *e.g.* the Laplace distribution. Multimodality results from separate parameters $\tau^{(m)}$ for different maneuvers. Finally, all parameters of the likelihood distribution can be summarized in one vector:

$$\lambda = (\alpha^{(\mathrm{p})}, \alpha^{(\mathrm{c})}, \tau_1^{(\mathrm{p})\top}, \ldots, \tau_L^{(\mathrm{p})\top}, \tau_1^{(\mathrm{c})\top}, \ldots, \tau_L^{(\mathrm{c})\top})^\top. \tag{5.5}$$

Analogously to MDNs [Bishop, 1994], the NLL loss is leveraged for training and the parameters are estimated with an NN:

$$\lambda = f_\theta(x) = \left(f_\theta^{(\alpha)\top}(x), f_\theta^{(\mathrm{p})\top}(x), f_\theta^{(\mathrm{c})}(x)^\top\right)^\top. \tag{5.6}$$

The sequential form of the data is taken into account by utilizing an LSTM-based encoder-decoder model [Hochreiter and Schmidhuber, 1997]. LSTMs are capable of capturing short- and long-term dependencies in sequential data by using a gating mechanism and updating an internal cell state. Contrary to Bishop [1994] and according to Deo and Trivedi [2018b], the maneuvers m are not considered as latent variables and require labels. Thereby, the output is semantically interpretable at the expense of the necessity to annotate the data. For scalability, automatic labeling functions are leveraged and a manual review with possible corrections is performed on the validation and test set.

A challenge is the handling of multimodal distributed data. Common loss functions like the MSE loss function can be interpreted as the NLL loss of a unimodal distribution. Theoretically, modeling the loss for multimodal outputs is possible as well. However, in practice frequently the learned distributions turn out to be less diverse than the true distributions. This is known as collapsing of the modes [Makansi et al., 2019]. To address this, soft and hard constraints are applied and supposed to separate the lateral components of the trajectories associated with the passing and cut-in maneuver. For the soft constraints, an additional loss term is introduced [Karpatne et al., 2017]. For the hard constraints, a final constraint guard layer [Brosowsky et al., 2021a] is constructed and applied. The constraint guard layer prevents unimodal output distributions by design.

To summarize, the main contributions of this chapter are as follows:

- For a probabilistic and interpretable behavior prediction, LSTM-based encoder decoder models are leveraged and the output is modeled as a bimodal distribution over trajectories. The two modes of the distribution are semantically assigned to a cut-in and a passing maneuver.

- Hard and soft output constraints are modeled for improving the separation of the cut-in and passing mode. The hard constraints are imposed by applying ConstraintNet [Brosowsky et al., 2021a] and the soft constraints are implemented with an additional loss term [Karpatne et al., 2017].

- A diverse and large-scale data set is created by leveraging a retrospective view on recorded measurement files of a test vehicle fleet. For the scenario detection, the pipeline leverages the sequencing framework [Elspas et al., 2020]. Automatic labeling functions and a retrospective view are leveraged and a data set with 1856 cut-in and an equal number of passing maneuvers is created. The ground truth of the trajectories is computed fully automatically from odometry and ego vehicle's object detection.
- All LSTM-based models predict the cut-ins far before the physical baseline and the lateral trajectory error is reduced by a factor of two over a constant velocity model. It is shown that ConstraintNet achieves the best overall performance *w.r.t.* cut-in prediction, trajectory accuracy, and mode separation. Furthermore, the lateral and longitudinal uncertainties of the trajectory waypoints are well-interpretable and increase with larger time horizons as expected.
- The applied LSTM-based encoder-decoder models depend only on the ego vehicle's sensor measurements and are intended for real-world application on highways.

5.2 Related Work

In general, the prediction of vehicle behavior [Deo and Trivedi, 2018a,b; Bansal et al., 2019; Cui et al., 2019, 2020; Pan et al., 2020] is challenging for several reasons: multiple actors are interacting, the dependence on static infrastructure, the relevance of the temporal context, and the uncertainty about the intention of other drivers. For safe and predictive planning, the prediction of cutting-in vehicles is of particular importance. As visualized in Figure 5.2, existing approaches can be categorized in a) task-specific models for lane change prediction [Schmüdderich et al., 2015; Wirthmüller et al., 2021; Wissing et al., 2017; Schlechtriemen et al., 2014], b) trajectory prediction models [Bansal et al., 2019; Cui et al., 2019, 2020; Pan et al., 2020; Altche and Fortelle, 2018], and c) joint models combining a lane change and a trajectory prediction [Wirthmüller et al., 2020; Deo and Trivedi, 2018a].

a) Schmüdderich et al. [2015] model a cut-in prediction as a classification task and introduce a physics-based and a context-based model for short- and long-term prediction horizons, respectively. The physics-based approach relies on matching tracked trajectories of the target vehicle with trajectory prototypes, while the context-based approach incorporates spatial and dynamic relations to surrounding vehicles. The latter leverages so-called indicator functions that quantify whether a scene matches to a certain situation. The approach is reliable, interpretable, and works on fused camera and radar data. However, the situation complexity is reduced to typical behavior classes, the context-based approach neglects the time context, and learning the optimal parameters requires a black box optimizer. Wissing et al. [2017] propose a similar model consisting of a movement-based and situation-based model. Analogously to the context-based model, the situation-based model relies on inter-vehicle features, which are designed by experts. In comparison to the trajectory matching of the physics-based approach in Schmüdderich et al. [2015], Wissing et al. [2017] perform a classification with a Support Vector Machine and con-

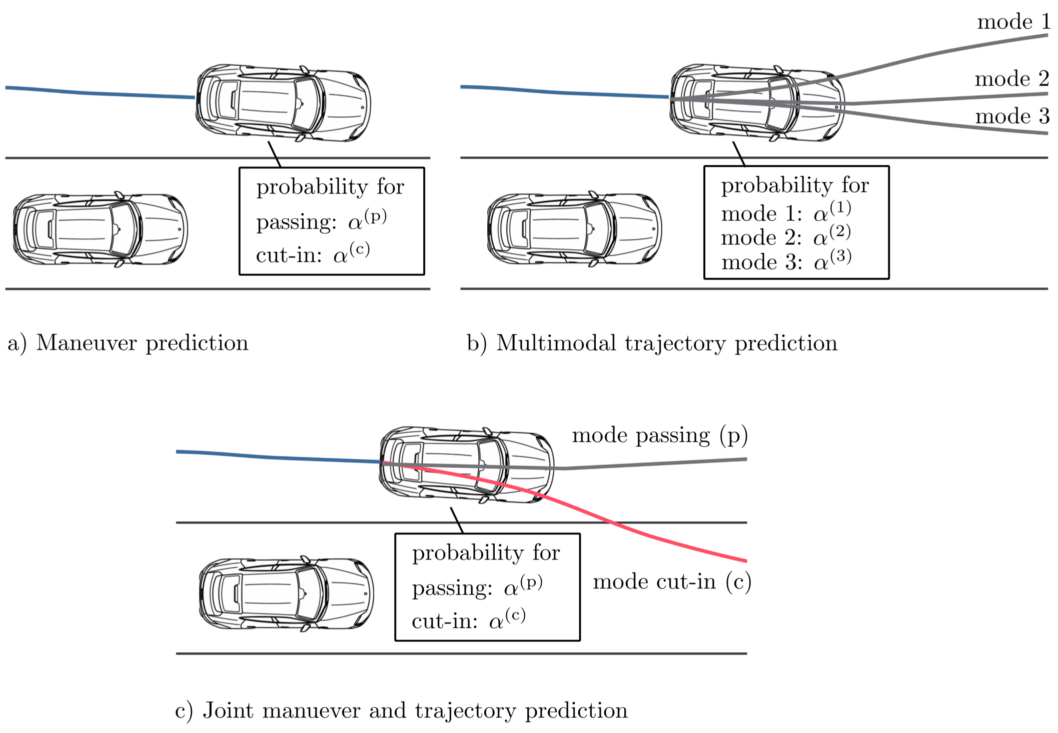


Figure 5.2: The figure shows three categories of models for behavior prediction. For illustration, a target vehicle is shown that either passes the ego vehicle and keeps its lane (passing) or changes to the ego vehicle's lane (cut-in). a) Models of the first category perform a classification over maneuvers. b) Models of the second category predict a multimodal distribution over trajectories. In general, no semantic labels are associated with the modes. c) Models of the third category perform a multimodal trajectory prediction and additionally assign maneuvers to the modes.

sider only the lateral components of the trajectory. Further publications [Weidl et al., 2018; Schlechtriemen et al., 2014] treat maneuver prediction as a classification problem. Contrary, Wirthmüller et al. [2021] model the lane change prediction as a regression task by estimating the time until the vehicle changes the lane with an LSTM. If the vehicle keeps the lane, a maximum value of 7 s should be predicted. Contrary to classification approaches, it is no longer necessary to define time intervals for cut-in labels, which are difficult to choose. Furthermore, the LSTM-based approach combines features that are related to the context and the movement of the target vehicle. In comparison to Schmüdderich et al. [2015] and Wissing et al. [2017], this allows to leverage the full temporal context and all features at once. In rule-based models, this would lead to rapidly increasing complexity. However, the pure data-driven approach, the expressive power of NNs, and the efficient recognition of temporal patterns with LSTMs make this holistic approach feasible.

b) In the last years, the majority of state-of-the-art trajectory prediction models leverage the expressive power of deep NNs. Commonly applied architectures are CNNs [Bansal et al., 2019; Cui et al., 2019], Graph Neural Networks (GNNs) [Scarselli et al., 2009; Pan et al., 2020], RNNs [Altche and Fortelle, 2018], Transformers [Vaswani et al., 2017; Giuliari et al., 2021], or models combining mechanisms of these architectures [Deo and Trivedi, 2018a; Liang et al., 2020; Alahi et al., 2016]. Depending on the architecture, different input representations are preferred. A class of approaches [Djuric et al., 2020; Cui et al., 2019, 2020; Bansal et al., 2019] propose to render the static and dynamic elements of the environment in a multichannel bird's-eye view representation. This high dimensional input is able to represent the spatial arrangement of a varying number of objects and environment features can be added in a modular way. Given the rasterized input, typically CNNs are applied. CNNs are equivariant *w.r.t.* translations and are efficient and successful to process data in grid-like structures. *E.g.* in Djuric et al. [2020], the spatial arrangement of elements in the environment is translated into pixel space of an RGB image and different types of elements are encoded with different colors. Furthermore, temporal context is included by representing objects of past time steps with reduced brightness. On the one hand, this input format deals naturally with different numbers of objects in the scene and allows to encode a number of different environment elements efficiently. On the other hand, setting the dimensions of the tensor requires a trade-off between the field of view, spatial resolution, and tensor size. Furthermore, the rendering of environment elements suffers from information loss [Liang et al., 2020] and the design choices are not obvious and require intuition, creativity, and experience, *e.g.* for rasterizing traffic signs. To conclude, the transformation in a bird's-eye view representation enables processing with CNNs at the expense of information loss and expanding the data. A more compact representation of the environment is a graph, which can be processed with GNNs [Pan et al., 2020; Gao et al., 2020; Liang et al., 2020]. Graph Convolutional Networks (GCNs) [Kipf and Welling, 2017; Henaff et al., 2015; Duvenaud et al., 2015] are GNNs that apply a graph convolution operator to extract features. Analogously to CNNs, GCNs share parameters over all locations in the graph. *E.g.* Liang et al. [2020] represent the geometry and direction of lanes in the map as a graph and extract features with a so-called LaneGCN. The past trajectories of the actors are processed with CNNs and attention mechanisms are used to fuse the map and actor information. In Gao et al. [2020], map features and the actors' past trajectories are represented as graphs and processed with a hierarchical GNN. While GNNs work on a compact representation of the environment, a unified graph-based representation of all aspects of the environment and processing with pure GNNs is challenging as well. Consequently, frequently GNNs are applied in combination with other architectures like CNNs, LSTMs and attention-mechanisms [Liang et al., 2020; Schmidt et al., 2022]. Alternatively, a grid of a fixed number of neighboring vehicles around the target vehicle can be introduced and their spatial-relations summarized as features among others in a vector [Altche and Fortelle, 2018; Wirthmüller et al., 2020]. Typically, RNNs [Altche and Fortelle, 2018] or Transformers [Giuliari et al., 2021] are applied to capture the temporal context of these feature vectors, *e.g.* in an encoder-decoder architecture. Trajectory prediction is intrinsically probabilistic and multimodal and therefore the output of the model is optimally a multimodal distribution. The prediction of a single future trajectory [Altche and Fortelle, 2018] can lead to a misleading output that represents the mean value of the underlying multimodal distribution. For an intersection scenario, this is demonstrated by Cui et al. [2019, Figure 1]. A model with two modes is able to predict reasonable trajectories and probabilities for driving straight and a turn maneuver. Contrary, unimodal models tend to predict the mean trajectory of both maneuvers, which is outside the drivable space and not reasonable. Multimodal models can be realized by predicting several trajectories in combination with an appropriate loss function, *e.g.* the Winner-Takes-All (WTA) loss [Guzman-Rivera et al., 2012; Makansi et al., 2019; Cui et al., 2019, 2020]. Further methods are MDNs, classificators over a set of fixed trajectories [Phan-Minh et al., 2020], or latent variable models [Casas et al., 2020]. The latter samples a latent variable to generate different output realizations. MDNs estimate the parameters of a multimodal distribution. However, MDNs are difficult to train and suffer from the collapsing mode problem. In Makansi et al. [2019], an NN with two stages is proposed to overcome this limitation. In the first stage, multiple hypotheses are predicted with a CNN and a modified WTA loss. In the second stage, a layer that fits a mixture density model to the hypotheses is added. In comparison, this chapter addresses mode separation by enforcing a sufficient lateral distance between the trajectory modes with soft and hard output constraints. The soft constraints are implemented with an additional loss term that penalizes constraint violations [Karpatne et al., 2017]. The hard output constraints require an additional constraint guard layer according to ConstraintNet [Brosowsky et al., 2021a].

c) A model for both, lane change and trajectory prediction, is presented in Wirthmüller et al. [2020]. A mixture of experts approach is leveraged consisting of a maneuver classification (gating model) and a prediction of distributions over future positions for each maneuver (experts). For each expert, a Gaussian mixture model is fitted to the maneuver-specific data in the joint space of in- and outputs. For prediction, Gaussian mixture regression is leveraged, *i.e.* the distributions over the future positions are determined depending on the input. In Wirthmüller et al. [2020], the ego vehicle is considered as target vehicle. This allows to capture accurate motion information and to detect all surrounding vehicles except following vehicles. However, for the behavior prediction of surrounding vehicles it may suffer from a domain change or requires an advanced and accurate perception system. Deo and Trivedi [2018a] propose to estimate the parameters of a multimodal distribution over future trajectories with an NN. Contrary to a conventional MDN, the modes are associated with six maneuvers, *e.g.* slow lane change to the left. Analogously, in this chapter cut-in and passing maneuvers are assigned to the modes of a bimodal trajectory prediction. Both trajectories are predicted simultaneously with one decoder and doubled output dimensions. Contrary in Deo and Trivedi [2018a], an one-hot encoded vector of the maneuver is added to the context vector to predict maneuver-specific trajectories. The choice of doubling the output dimension instead of using a one-hot encoded vector is motivated by observing a better separation of the modes with this modification. In Deo and Trivedi [2018a], first the dynamic behavior of the involved actors is captured with one LSTM per actor and shared parameters. Second, the interactions between them are considered with a convolutional and pooling unit. In comparison, in this chapter the spatial relations to neighboring vehicles and the lane geometry are encoded in the feature vectors time step-wise and the temporal context is leveraged with an LSTM afterwards.

5.3 Methods

5.3.1 Data Set and Automatic Scenario Detection

The data set is created from measurement files of real world test drives and consists of a collection of highway scenarios represented by $S = \{s_i\}_{i \in I}$. Each scenario includes a tracked target vehicle that performs either a cut-in or a passing maneuver among possibly

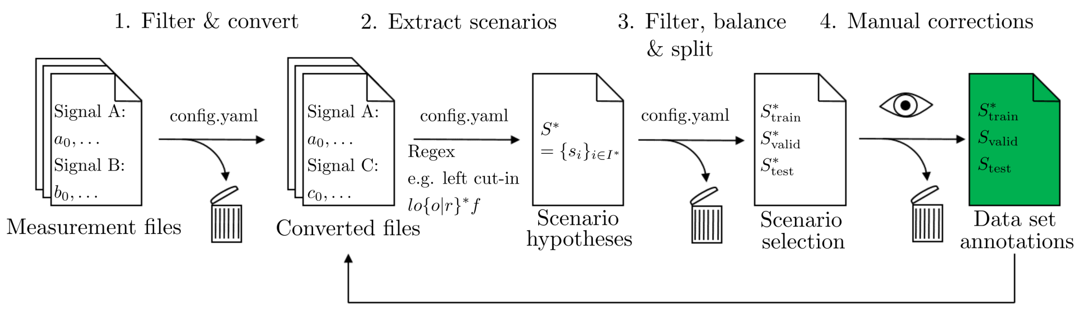

1. Filter & convert
2. Extract scenarios
3. Filter, balance & split
4. Manual corrections
Signal A: $a_0, \dots$ Signal B: $b_0, \dots$
Measurement files
config.yaml
Signal A: $a_0, \dots$ Signal C: $c_0, \dots$
Converted files
config.yaml
Regex
e.g. left cut-in
$lo\{o|r\}^*f$
$S^* = \{s_i\}_{i \in I^*}$
Scenario hypotheses
config.yaml
S^*_{train} S^*_{valid} S^*_{test}
Scenario selection
S^*_{train} S_{valid} S_{test}
Data set annotations

Figure 5.3: Schematic visualization of the pipeline for the creation of a large and diverse data set. The pipeline starts with measurement files of real-world test drives. 1) The measurement files are checked for required signals, the data is resampled to a fixed frequency, and saved in a column-oriented data format. 2) Cut-in and passing maneuvers are detected automatically with regular expressions on discretized signal states. 3) The resulting set of scenarios is balanced and split into training S^*_{train}, validation S^*_{valid}, and test sets S^*_{test}. The asterisk denotes the state before manual verification. 4) Finally, the validation and test sets are manually checked and wrong annotations are excluded. The arrow in the bottom indicates that the annotations are pointers to the converted measurement files. The figure is based on Brosowsky et al. [2021c].

other vehicles. The numbers of cut-in and passing scenarios are chosen equally. For a cut-in scenario, the cut-in time t_c is defined as the time when the target vehicle has crossed the lane marking with its center and is just assigned as the CIPV. Formally, a scenario is represented as a tuple $s = (e, f, t_s, t_e, i, a_1, \dots, a_n)^\intercal$ with $e \in \{\text{cut-in, passing}\} \times \{\text{left,right}\}$ for the semantic label, f for the reference to the measurement file, t_s and t_e for the start and end time within the measurement file, i for the identifier of the target vehicle, and $a_1, \dots, a_n$ for additional attributes, *e.g.* the cut-in time t_c. The start and end time are defined as the time points when the track of the target vehicle starts and stops, respectively. For each time step k in a scenario, the targets of the NN, namely the future trajectory y_k and the future maneuver m_k, are computed automatically from the measurement file and the scenario descriptor s. The procedures leverage a retrospective view and are explained in Section 5.4.2. For the detection of the scenarios S, a scalable and largely automated pipeline is applied and visualized in Figure 5.3. The pipeline uses the sequencing framework that is proposed in Elspas et al. [2020]. The automated steps one to three are specified with configuration files and allow continuous and iterative improvements. In the final step, manual corrections are performed with a labeling tool. The user interface of the tool is shown in Figure 5.5.

The toolchain starts with a set of measurement files, which were recorded during nine vehicle test campaigns. In general, a test campaign includes test drives of several vehicles and covers multiple days. The measurement files of all nine campaigns comprise 19 885 km driving data from ten European countries. From this distance, 11 905 km were driven on highways. The files contain sensor and processed signals in the form of multivariate time series. In a first processing step, the files are checked for the required signals and the verified files are converted from a write-optimized file format into the column-oriented Apache Parquet format. The required signals comprise egomotion signals, map information about the road type, camera signals for lane markings, and fused camera and radar data of detected and tracked surrounding vehicles. Additionally, the signals are resampled to a fixed frequency of 100 Hz. This relatively high frequency is used for the

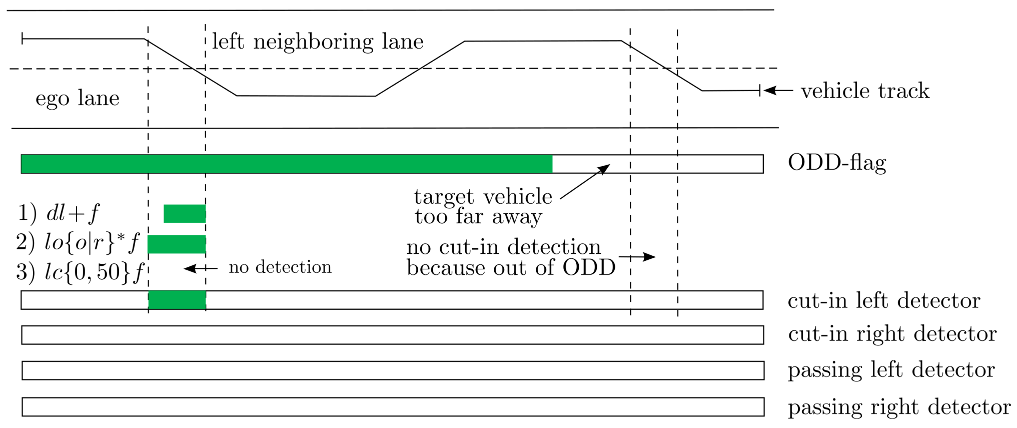


Figure 5.4: For the automatic detection of cut-in and passing scenarios, the tracks of surrounding vehicles are first extracted. For each vehicle track, a number of conditions are checked on time step level and define the ODD. The corresponding Boolean signal is called ODD-flag. In the visualized example, the target vehicle leaves the ODD at a certain point in time because the distance to the ego vehicle exceeds a specified maximum value. For the detection of cut-in and passing maneuvers, signals are discretized into states and regular expressions are defined on the alphabet given by these states. The detection within the ODD is ensured by defining states that require a true ODD-flag. For the detection of cut-ins from the left and from the right, three regular expressions are used. If at least one of the cut-in detectors has a match, the vehicle track is labeled as a cut-in scenario. The detectors of passing maneuvers require that no cut-ins are detected.

scenario detection. The input of the NN is generated less frequent and the sampling rate is reduced by a preprocessing step (see Section 5.4.2). The next steps of the data pipeline generate metadata and pointers to the files of this stage.

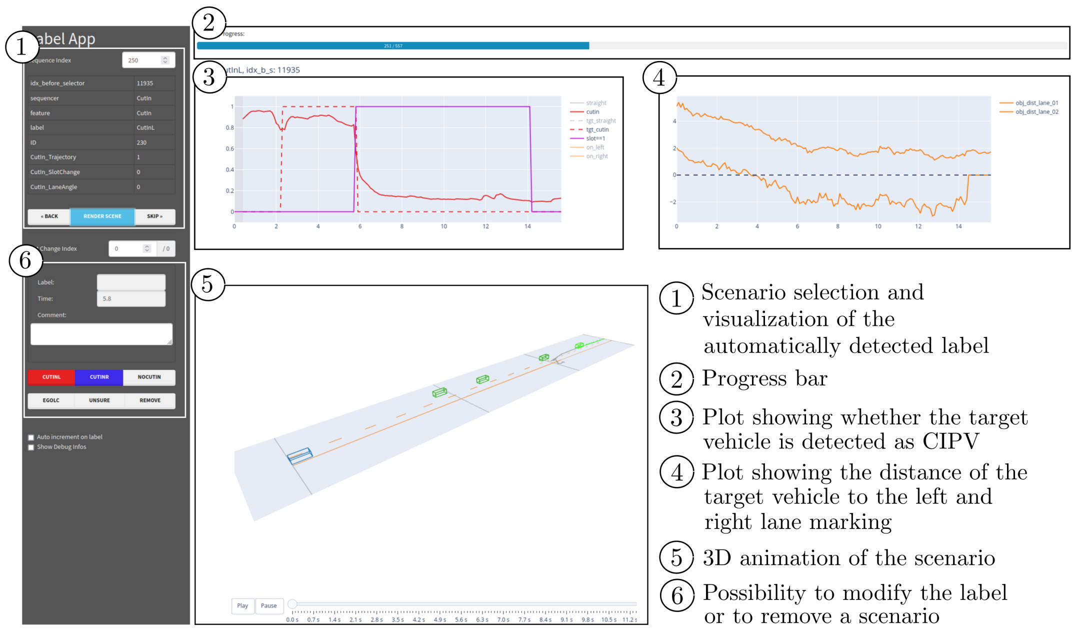
In the second step, a superset $S^* = \{s_i\}_{i \in I^*}$ of the final scenarios $S = \{s_i\}_{i \in I}$ is automatically generated and consists of scenario hypotheses. An overview of the automatic scenario detection is visualized in Figure 5.4. The detection starts with the extraction of tracks of all surrounding vehicles and pointers to their start and end points. A vehicle track consists by definition of a sequence of object detections over time that belong to the same vehicle. The fusion system attaches unique Identifiers (IDs) i to each object detection and encodes in this way the links between objects over time steps. Each vehicle track defines the following features of a potential scenario s: the reference to the measurement file f, the start t_s and end time t_e, and the identifier i of the target vehicle. The next steps, check the ODD and determine the label $e \in \{\text{cut-in, passing}\} \times \{\text{left, right}\}$. A time step of a vehicle track fulfills the ODD conditions if the distance between target and ego vehicle is less than 150 m, the velocity of the target and ego vehicle is above $10\,\mathrm{ms}^{-1}$, both lane markings are detected, and the ego vehicle is not closer than 1 m to the lane markings. The purpose of the last condition is the exclusion of ego lane changes. The evaluations of these checks are attached to each vehicle track as a Boolean signal that is true if all conditions are fulfilled in the considered time step and false otherwise. In the following, this signal is called ODD-flag. Next, each of the vehicle tracks is scanned for patterns corresponding to cut-in and passing maneuvers within the ODD. This is achieved by discretizing relevant signals into semantically meaningful states. Subsequently, regular expressions on the alphabet given by these states allow the extraction of patterns over time. The detection within the ODD is ensured by defining only states that require the ODD-flag being true. Empirically, it is found that the cut-in detection improves by defining three different regular expressions for each side. The regular expressions to detect cut-ins from the left and right lane are symmetric versions of each other. The three expressions rely on 1) the target and ego vehicle's trajectory, 2) the angles between the target vehicle and the corresponding lane marking, and 3) changes in the lane assignment of the target vehicle given by the perception system. All three detectors require that the target vehicle has finally crossed with its center the lane marking and is labeled as CIPV. This requirement is described by the final state f. For detector 1), the trajectories of the target and the ego vehicle are computed in a stationary reference frame from ego motion signals and the measured relative position of the target vehicle *w.r.t.* the ego vehicle (see Section 5.4.2 and Algorithm 3). In the stationary reference frame and for each time step, the minimum distance d_{min} of the target vehicle's position to the trajectory of the ego vehicle is computed. Additionally, for each time step the relative position of the target vehicle *w.r.t.* the ego vehicle's closest position is distinguished in to the left or to the right of the ego vehicle. For the detection of cut-ins, the minimum distance d_{min} and the relative position are binned into the following states: large distance ($1.5\,\text{m} \leq d_{\text{min}} < 5\,\text{m}$, state d), medium distance and the target vehicle is on the left or right-hand side ($1\,\text{m} \leq d_{\text{min}} < 1.5\,\text{m}$, state l or r), and small distance and target vehicle has become CIPV ($d_{\text{min}} < 1\,\text{m}$, state f). A cut-in is going to leave state d at some point in time, followed by at least one time step being in state l or r, and finally it becomes the CIPV (state f). The two patterns are formalized with the following regular expressions:

$$\begin{aligned} \text{regex\_1(cut-in, left)}&: dl{+}f, \\ \text{regex\_1(cut-in, right)}&: dr{+}f. \end{aligned} \tag{5.7}$$

For detector 2), the angle between the target vehicle and the lane marking is binned into three states: target vehicle to the left of the (state l), on the (state o), or to the right of the (state r) lane marking. Cut-ins from left are characterized as follows. After the target vehicle is to the left of the lane marking the last time, it is going to be on the lane marking, followed by an arbitrary number of time steps of being on or right to the lane marking. Finally, the target vehicle becomes the CIPV (state f). For cut-ins from right, the pattern is a symmetric version. Consequently, the regular expressions are given by:

$$\begin{aligned} \text{regex\_2(cut-in, left)}&: lo\{o|r\}^{*}f, \\ \text{regex\_2(cut-in, right)}&: ro\{o|l\}^{*}f. \end{aligned} \tag{5.8}$$

For detector 3), the lane assignment of the perception system is leveraged. The perception provides labels for the closest vehicles on the left and right neighboring lane as well as for the CIPV. Consequently, the following expressions detect cut-ins that start as closest vehicle on the left or right neighboring lane (state l or r), are optionally no longer one



1 Scenario selection and visualization of the automatically detected label
2 Progress bar
3 Plot showing whether the target vehicle is detected as CIPV
4 Plot showing the distance of the target vehicle to the left and right lane marking
5 3D animation of the scenario
6 Possibility to modify the label or to remove a scenario

Figure 5.5: The figure shows a screenshot of the developed labeling tool to verify the automatically generated scenario labels $e \in \{\text{cut-in, passing}\} \times \{\text{left, right}\}$. The tool provides functionality to iterate over the detected scenarios, to show relevant plots, to render a 3D animation of the scenario, and to remove scenarios or to modify the label if necessary.

of the closest vehicles but remain in the ODD (state c), and finally become the CIPV (state f):

$$\begin{aligned} \text{regex\_3(cut-in, left):}\ & lc\{0,50\}f, \\ \text{regex\_3(cut-in, right):}\ & rc\{0,50\}f. \end{aligned} \tag{5.9}$$

If at least one of the cut-in detectors matches, the scenario s is labeled as cut-in and the matching regular expression determines whether the cut-in is from the left or from the right neighboring lane $e = (\text{cut-in, left/right})$. Note that the cut-in time t_c can always be determined, because all cut-in detectors require that the target vehicle becomes finally the CIPV (state f). For passing vehicles, vehicle tracks without any cut-in detection are extracted. Additionally, in each time step the passing vehicle must be not the CIPV and the center needs to have a maximum distance of 3 m to the closest of the two ego lane markings. If these conditions are fulfilled and the passing vehicle is to the left or to the right of the ego lane, time steps are labeled with the state l or r, respectively. For the detection that the vehicle passes on the left or on the hand side, the regular expressions can be written as:

$$\begin{aligned} \text{regex(passing, left):}\ & (?<=s)l^*(?=e), \\ \text{regex(passing, right):}\ & (?<=s)r^*(?=e), \end{aligned} \tag{5.10}$$

with state s marking the start and state e marking the end of a vehicle track. The expression $(?<=\dots)$ is a lookbehind and $(?=\dots)$ a lookahead pattern, which are checked but not

Description	Total	Train	Valid	Test
Number of scenarios.	3712	2598	557	557
Number of scenarios with cut-ins from left / right lane.	691/ 1165	491/ 800	108/ 174	92/ 191
Number of scenarios with passing vehicles on left / right lane.	672/ 1184	455/ 852	106/ 169	111/ 163
Mean tracking time of cutting-in vehicle from left / right lane before it becomes CIPV.	5.8/ 8.7s	5.7/ 8.6s	6.5/ 8.9s	5.7/ 8.6s
Mean tracking time of passing vehicle on left / right lane.	14.1/ 12.0s	13.8/ 12.0s	14.5/ 11.6s	14.7/ 12.2s
Mean relative velocity of cutting-in vehicle from left / right lane at t_c.	2.85/ $-3.89\mathrm{m\,s^{-1}}$	2.88/ $-3.97\mathrm{m\,s^{-1}}$	2.45/ $-3.83\mathrm{m\,s^{-1}}$	3.22/ $-3.62\mathrm{m\,s^{-1}}$
Mean longitudinal distance of cutting-in vehicle from left / right lane at t_c.	42.6/ 54.4 m	42.6/ 54.4 m	43.5/ 53.5 m	41.6/ 55.0 m

Table 5.1: The table summarizes a range of statistics of the created data set. The numbers are denoted for the total data set (total), the training (train), the validation (valid), and the test set (test).

part of the match. The matches of the two detectors are mutually exclusive. The scenario is labeled as $e = (\text{passing, left})$ or $e = (\text{passing, right})$ depending on the detector that found the match. Vehicle tracks without any cut-in and passing detection are out of the ODD and excluded from the scenario set S^*.

In the third step, a subset of the automatically detected scenario hypotheses is selected, the number of scenarios between cut-in and passing maneuvers is balanced, and the scenarios are split into a training, a validation, and a test set. Scenarios are kept if the target vehicle is tracked for at least 6 s and if the road type is highway. Furthermore, the number of cut-in and passing scenarios is set equally and the data set is partitioned into a training, validation, and test set according to a split of 70%, 15%, and 15%, respectively.

For the evaluation metrics, it is important that all labels $e \in \{\text{cut-in, passing}\} \times \{\text{left, right}\}$ of the automatically generated validation S^*_{valid} and test set S^*_{test} are verified. For the verification, a labeling tool has been developed with the Dash[1] framework for Python (see Figure 5.5). The tool provides functionality to iterate over the scenario descriptors, to create plots of the corresponding multivariate time series, to show a 3D animation of the scenario, and to remove the scenario or to modify the scenario label $e \in \{\text{cut-in, passing}\} \times \{\text{left, right}\}$ if necessary. The verified validation and test sets are denoted as S_{valid} and S_{test} without asterisks.

The presented automated pipeline enables the creation of a large data set consisting of 1856 cut-in and 1856 passing scenarios. Approximately, a third of the maneuvers occurs on the left lane and two third on the right lane. Cutting-in vehicles from the left (right) lane have on average a distance of 42.6 m (54.4 m) and a relative velocity of $2.85\mathrm{m\,s^{-1}}$ ($-3.89\mathrm{m\,s^{-1}}$). Further statistics of the data set are summarized in Table 5.1. In the table, the numbers are given for the training, the validation, the test, and the total data set. Due to the large

[1] `https://dash.plotly.com`, accessed on 08/03/2022

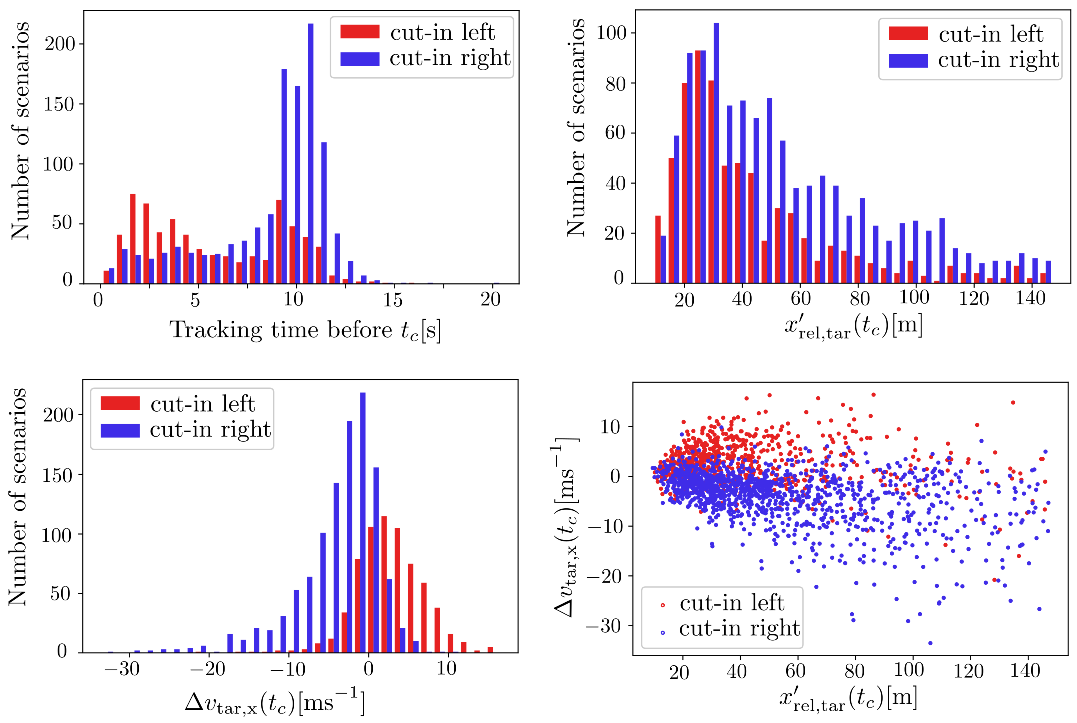


Figure 5.6: For the 1856 cut-in scenarios in the data set, three histograms and one scatter plot show distributions over the following quantities. *Top left*: The tracking time of the target vehicle before the cut-in time t_c. *Top right*: The distance $x'_{\mathrm{rel,tar}}(t_c)$ of the target vehicle at the cut-in time t_c. *Bottom left*: The relative longitudinal velocity $\Delta v_{\mathrm{tar,x}}(t_c)$ at the cut-in time t_c. *Bottom right*: The relative longitudinal velocity $\Delta v_{\mathrm{tar,x}}(t_c)$ over the distance $x'_{\mathrm{rel,tar}}(t_c)$ at the cut-in time t_c. Cut-ins from the left lane are visualized in red and cut-ins from the right lane in blue.

number of scenarios in each of the sets, the average values and ratios are similar between them. For the cut-in scenarios, Figure 5.6 shows three histograms and one scatter plot of relevant quantities. In addition to the average values in Table 5.1, these plots provide insights about the distributions of the quantities, the completeness and representativeness of the data set. The plots distinguish between cut-ins from the left and the right lane and the differences can be interpreted with the following prototypical behavior. Usually, cut-ins from the left lane overtake the ego vehicle and change to the ego lane as soon as possible. Contrary, cut-ins from the right lane typically change to the ego lane to overtake a slower vehicle on their lane, *e.g.* a truck. The histogram in the top left of Figure 5.6 presents the distribution of the tracking time of the target vehicle before becoming the CIPV. As expected, the cut-ins from the left lane are tracked on average for a shorter time because they typically enter the front-facing field of view from behind. Contrary, cut-ins from the right enter the field of view usually from a large distance, which results in a longer tracking time. The histogram in the top right of Figure 5.6 shows the distribution of the distance $x'_{\mathrm{rel,tar}}(t_c)$ of the target vehicle when cutting-in. Typically, cut-ins from the left lane change to the ego lane directly in front of the ego vehicle after completing the overtake maneuver. Thus, the histogram shows a strong peak for distances between 20 m and 30 m. In comparison, cut-ins from the right occur more frequently also at larger distances. This characteristic is reasonable since overtaking a slower vehicle on the target vehicle's lane may happen also at large distances to the ego vehicle. Moreover, short

Feature f	Description
$x_{\text{tar}}, y_{\text{tar}}$	Past trajectory points of the target vehicle.
$v_{\text{tar},x}, v_{\text{tar},y}$	Velocity of the target vehicle.
w_{tar}	Width of the target vehicle.
$v'_{\text{ego},x}, a'_{\text{ego},x}$	Longitudinal velocity and acceleration of the ego vehicle.
$x'_{\text{rel, tar}}, y'_{\text{rel, tar}}$	Relative position of the target vehicle *w.r.t.* the ego vehicle.
$x'_{\text{rel, CIPV}}$	Longitudinal position of the CIPV *w.r.t.* the ego vehicle.
$y_{\text{env, l/r, s}}$	Lateral position of the left/right lane marking in a distance of s *w.r.t.* the target vehicle for encoding the lane curvature, $s = 0/20/40/60/80\,\text{m}$.
$l_{\text{env,l/r}}$	Type of the lane marking (solid, broken).

Table 5.2: The table shows the available features for the behavior prediction. Features that are measured in the reference frame of the ego vehicle are denoted with an apostrophe. In Figure 5.7, the features are visualized.

distances occur rarely because the ego vehicle is typically faster and a certain minimum distance is required for safety reasons. The histogram in the bottom left of Figure 5.6 shows the distribution of the relative longitudinal velocity $\Delta v_{\text{tar,x}}(t_c)$ between the target and the ego vehicle. As expected, cut-ins from the left are faster on average and cut-ins from the right typically slower than the ego vehicle. The scatter plot in the bottom right of Figure 5.6 correlates the relative longitudinal velocity $\Delta v_{\text{tar,x}}(t_c)$ with the distance $x'_{\text{rel,tar}}(t_c)$ of the target vehicle when cutting-in. Critical cut-ins have a large negative relative velocity and occur at small distances. In general, the plots confirm that the data set covers various cut-in scenarios with a wide range of relative velocities and distances. In conclusion, the representativeness and the completeness of the data set is considered to be strong based on this statistical analysis.

5.3.2 Model Input

The behavior prediction is designed in an agent-centric way, *i.e.* for each vehicle of interest, a separate prediction is performed. This agent-wise prediction enables the handling of a variable number of agents in a natural way. For each selected vehicle, an input sequence $x_k = (x_{k,-H}, \ldots, x_{k,0})$ is computed from historical and current perception and sensor data. The time step of prediction is denoted as k and $x_{k,-h}$ represents a feature vector at time step $k-h$. The input sequence consists of at least $H=5$ and maximally $H=60$ time steps depending on the duration the target vehicle is already tracked. For the chosen step size of $dt = 0.1$ s, this corresponds to a duration T_H between 0.5 s and 6 s. The input sequence enables the NN to leverage temporal patterns of the maneuvers.

For an input sequence x_k with prediction time step k, a set of features is evaluated for each time step $k-h$ of the considered history and standardized. An overview of all available features $\{f_j\}_{j\in J}$ is shown in Table 5.2 and visualized in Figure 5.7. Figure 5.7 visualizes additionally the used reference frames, which are either fixed to the ego or to the target vehicle. Features in the reference frame of the ego vehicle are marked with an apostrophe and features fixed to the target vehicle without an apostrophe. Features in the form of

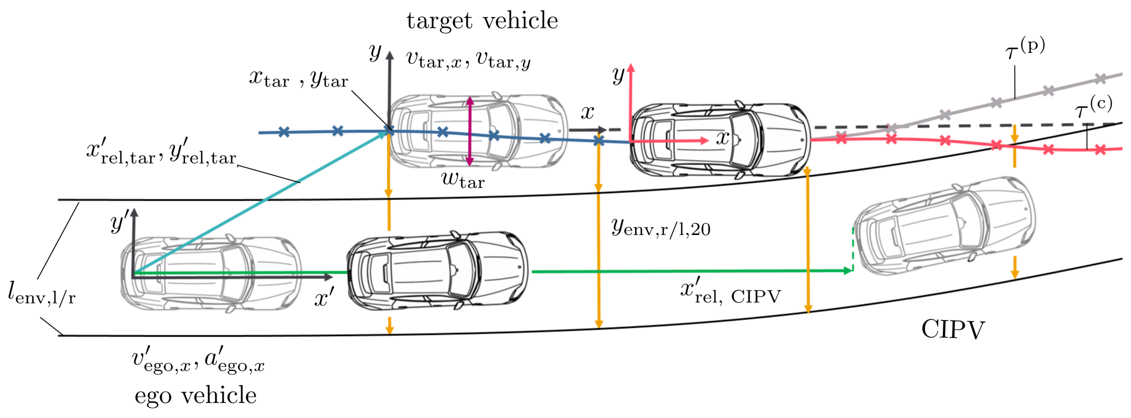

target vehicle
$v_{\mathrm{tar},x}, v_{\mathrm{tar},y}$
x_{tar} , y_{tar}
$x'_{\mathrm{rel,tar}}, y'_{\mathrm{rel,tar}}$
y
x
w_{tar}
y'
x'
$l_{\mathrm{env,l/r}}$
$y_{\mathrm{env,r/l,20}}$
$x'_{\mathrm{rel,\ CIPV}}$
CIPV
$\tau^{(\mathrm{p})}$
$\tau^{(\mathrm{c})}$
$v'_{\mathrm{ego},x}, a'_{\mathrm{ego},x}$
ego vehicle

Figure 5.7: Visualization of the predicted bimodal trajectory (red and grey) at time step k (bold vehicles) and of the input features at time step $k-h$ (shaded vehicles). The past trajectory (blue, x_{tar}, y_{tar}) and the predicted trajectories ($\tau^{(\mathrm{p})}$, $\tau^{(\mathrm{c})}$) are measured in a reference frame with origin at the location of the target vehicle at prediction time k. The other features with position or distance information are either in the reference frame with origin at the target vehicle's (without apostrophe) or ego vehicle's (with apostrophe) location at time step $k-h$. Table 5.2 describes all features shortly. The input sequence for the NN comprises at least features of the latest five times steps and maximally 60 time steps corresponding to a duration between 0.5 s and 6 s. The figure is based on Brosowsky et al. [2021c].

velocities and accelerations are measured in a global reference frame and rotated according to the orientation of the ego vehicle's reference frame.

The features are categorized via the subscript into ego vehicle f_{ego}, target vehicle f_{tar}, environment f_{env}, and interaction or relational features f_{rel}. The environment features describe static elements, while the interaction features are used for spatial relations between surrounding vehicles. The total feature set $\{f_j\}_{j\in J}$ includes features related to the type and geometry of the lane markings, the past trajectory of the target vehicle, the target vehicle's velocity and width, the longitudinal velocity and acceleration of the ego vehicle, and the relative position of the target vehicle and the CIPV *w.r.t.* the ego vehicle. The latter features indicate whether gaps in the traffic are available for a cut-in maneuver. If no CIPV is detected, a default value is used as input. Features regarding the interaction of the target vehicle and a potential vehicle in front of the target vehicle are of interest as well. *E.g.* the approaching of a slower preceding vehicle is a typical reason for a lane change. However, these features are not included due to limitations of the perception system. The width of the target vehicle is included to inform the NN about the vehicle type. This allows to take different dynamics into account, *e.g.* for cars and trucks. Instead of curvilinear coordinates, positions are given in Cartesian coordinates and features about the lane marking geometry $y_{\mathrm{env,l/r,s}}$ are included. Thereby, a direct degradation of the prediction accuracy due to noisy lane marking detections is avoided. Instead, the NN is enabled to reason about the temporal context of the noisy signals by itself. Furthermore, the cut-in behavior depends potentially on the road topology and the corresponding features $y_{\mathrm{env,l/r,s}}$ are important for situation-aware predictions.

Except for the past trajectory of the target vehicle, the input features are either directly given by the perception and ego motion sensors or can be computed with basic operations from several of these signals. Contrary, the past trajectory of the target vehicle requires higher computational effort and is performed in a twofold procedure as shown in Figure 5.14. First, odometry is used to determine the ego vehicle's trajectory. Second, the target vehicle's trajectory is determined from the ego vehicle's trajectory and the target vehicle's relative positions *w.r.t.* the ego vehicle. The details are explained in Section 5.4.2 and Algorithm 3.

5.3.3 Encoder-Decoder Architecture

For the bimodal trajectory prediction model, two LSTMs are leveraged in an encoder-decoder architecture [Cho et al., 2014; Sutskever et al., 2014]. LSTMs are enhanced versions of RNNs and explicitly designed to exploit temporal dependencies over long and short terms. Thereby, LSTMs are a good choice for the encoder and decoder. In the following, first LSTMs and then the encoder-decoder architecture are explained.

Long Short-Term Memory

RNNs are explicitly designed to process sequential data of variable length and propagate information over time steps with recurrent connections (see Section 2.2.1). However, training RNNs is difficult and the learning of long-term dependencies is challenging. An LSTM [Hochreiter and Schmidhuber, 1997] is an enhanced version of an RNN that leverages a forget, an input, and an output gate to control the flow of information. Thereby, LSTMs improve the capability of learning long-term dependencies significantly over vanilla RNNs. The repeating module of an LSTM is called LSTM cell and visualized in Figure 5.8.

Additionally to the hidden state h_k of vanilla RNNs, LSTMs leverage a second recurrent connection to propagate the so-called cell state c_k. For each time step k, LSTMs update their cell c_k and hidden state h_k by using forget f_k, input i_k, and output gates o_k. The explicit equations are given by the following formulas:

$$\begin{aligned}
f_k &= \sigma\big(W^{(xf)}x_k + b^{(xf)} + W^{(hf)}h_{k-1} + b^{(hf)}\big), \\
i_k &= \sigma\big(W^{(xi)}x_k + b^{(xi)} + W^{(hi)}h_{k-1} + b^{(hi)}\big), \\
g_k &= \tanh\big(W^{(xg)}x_k + b^{(xg)} + W^{(hg)}h_{k-1} + b^{(hg)}\big), \\
o_k &= \sigma\big(W^{(xo)}x_k + b^{(xo)} + W^{(ho)}h_{k-1} + b^{(ho)}\big), \\
c_k &= f_k \odot c_{k-1} + i_k \odot g_k, \\
h_k &= o_k \odot \tanh(c_k).
\end{aligned} \tag{5.11}$$

For the cell state c_k, the forget gate f_k determines the weighting of the previous cell state c_{k-1} and the input gate i_k controls the weighting of the cell input g_k of the current time step. The hidden state h_k is computed from the updated cell state c_k and the output gate o_k. The three gates and the cell input g_k are determined by linearly combining the input x_k and the previous hidden state h_{k-1} and subsequently applying an activation function. In the equations (5.11), $\odot$ denotes the Hadamard product, tanh the hyperbolic tangent

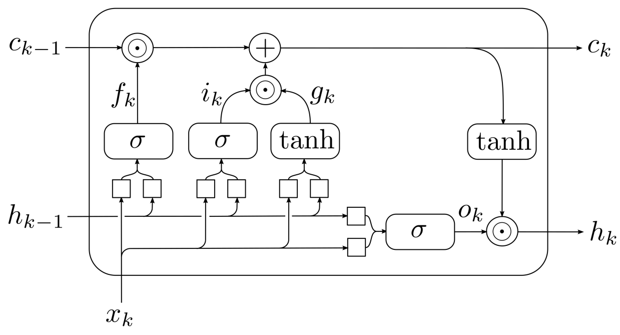


Figure 5.8: An LSTM cell processes sequential data by updating and passing a hidden state h_k and an internal cell state c_k over time steps k. An input i_k, a forget f_k, and an output gate o_k are leveraged to control the flow of information. For each gate, the input x_k of the current time step and the hidden state of the previous time step h_{k-1} are passed through linear layers (□) and the sigmoid function (σ) is applied element-wise as activation function. For updating the cell c_k and the hidden state h_k, the gates balance the influence of the previous cell state c_{k-1}, the previous hidden state h_{k-1}, and the cell input g_k of the current time step. In the visualization, $\odot$ denotes the Hadamard product, + an element-wise addition, and tanh an element-wise applied hyperbolic tangent.

(element-wise applied), σ the sigmoid function (element-wise applied), W the weight matrices, and b the bias vectors. Note that the three gates f_k, i_k, o_k, the cell input g_k, the cell state c_k, and the hidden state h_k have the same dimensionality.

Theoretically, even vanilla RNNs are capable of storing information over many time steps. However, in practice the training suffers from vanishing and exploding gradients [Hochreiter and Schmidhuber, 1997; Bengio et al., 1994; Pascanu et al., 2013]. For an illustration of this problem, a standard loss L in the form of a sum of losses per time step L_k is considered:

$$L(\theta) = \sum_{k=1}^{K} L_k(h_k), \tag{5.12}$$

$$h_k = f(x_k, h_{k-1}, \theta). \tag{5.13}$$

Here, K is the length of the sequence and f is an RNN with parameters θ. In practice, the gradient of the loss can be computed first by unfolding the RNN into a feedforward NN and second by performing the differentiation with the backpropagation algorithm (see Section 2.2.2). The unfolding of RNNs is visualized in Figure 2.6. Analytically, the gradient can be written as a sum of products [Pascanu et al., 2013]:

$$\frac{\partial L}{\partial \theta} = \sum_{k=1}^{K} \sum_{l=0}^{k-1} \underbrace{\frac{\partial L_k}{\partial h_k} \left(\prod_{k \geq m > k-l} \frac{\partial h_m}{\partial h_{m-1}} \right) \frac{\partial h_{k-l}}{\partial \theta}}_{P_{(k-l)\to k}}. \tag{5.14}$$

Each product $P_{(k-l)\to k}$ can be interpreted as the effect of the parameter θ from a previous time step $k-l$ on the loss L_k at time step k. For the vanilla RNN in Equations (2.10), the long-term contributions ($l \gg 1$) involve many multiplications of the same weight matrix.

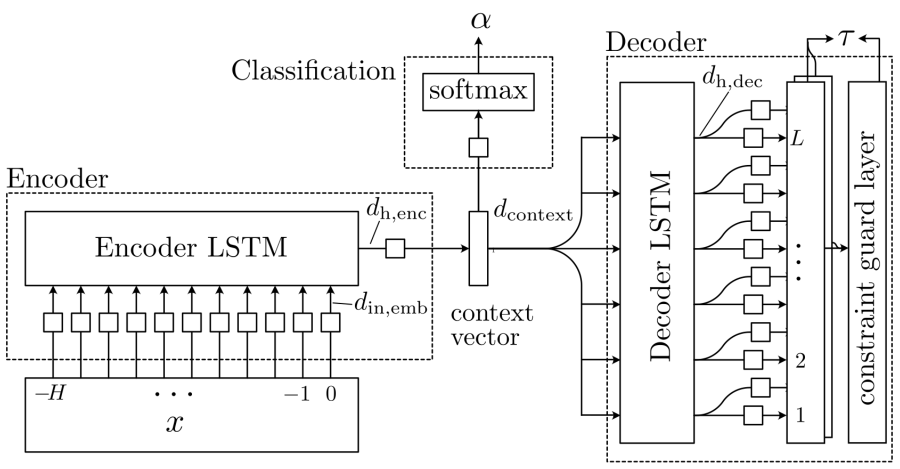


Figure 5.9: Illustration of the applied encoder-decoder architecture that consists of two LSTMs, one LSTM for the encoder and one for the decoder. The encoder processes the input x and outputs the context vector. The decoder generates the trajectory distributions $\tau^{(\mathrm{p/c})}$ for the passing and cut-in mode. For the case that hard constraints are imposed, additionally a constraint guard layer is applied. The dimension of the input embedding is denoted with $d_{\mathrm{in,emb}}$, of the hidden state of the encoder LSTM with $d_{\mathrm{h,enc}}$, of the context vector with d_{context}, and of the hidden state of the decoder LSTM with $d_{\mathrm{h,dec}}$. The mode probabilities α are generated with a softmax function. The figure is based on Brosowsky et al. [2021c].

Similar to a product of scalars with repeating multiplications of the same number, the final product represents either a very large or a very small contribution to the gradient. This behavior explains the difficulty of learning long-term dependencies with RNNs. LSTMs address this downside by controlling the flow of information with gates. In each time step, the gates of LSTMs enable to remove, to add, and to read information systematically from the cell state. This can be interpreted as information is kept by default and modifications require actively forgetting or adding information. Thereby, the learning of storing information over long terms is supported.

Encoder-Decoder Architecture

The joint vehicle trajectory and cut-in prediction is modeled with an encoder-decoder architecture [Cho et al., 2014; Sutskever et al., 2014] based on LSTMs, which is visualized in Figure 5.9. The model consists of two parts: 1) The encoder compresses the input sequence into a context vector. 2) The decoder generates the output sequence from the compressed context vector. Note, that the input and the output sequences have different lengths in general. The length of the input sequence $x = (x_{-H}, \dots, x_0)$ depends on the time the target vehicle is already tracked and covers between $T_H = 0.5$ s and $T_H = 6$ s. The output sequence $\tau^{(\mathrm{p/c})} = (\tau_1^{(\mathrm{p/c})}, \dots, \tau_L^{(\mathrm{p/c})})$ comprises the Gaussian distributions over the waypoints of the passing and the cut-in maneuver and covers $T_L = 3$ s. The topology of such an RNN with in- and output sequences of different lengths is called many-to-many (compare the classification in Figure 2.7).

Here, the encoder and the decoder are modeled with separate LSTMs. Furthermore, only single layer LSTMs are considered because no improvements were found with stacked

LSTMs. The first LSTM (encoder) encodes the input sequence of variable length in a context vector with fixed length d_{context}. Before the LSTM is applied, each element of the input sequence is passed through a fully connected layer with shared weights. This procedure generates an input embedding of dimension $d_{\text{in,emb}}$. After the LSTM is applied, the context vector is generated from the last hidden state (dimension $d_{\text{h,enc}}$) with an additional fully connected layer. For both modes $m = \text{p/c}$ and for each future time step l, the second LSTM (decoder) generates the mean values, the standard deviations, and the correlation term of the trajectory waypoints $\tau_l^{(m)} = (\mu_{l,x}^{(m)}, \mu_{l,y}^{(m)}, \sigma_{l,x}^{(m)}, \sigma_{l,y}^{(m)}, \rho_l^{(m)})^\intercal$ from the context vector. The decoder LSTM uses the context vector as input at each future time step l and different outputs arise from updating the cell and hidden state. The value ranges of the variances $\sigma_x, \sigma_y \in (0, \infty)$ and of the correlation term $\rho \in (-1, 1)$ are enforced with the activation functions $1/\exp(x)$ and $\tanh(x)$, respectively. The outputs $\tau_l^{(m)}$ of the two maneuvers $m = \text{p/c}$ are generated by applying the same linear projection on different halves of the decoder's hidden state (dimension $d_{\text{h,dec}}$). For enforcing a lateral separation of the two trajectory modes, an additional constraint guard layer is added. This is a concept of ConstraintNet [Brosowsky et al., 2021a]. The design of the constraint guard layer is explained in detail in the next Section 5.3.4. Finally, the probabilities $\alpha = (\alpha^{(\text{p})}, \alpha^{(\text{c})})^\intercal$ for the passing $\tau^{(\text{p})}$ and the cut-in $\tau^{(\text{c})}$ trajectory must be predicted. These probabilities are generated with a linear layer and a softmax function on top of the context vector.

5.3.4 Soft and Hard Output Constraints

A challenge of multimodal trajectory prediction is the prediction of well-separated trajectory modes [Makansi et al., 2019]. In the following, soft and hard constraints on the NN's output are considered to address this difficulty. The constraints are defined on the mean values of the waypoints of the passing $\mu^{(\text{p})} = (\mu_1^{(\text{p})}, \dots, \mu_L^{(\text{p})})$ and the cut-in trajectory $\mu^{(\text{c})} = (\mu_1^{(\text{c})}, \dots, \mu_L^{(\text{c})})$ with the goal to encourage a clear lateral separation between these two trajectories.

For the soft constraints, a loss term is defined that penalizes converging and diverging modes. The lateral mean displacement error ℓ_{LMDE} between the two trajectories is used as distance measure and an optimal value v_{LMDE} for this distance is set. The loss of the soft constraint ℓ_{C} is the deviation of the actual distance and the set optimal value:

$$\ell_{\text{C}}(\mu^{(\text{p})}, \mu^{(\text{c})}) = |\ell_{\text{LMDE}}(\mu^{(\text{p})}, \mu^{(\text{c})}) - v_{\text{LMDE}}|, \tag{5.15}$$

$$\ell_{\text{LMDE}}(\mu^{(\text{p})}, \mu^{(\text{c})}) = \frac{1}{L} \sum_l |\mu_{l,y}^{(\text{p})} - \mu_{l,y}^{(\text{c})}|. \tag{5.16}$$

The loss of the soft constraint ℓ_{C} is added to the training loss (see Section 5.3.5) with the weight w_{C}. Empirically, the optimal mean lateral distance was determined to $v_{\text{LMDE}} = 0.75\,\text{m}$ and the weight for the loss term to $w_{\text{C}} = 1\,\text{m}^{-1}$.

For the hard output constraints, the lateral components of the cut-in trajectory are forced to be separated from the passing trajectory with increasing distances for waypoints in the more distant future. Formally, the minimum lateral distances are set via a quadratic

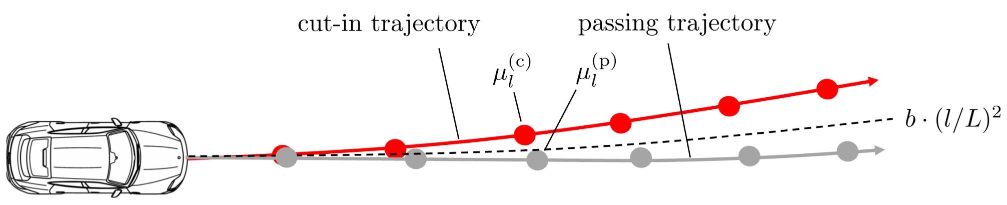


Figure 5.10: The hard constraints enforce a quadratically increasing minimum lateral separation of the cut-in trajectory (red) from the passing trajectory (grey). For the lth waypoint, the minimum lateral separation is given by $|\mu_{l,y}^{(c)} - \mu_{l,y}^{(p)}| \geq b \cdot (l/L)^2$.

function $|\mu_{l,y}^{(c)} - \mu_{l,y}^{(p)}| \geq b \cdot (l/L)^2$ as visualized in Figure 5.10. Here, b is the minimum lateral distance between the waypoints at the end of the prediction horizon. Based on empirical observations, the value was set to $b = 0.7$ m. The output constraint is embedded in the NN architecture with the concept of ConstraintNet [Brosowsky et al., 2021c]. According to the constraint guard layer in Brosowsky et al. [2021c], the output constraint is parametrized:

$$\mu_{l,y}^{(c)} = \mu_{l,y}^{(p)} \pm [b \cdot (l/L)^2 + \exp(z_l)]. \tag{5.17}$$

Here, z_l is an intermediate unconstrained variable for the lateral component of the cut-in trajectory. The exponential function generates a positive value and is added to the minimum lateral distance. Depending on whether the target vehicle is on the left ($-$) or right ($+$) neighboring lane, the sign in front of the bracket is determined *i.e.* the cut-in trajectory is on the right or left-hand side of the passing trajectory. The target vehicle is assigned to the left lane when the distance to the left lane marking is smaller than to the right lane marking and *vice versa*. As visualized in Figure 5.9, the constraint guard layer can be added on top of the architecture without constraints. Starting with the architecture without constraints, the lateral components of the waypoints $\mu_{l,y}^{(c)}$ of the cut-in trajectory can be used as intermediate unconstrained variable z_l for the architecture with hard output constraints. Finally, the constraint guard layer is applied and ensures that the output constraints hold.

5.3.5 Loss

In many classification and regression tasks, the loss is interpreted as an average error that measures the discrepancy between the model's prediction and the label. According to this intuition, the joint trajectory and cut-in prediction can be trained with a cross-entropy loss for the maneuver classification and an MSE loss for the trajectory part. However, the output also includes uncertainties of the trajectory waypoints as visualized in Figure 5.1. For these uncertainties, no labels exist. A solution provides the MLE approach, which was introduced in Section 2.2.3. The approach is equivalent to minimizing the NLL loss. Thus, in the following the NLL loss is derived for the joint trajectory and cut-in prediction. In practice, a mixture of error-based losses and the NLL loss performed optimal.

As introduced in Chapter 5.1, the NN output λ can be interpreted as parameters of an output distribution. The output distribution is bimodal ($m = \text{p}/\text{c}$) and independent 2D

Gaussian distributions are assumed for the waypoints. According to the Equations (5.1), (5.2), and (5.4), the likelihood of a training sample (y, m, x) can be written as:

$$p(y, m|\lambda) = p(m|\alpha^{(\mathrm{p})}, \alpha^{(\mathrm{c})}) \prod_{l=1}^{L} \mathcal{N}(y_l|\tau_l^{(m)}), \tag{5.18}$$

with $\lambda = f_\theta(x)$ comprising the maneuver probabilities $\alpha^{(\mathrm{p/c})}$ and the parameters of the waypoint distributions $\tau_l^{(\mathrm{p/c})}$. The NLL loss for this distribution splits into a cross-entropy part for the maneuvers and a separate term for the trajectories:

$$\ell_{\mathrm{NLL}}(y, m, \lambda) = \ell_{\mathrm{CE}}(m, \alpha^{(m)}) + w \cdot \ell_{\mathrm{TNLL}}(y, \tau^{(m)}), \tag{5.19}$$

with w a scaling factor. While the cross-entropy loss function ℓ_{CE} is well known, the loss function of the trajectory part ℓ_{TNLL} requires further explanation. An analytical expression for ℓ_{TNLL} can be derived as follows:

$$\ell_{\mathrm{TNLL}}(y, \tau^{(m)}) \propto -\log \prod_{l=1}^{L} \mathcal{N}(y_l|\tau_l^{(m)}) \propto -\frac{1}{L} \sum_l \log \mathcal{N}(y_l|\tau_l^{(m)}). \tag{5.20}$$

Here, $\mathcal{N}(\cdot|\cdot)$ are 2D Gaussian distributions, which are parametrized by mean values, variances, and a correlation coefficient:

$$\begin{aligned}\mathcal{N}(y_x, y_y|\mu_x, \mu_y, \sigma_x, \sigma_y, \rho) =& \frac{1}{2\pi\sigma_x\sigma_y\sqrt{1-\rho^2}} \exp\Bigg(-\frac{1}{2(1-\rho^2)}\Bigg[\frac{(y_x-\mu_x)^2}{\sigma_x^2} + \\ & \frac{(y_y-\mu_y)^2}{\sigma_y^2} - \frac{2\rho(y_x-\mu_x)(y_y-\mu_y)}{\sigma_x\sigma_y}\Bigg]\Bigg).\end{aligned} \tag{5.21}$$

The final expression for the NLL loss of the trajectory part is obtained by plugging Equation (5.21) in Equation (5.20):

$$\begin{aligned}\ell_{\mathrm{TNLL}}(y, \tau^{(m)}) =& \frac{1}{L} \sum_l \Bigg[\log\Big(2\pi\sigma_x\sigma_y\sqrt{1-\rho^2}\Big) + \frac{1}{2(1-\rho^2)}\Bigg(\frac{(y_x-\mu_x)^2}{\sigma_x^2} + \frac{(y_y-\mu_y)^2}{\sigma_y^2} \\ & - \frac{2\rho(y_x-\mu_x)(y_y-\mu_y)}{\sigma_x\sigma_y}\Bigg)\Bigg]_l^{(m)}.\end{aligned} \tag{5.22}$$

In this equation, the indices l and m for $\tau_l^{(m)} = (\mu_{l,x}^{(m)}, \mu_{l,y}^{(m)}, \sigma_{l,x}^{(m)}, \sigma_{l,y}^{(m)}, \rho_l^{(m)})^\intercal$ are shifted to the outer closing bracket. An isotropic homoscedastic model is obtained by assuming a constant and isotropic standard deviation ($\sigma_{l,x/y}^{(m)} = \mathrm{const}$ and $\rho_l^{(m)} = 0$). In the homoscedastic case, ℓ_{TNLL} turns out to be the mean squared error loss except for a weighting factor and an additive constant:

$$\ell_{\mathrm{MSE}}\big(y, \mu^{(m)}\big) = \frac{1}{2L} \sum_l \big(y_{l,x} - \mu_{l,x}^{(m)}\big)^2 + \big(y_{l,y} - \mu_{l,y}^{(m)}\big)^2. \tag{5.23}$$

In the following, ℓ_{TNLL} refers only to the heteroscedastic case and ℓ_{MSE} is used as an additional loss term to stabilize training. Empirically, the best performance was found when using the following mixture of different loss terms:

$$\begin{aligned}\ell(y, m, \lambda) =& w_{\text{CE}}\ell_{\text{CE}}\big(m, \alpha^{(m)}\big) + w_{\text{TNLL}}\ell_{\text{TNLL}}\big(y, \tau^{(m)}\big) \\ &+ w_{\text{MSE}}\ell_{\text{MSE}}\big(y, \mu^{(m)}\big) + w_{\text{LMSE}}\ell_{\text{LMSE}}\big(y, \mu^{(m)}\big),\end{aligned} \tag{5.24}$$

with weighting factors $w_{\text{CE}} = 20$, $w_{\text{LMSE}} = 20\,\text{m}^{-2}$, $w_{\text{MSE}} = 1\,\text{m}^{-2}$, and $w_{\text{TNLL}} = 0.01$. In comparison to ℓ_{MSE}, the lateral mean squared error loss ℓ_{LMSE} is the mean squared value of only the lateral error $y_{l,y} - \mu_{l,y}^{(m)}$. In the probabilistic picture, adding ℓ_{LMSE} can be interpreted as assuming less uncertainty for the lateral part $\sigma_{y,l}^{(m)} < \sigma_{x,l}^{(m)}$.

5.4 Experiments

5.4.1 Evaluation Metrics

In this section, evaluation metrics are defined for 1) the maneuver and 2) the trajectory prediction.

1) For the maneuver prediction, in each time step t_k a cut-in $\hat{m}_{\tilde{\alpha}}(t_k) = \text{c}$ or a passing maneuver $\hat{m}_{\tilde{\alpha}}(t_k) = \text{p}$ is assigned by thresholding the model's probability for a cut-in $\alpha^{(\text{c})}$:

$$\hat{m}_{\tilde{\alpha}}(t_k) = \begin{cases} \text{c} & \text{if } \alpha^{(\text{c})}(x_k) \geq \tilde{\alpha}, \\ \text{p} & \text{if } \alpha^{(\text{c})}(x_k) < \tilde{\alpha}, \end{cases} \tag{5.25}$$

with $\tilde{\alpha}$ a specifiable threshold. Next, the predictions of a scenario are summarized in one single event. This results in a scenario-wise evaluation and supports the interpretability of the evaluation. For a cut-in scenario, a true positive (tp) is defined if and only if the model predicts the cut-in at least $T_{\min} = 0.6$ s and maximally $T_{\max} = 12$ s before the labeled cut-in time t_{c}:

$$\text{tp} \Leftrightarrow \exists t_k \in [t_c\text{-}T_{\max}, t_c\text{-}T_{\min}] : \hat{m}_{\tilde{\alpha}}(t_k) = \text{c}. \tag{5.26}$$

A true positive is visualized in Figure 5.11. Otherwise, the cut-in scenario is evaluated as a false negative (fn):

$$\text{fn} \Leftrightarrow \forall t_k \in [t_c\text{-}T_{\max}, t_c\text{-}T_{\min}] : \hat{m}_{\tilde{\alpha}}(t_k) = \text{p}. \tag{5.27}$$

For a passing scenario, a true negative (tn) is defined if and only if the model predicts a passing maneuver for all time steps of the scenario:

$$\text{tn} \Leftrightarrow \forall t_k \in [t_{\text{s}}, t_{\text{e}}] : \hat{m}_{\tilde{\alpha}}(t_k) = \text{p}. \tag{5.28}$$

Otherwise, the passing scenario is evaluated as a false positive (fp):

$$\text{fp} \Leftrightarrow \exists t_k \in [t_{\text{s}}, t_{\text{e}}] : \hat{m}_{\tilde{\alpha}}(t_k) = \text{c}. \tag{5.29}$$

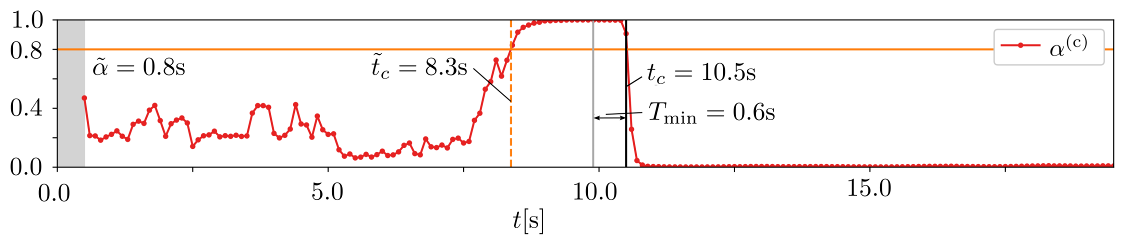


Figure 5.11: For a correctly predicted cut-in, *i.e.* a true positive, the figure shows the model's cut-in probability $\alpha^{(c)}$ over time t. First, the cut-in probability is on a relatively low level. Then, the probability increases rapidly and overshoots the threshold of $\tilde{\alpha} = 0.8$ at $\tilde{t}_c = 8.3$ s. This time point $\tilde{t}_c$ of prediction happens $T_{\text{gain}} = 2.2$ s before the target vehicle crosses the lane marking and is assigned to the ego lane at $t_c = 10.5$ s. The time gain of $T_{\text{gain}} = 2.2$ s is above the minimum value of $T_{\min} = 0.6$ s. Therefore, this scenario is evaluated as a true positive. If the prediction would be a false negative, the cut-in probability $\alpha^{(c)}$ would be below the threshold $\tilde{\alpha}$ for all time steps in the window $[t_c - T_{\max}, t_c - T_{\min}]$, with $T_{\max} = 12$ s and $T_{\min} = 0.6$ s.

Based on these definitions, the following quantities are considered as evaluation metrics. The *precision* is the ratio of the number of all correctly detected cut-ins divided by the number of all detected cut-ins including the false positives:

$$\text{precision} = \frac{\#\text{tp}}{\#\text{tp} + \#\text{fp}}. \tag{5.30}$$

The *recall* is the ratio of the number of all correctly detected cut-ins divided by the number of all actual cut-ins including the false negatives:

$$\text{recall} = \frac{\#\text{tp}}{\#\text{tp} + \#\text{fn}}. \tag{5.31}$$

The trade-off between precision and recall is adjusted with the threshold $\tilde{\alpha}$. In this thesis, the threshold $\tilde{\alpha}$ is increased in steps of 0.1 until a minimum precision of 94% is reached. Furthermore, the mean time gain is considered as evaluation metric. For a true positive, the first time of a cut-in prediction in the relevant time window is denoted with $\tilde{t}_\text{c}$ and the time gain is defined as $T_{\text{gain}} = t_\text{c} - \tilde{t}_\text{c}$. Finally, the mean time gain is determined over all true positives:

$$\langle T_{\text{gain}} \rangle = \langle t_\text{c} - \tilde{t}_\text{c} \rangle. \tag{5.32}$$

Note that the cut-in time t_c is defined as the time point when the target vehicle has crossed the lane marking with its center and the target vehicle is assigned to the ego lane. Thereby, the metric represents the time gain for a downstream system, *e.g.* an ACC. Additionally, the time gain is computed *w.r.t.* the time when only the center of the target vehicle has crossed the lane marking independently of the lane assignment. In the following, the former definition is meant if not specified otherwise.

2) For evaluating the trajectory prediction, different metrics are measured at random time steps within the scenarios. The sampling procedure of the time steps is analogous to the sampling step in the preprocessing, which is described in Section 5.4.2. For the sampled time steps, the two trajectories $\mu^{(\text{p})}, \mu^{(\text{c})}$ are predicted and the winner trajectory $\mu^{(\hat{m})}$ is determined by selecting the maneuver with the higher probability $\hat{m} = \hat{m}_{\tilde{\alpha}=0.5}$. Next, the

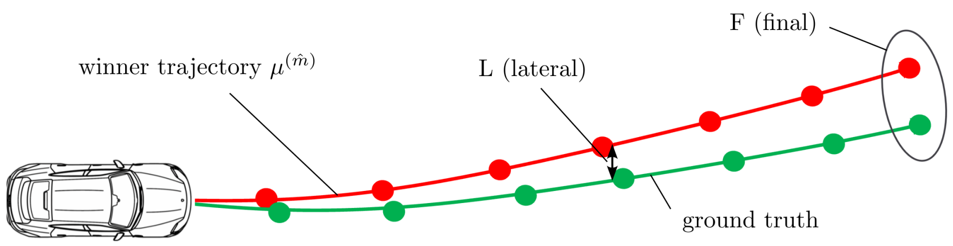


Figure 5.12: The trajectory metrics measure distances between the more probable of the two predicted trajectories, the so-called winner trajectory $\mu^{(\hat{m})}$, and the ground truth y. For the distance measures, the Mean Squared Error (MSE) and the mean Euclidean distance, here referred to as Mean Displacement Error (MDE), are used and evaluated on waypoint-level. Additionally, the prefix L for lateral indicates that only lateral components are considered and the prefix F for final denotes that only the waypoints in the most distance future are evaluated. *E.g.* LMDE is the Lateral Mean Displacement Error.

distance between the winner trajectory and the ground truth y is measured with different metrics ℓ_{X} as visualized in Figure 5.12. Finally, the mean value of a distance metric over all samples results in the evaluation metric:

$$\mathrm{X} = \langle \ell_{\mathrm{X}}(y, \mu^{(\hat{m})}) \rangle. \tag{5.33}$$

The following evaluation metrics X are considered:

- Mean Squared Error (MSE): $\ell_{\mathrm{MSE}}(y, \mu^{(\hat{m})}) = \frac{1}{2L} \sum_l \big(y_{l,x} - \mu_{l,x}^{(\hat{m})}\big)^2 + \big(y_{l,y} - \mu_{l,y}^{(\hat{m})}\big)^2$,
- Mean Displacement Error (MDE): $\ell_{\mathrm{MDE}}(y, \mu^{(\hat{m})}) = \frac{1}{L} \sum_l \|y_l - \mu_l^{(\hat{m})}\|_2$,
- Final Mean Displacement Error (FMDE): $\ell_{\mathrm{FMDE}}(y, \mu^{(\hat{m})}) = \|y_L - \mu_L^{(\hat{m})}\|_2$,
- Lateral Mean Displacement Error (LMDE): $\ell_{\mathrm{LMDE}}(y, \mu^{(\hat{m})}) = \frac{1}{L} \sum_l |y_{l,y} - \mu_{l,y}^{(\hat{m})}|$,
- Final Lateral Mean Displacement Error (FLMDE): $\ell_{\mathrm{FLMDE}}(y, \mu^{(\hat{m})}) = |y_{L,y} - \mu_{L,y}^{(\hat{m})}|$.

In comparison to the MSE, the MDE measures the mean Euclidean distance $\|\cdot\|_2$ between trajectory waypoints. Analogously to the loss, the prefix L denotes that the corresponding distance metric is restricted only to the lateral components. A second prefix F is used if the distance metric is applied only to the final trajectory waypoint, *i.e.* the trajectory point in the most distant future $l = L$. Furthermore, the distance metrics ℓ_{LMDE} and ℓ_{FLMDE} are applied on both trajectory modes $\mu^{(\mathrm{p})}, \mu^{(\mathrm{c})}$ instead of the winner trajectory $\mu^{(\hat{m})}$ and the ground truth y. This allows the evaluation of the lateral separation of the two trajectories. For these metrics, the prefix M- for mode separation is used and followed by the name of the distance metric X:

$$\text{M-X} = \langle \ell_{\mathrm{X}}(\mu^{(\mathrm{p})}, \mu^{(\mathrm{c})}) \rangle. \tag{5.34}$$

E.g. M-FLMDE is the final lateral mean displacement between the predicted cut-in and passing trajectory.

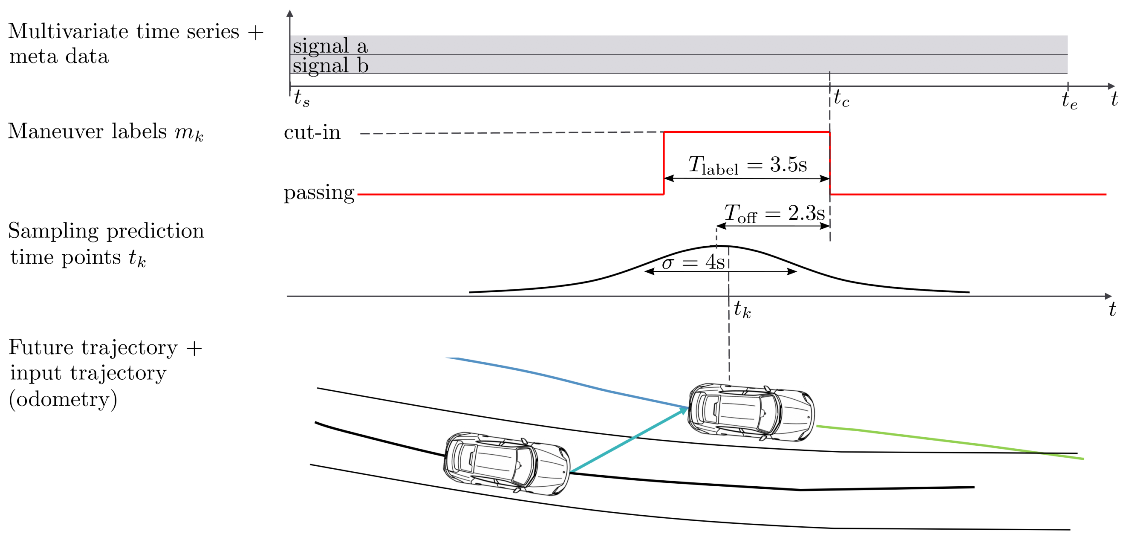

Multivariate time series + meta data
signal a
signal b
t_s
t_c
t_e
t
Maneuver labels m_k
cut-in
passing
$T_{\text{label}} = 3.5\text{s}$
$T_{\text{off}} = 2.3\text{s}$
Sampling prediction time points t_k
$\sigma = 4\text{s}$
t_k
t
Future trajectory + input trajectory (odometry)

Figure 5.13: The figure shows an overview of the preprocessing steps. The steps generate the input sequence x_k of the model, set the label of the maneuver $m_k = \text{p/c}$, and determine the ground truth of the future trajectory y_k. For training and for the evaluation of the trajectory metrics, procedures are defined to sample time points t_k randomly from scenarios. For a cut-in scenario, the maneuver labels m_k are set to cut-in in the time window that covers $T_{\text{label}} = 3.5$ s before the target vehicle becomes the CIPV at t_c. For the other time steps, the maneuver labels are set to passing. In order to sample more frequently from the time interval with cut-in labels, the time points t_k are sampled from a Gaussian distribution with a mean value $T_{\text{off}} = 2.3$ s before t_c and a standard deviation of $\sigma = 4$ s. If the target vehicle passes, all maneuver labels are set to passing $m_k = \text{p}$ and the time point t_k for prediction is sampled uniformly between the start t_s and the end time t_e of the scenario. For the trajectory of the target vehicle, an odometry algorithm is performed on motion data of the ego vehicle and combined with measurements of the relative position of the target vehicle *w.r.t.* the ego vehicle (compare Figure 5.14 and Algorithm 3).

5.4.2 Preprocessing

If a scenario s is selected from the data set, the preprocessing steps enable to sample a time point t_k and to compute the input sequence x_k, the future trajectory y_k, and the maneuver label m_k automatically. An overview of the preprocessing steps is shown in Figure 5.13. The preprocessing steps leverage the information of the scenario descriptor s from the data set. The descriptor s includes pointers to the file f with the multivariate sensor data, the start t_s and the end time t_e of the scenario in the file, an identifier i of the target vehicle, and a label $e = \text{p/c}$ for whether the target vehicle passes the ego vehicle or performs a cut-in maneuver. If the target vehicle cuts in, additionally the time point when the target vehicle becomes CIPV t_c is part of the scenario descriptor s.

Sampling time points t_k: In training and for the evaluation of the trajectory metrics, time points for the prediction t_k are sampled randomly from a scenario. For passing scenarios, the time points are sampled from a uniform distribution over the total scenario $[t_{\text{s}}, t_{\text{e}}]$. For cut-in scenarios, a Gaussian with mean value $\mu = t_c – T_{\text{off}}$ ($T_{\text{off}} = 2.3$ s) and standard deviation $\sigma = 4$ s is used to sample more frequently when cut-in maneuvers become predictable.

Maneuver labels m_k: For all time steps of a passing scenario ($e = \mathrm{p}$), the maneuver labels m_k are set to passing:

$$e = \mathrm{p} \Rightarrow \forall t_k \in [t_s, t_e] : m_k = \mathrm{p}. \tag{5.35}$$

A cut-in scenario ($e = \mathrm{c}$) includes also many time steps with the label passing. Only for a duration of $T_{\text{label}} = 3.5$ s before t_c, the labels m_k are set to cut-in:

$$e = \mathrm{c} \Rightarrow m_k = \begin{cases} \mathrm{c} & \text{for } t_k \in [t_c - T_{\text{label}}, t_c], \\ \mathrm{p} & \text{else.} \end{cases} \tag{5.36}$$

Input sequence x_k and future trajectory y_k: Except for the target vehicle's trajectory, all features of the input sequence x_k are either directly available in the signals or can be computed from several signals with straightforward expressions. Thus, the focus is on computing the trajectory. First, the target vehicle's trajectory $(x_{\text{tar},k}, y_{\text{tar},k})^\top_{k \in I}$ is determined in a global reference system, which is explained in the next paragraph. Then, the past and future trajectory are transformed in a target vehicle-centric reference system for arbitrary time steps t_k by applying a translation and a rotation:

$$\text{past trajectory :} \begin{pmatrix} \tilde{x}_{\text{tar},k-h} \\ \tilde{y}_{\text{tar},k-h} \end{pmatrix} \leftarrow R(-\theta_{\text{tar},k}) \left[\begin{pmatrix} x_{\text{tar},k-h} \\ y_{\text{tar},k-h} \end{pmatrix} - \begin{pmatrix} x_{\text{tar},k} \\ y_{\text{tar},k} \end{pmatrix} \right] \text{ with } h \in \{0, \dots, H\}, \tag{5.37}$$

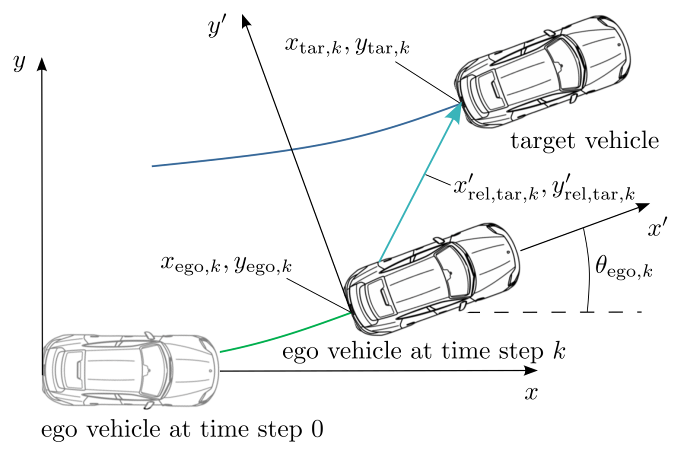
$$\text{future trajectory:} \begin{pmatrix} \tilde{x}_{\text{tar},k+l} \\ \tilde{y}_{\text{tar},k+l} \end{pmatrix} \leftarrow R(-\theta_{\text{tar},k}) \left[\begin{pmatrix} x_{\text{tar},k+l} \\ y_{\text{tar},k+l} \end{pmatrix} - \begin{pmatrix} x_{\text{tar},k} \\ y_{\text{tar},k} \end{pmatrix} \right] \text{ with } l \in \{1, \dots, L\}.$$

In the formula, $R(\cdot)$ is the 2D rotation matrix, $(\tilde{x}_{\text{tar},k}, \tilde{y}_{\text{tar},k})^\top$ is the position in the target vehicle's reference system, $(x_{\text{tar},k}, y_{\text{tar},k})^\top$ is the position in the global reference system, and $\theta_{\text{tar},k}$ is the heading angle in the global reference frame.

Target vehicle's trajectory in a global reference frame: The target vehicle's trajectory is computed with odometry and from the relative positions of the target vehicle *w.r.t.* the ego vehicle. The twofold process is visualized in Figure 5.14 and the explicit computations are summarized in Algorithm 3. First, the trajectory of the ego vehicle is computed with odometry in a global reference frame that is fixed to the ego vehicle's position and orientation at the initial time step zero. Next, the waypoints $(x_{\text{ego},k}, y_{\text{ego},k})^\top$ and the heading angles $\theta_{\text{ego},k}$ are updated iteratively with the update rule:

$$\begin{aligned} x_{\text{ego},k+1} &= x_{\text{ego},k} + dx_{\text{ego},k}, \\ y_{\text{ego},k+1} &= y_{\text{ego},k} + dy_{\text{ego},k}, \\ \theta_{\text{ego},k+1} &= \theta_{\text{ego},k} + d\theta_{\text{ego},k}, \end{aligned} \tag{5.38}$$

with $dx_{\text{ego},k}, dy_{\text{ego},k}$ for the driven distance, and $d\theta_{\text{ego},k}$ the change in the heading angle between time steps k and $k+1$. To support readability, in the following the subscript for the ego vehicle is skipped. The driven distance between two time steps dx'_k, dy'_k is computed in the reference system of the ego vehicle at time step k and then transformed to the distance dx_k, dy_k in the reference system of the ego vehicle at time step zero by



y'
$x_{\mathrm{tar},k}, y_{\mathrm{tar},k}$
y
target vehicle
$x'_{\mathrm{rel,tar},k}, y'_{\mathrm{rel,tar},k}$
x'
$x_{\mathrm{ego},k}, y_{\mathrm{ego},k}$
$\theta_{\mathrm{ego},k}$
ego vehicle at time step k
x
ego vehicle at time step 0

Figure 5.14: The trajectory of the target vehicle (blue) is computed in two steps. First, the trajectory of the ego vehicle (green) is determined in a global reference frame that is fixed to the ego vehicle's position and orientation at the initial time step zero. Second, the waypoints of target vehicle's trajectory $x_{\mathrm{tar},k}, y_{\mathrm{tar},k}$ are given by adding the relative positions between target and ego vehicle $x'_{\mathrm{rel,tar},k}, y'_{\mathrm{rel,tar},k}$ to the ego vehicle's waypoints $x_{\mathrm{ego},k}, y_{\mathrm{ego},k}$ after rotating with the heading angle $\theta_{\mathrm{ego},k}$ at the corresponding time step. The relative positions $x'_{\mathrm{rel,tar},k}, y'_{\mathrm{rel,tar},k}$ are measured in the reference system that is fixed to the ego vehicle's position and orientation at time step k.

applying a rotation. Both reference systems are visualized in Figure 5.14. In the ego vehicle's reference frame at time step k, the ego vehicle's velocity can be written by its absolute value v and its heading angle θ':

$$\begin{aligned} v'_x(t) &= v(t)\cos\big(\theta'(t)\big), \\ v'_y(t) &= v(t)\sin\big(\theta'(t)\big). \end{aligned} \tag{5.39}$$

Between two time steps $[t_k, t_{k+1}]$, the heading angle and the absolute velocity is approximated with a first order Taylor expansion:

$$\begin{aligned} \theta'(t') &= \omega_k t', \\ v(t') &= v_k + a_k t', \end{aligned} \tag{5.40}$$

with $\omega_k = \dot{\theta}(t_k)$ the angular velocity, $a_k = \dot{v}(t_k)$ the longitudinal acceleration, and $t' = t - t_k \in [0, dt]$ the time starting at time step k. Using this approximation, the driven distance of the ego vehicle between the two time steps can be computed as follows:

$$\begin{aligned} dx'_k &= \int_0^{dt} v'_x(t')dt' = \int_0^{dt} (v_k + a_k t')\cos(\omega_k t')dt' \\ &= \frac{v_k + a_k dt}{\omega_k}\sin(\omega_k dt) + \frac{a_k}{\omega_k^2}\big(\cos(\omega_k dt) - 1\big) \\ &\overset{|\omega_k|<\epsilon}{\approx} v_k dt + \frac{1}{2}a_k dt^2, \end{aligned} \tag{5.41}$$

Algorithm 3 Computation of the ego and the target vehicle's trajectory.

```
 1: Input
 2: $v \leftarrow (v_0, \dots, v_K)$    ▷ absolute velocity of the ego vehicle
 3: $a \leftarrow (a_0, \dots, a_K)$    ▷ longitudinal acceleration of the ego vehicle
 4: $\omega \leftarrow (\omega_0, \dots, \omega_K)$    ▷ angular velocity of the ego vehicle's heading
 5: $x'_{\mathrm{rel,tar}} \leftarrow (x'_{\mathrm{rel,tar},0}, \dots, x'_{\mathrm{rel,tar},K})$,    ▷ relative position of target vehicle w.r.t. ego vehicle
 6: $y'_{\mathrm{rel,tar}} \leftarrow (y'_{\mathrm{rel,tar},0}, \dots, y'_{\mathrm{rel,tar},K})$
 7:
 8: Output
 9: $x_{\mathrm{ego}} \leftarrow (x_{\mathrm{ego},0}, \dots, x_{\mathrm{ego},K})$    ▷ trajectory of the ego vehicle
10: $y_{\mathrm{ego}} \leftarrow (y_{\mathrm{ego},0}, \dots, y_{\mathrm{ego},K})$
11: $\theta_{\mathrm{ego}} \leftarrow (\theta_{\mathrm{ego},0}, \dots, \theta_{\mathrm{ego},K})$    ▷ heading angle of the ego vehicle
12: $x_{\mathrm{tar}} \leftarrow (x_{\mathrm{tar},0}, \dots, x_{\mathrm{tar},K})$    ▷ trajectory of the target vehicle
13: $y_{\mathrm{tar}} \leftarrow (y_{\mathrm{tar},0}, \dots, y_{\mathrm{tar},K})$
14:
15: $x_{\mathrm{ego},0} \leftarrow 0$, $y_{\mathrm{ego},0} \leftarrow 0$, $\theta_{\mathrm{ego},0} \leftarrow 0$    ▷ initial position and heading angle of the ego vehicle
16: $(x_{\mathrm{tar},0}, y_{\mathrm{tar},0})^\top \leftarrow (x'_{\mathrm{rel,tar},0}, y'_{\mathrm{rel,tar},0})^\top$    ▷ initial position of the target vehicle
17: for $k = 0$ to $K - 1$ do
18:     if $|\omega_k| < 10^{-4}$ then
19:         $dx'_k \leftarrow v_k dt + \frac{1}{2} a_k dt^2$
20:         $dy'_k \leftarrow \big(\frac{1}{2} v_k + \frac{1}{3} a_k dt\big) \omega_k dt^2$
21:     else
22:         $dx'_k \leftarrow \frac{v_k + a_k dt}{\omega_k} \sin(\omega_k dt) + \frac{a_k}{\omega_k^2} \big(\cos(\omega_k dt) - 1\big)$
23:         $dy'_k \leftarrow -\frac{v_k + a_k dt}{\omega_k} \cos(\omega_k dt) + \frac{a_k}{\omega_k^2} \sin(\omega_k dt) + \frac{v_k}{\omega_k}$
24:     end if
25:     $(dx_k, dy_k)^\top \leftarrow R(\theta_{\mathrm{ego},k})(dx'_k, dy'_k)^\top$    ▷ transformation to global reference system
26:     $d\theta_{\mathrm{ego},k} \leftarrow \omega_k dt$    ▷ change in heading angle
27:     $x_{\mathrm{ego},k+1} \leftarrow x_{\mathrm{ego},k} + dx_k$
28:     $y_{\mathrm{ego},k+1} \leftarrow y_{\mathrm{ego},k} + dy_k$
29:     $\theta_{\mathrm{ego},k+1} \leftarrow \theta_{\mathrm{ego},k} + d\theta_k$
30:     $(x_{\mathrm{tar},k+1}, y_{\mathrm{tar},k+1})^\top \leftarrow (x_{\mathrm{ego},k+1}, y_{\mathrm{ego},k+1})^\top + R(\theta_{\mathrm{ego},k+1})(x'_{\mathrm{rel,tar},k+1}, y'_{\mathrm{rel,tar},k+1})^\top$
31: end for
```

$$\begin{aligned} dy'_k &= \int_0^{dt} v'_y(t')dt' = \int_0^{dt} (v_k + a_k t') \sin(\omega_k t')dt' \\ &= -\frac{v_k + a_k dt}{\omega_k} \cos(\omega_k dt) + \frac{a_k}{\omega_k^2} \sin(\omega_k dt) + \frac{v_k}{\omega_k} \\ &\overset{|\omega_k|<\epsilon}{\approx} \big(\frac{1}{2} v_k + \frac{1}{3} a_k dt\big) \omega_k dt^2. \end{aligned} \tag{5.42}$$

For small angular velocities $|\omega_k| < \epsilon = 10^{-4}$, the given approximations are precise and used. The driven distance between two time steps in the reference system that is fixed to the ego vehicle at time step zero is given by applying a rotation with the heading angle θ_k:

$$\begin{pmatrix} dx_{\mathrm{ego},k} \\ dy_{\mathrm{ego},k} \end{pmatrix} = R(\theta_k) \begin{pmatrix} dx'_k \\ dy'_k \end{pmatrix}. \tag{5.43}$$

Hyperparameter	Value range	Best value	Description
N_{batch}	$8, 16, 32, \ldots, 256$	128	Batch size.
step_{max}	$100, 101, \ldots, 1000$	429	Warm start gradient descent steps.
w	$0.156 \cdot (\sqrt{10})^n : n{=}-4, -3, \ldots, 1$	0.049	Learning rate factor.
d_{context}	$8, 16, 32, \ldots, 512$	256	Dimension of context vector.
$d_{\text{in, emb}}$	$8, 16, 32, \ldots, 512$	32	Dimension of input embedding.
$d_{\text{h, enc}}$	$8, 16, 32, \ldots, 512$	128	Dimension of hidden state (encoder).
$d_{\text{h, dec}}$	$8, 16, 32, \ldots, 512$	512	Dimension of hidden state (decoder).

Table 5.3: The table summarizes the hyperparameters, their discrete value ranges, and the found optimal values. The hyperparameters that refer to dimensions in the NN architecture are visualized in Figure 5.9.

The update of the heading angle is $d\theta_{\text{ego},k} = \omega_k dt$. Given the increments $dx_{\text{ego},k}$, $dy_{\text{ego},k}$ and $d\theta_{\text{ego},k}$, Equations (5.38) allow to iteratively compute the ego vehicle's trajectory. Finally, the target vehicle's trajectory is obtained by shifting the ego vehicle's trajectory with the rotated relative distances:

$$\begin{pmatrix} x_{\text{tar},k} \\ y_{\text{tar},k} \end{pmatrix} = \begin{pmatrix} x_{\text{ego},k} \\ y_{\text{ego},k} \end{pmatrix} + R(\theta_{\text{ego},k}) \begin{pmatrix} x'_{\text{rel,tar},k} \\ y'_{\text{rel,tar},k} \end{pmatrix}. \tag{5.44}$$

The rotation is required to transform the relative positions to the reference frame that is fixed to the ego vehicle at time step zero.

5.4.3 Training

For training, the Adam optimizer [Kingma and Ba, 2014] is used with a standard parameter choice of $\beta_1 = 0.9$ and $\beta_2 = 0.98$. The explicit update rule of Adam and the meaning of the parameters are given in Equation (2.19). Furthermore, the learning rate η is set with a warm start scheduler [Vaswani et al., 2017]:

$$\eta = w/\sqrt{\text{step}_{\text{max}}} \cdot \min(\text{step}/\text{step}_{\text{max}}, \sqrt{\text{step}_{\text{max}}/\text{step}}), \tag{5.45}$$

with step for the current gradient descent step, w a learning rate scale factor and step_{max} the step with the maximum learning rate of $w/\sqrt{\text{step}_{\text{max}}}$. During training, the loss on the validation set is computed every 100 steps, the model weights are saved if the model has improved, and the learning rate is reduced with a factor of 10 if the validation loss has not improved four times in a row.

Additionally to the model weights, there are hyperparameters that must be tuned separately. An overview of the hyperparameters is given by Table 5.3. Instead of manual tuning, the hyperparameters are optimized systematically with a hyperparameter optimization algorithm. For the objective of the optimization, the loss in Equation (5.24) is evaluated on the validation set. The Tree-structured Parzen Estimator (TPE) [Bergstra et al., 2011] with Hyperband pruning [Li et al., 2017a] is chosen as the optimization algorithm and the algorithm is implemented with the framework Optuna [Akiba et al., 2019]. TPE is a strategy for sequential model-based global optimization. Instead of directly determining the surrogate $p(y|x)$ (y is the score, x is the hyperparameter configuration), $p(x|y)$ and $p(y)$ are modeled and Bayesian theorem is applied. TPE is particularly suitable if evaluating hyperparameters is expensive. Furthermore, Hyperband pruning terminates training of unpromising configurations. Thus, the principle is the allocation of resources for the promising trials. The explained hyperparameter optimization is performed on the bimodal LSTM-based model without constraints, a duration of cut-in labels of $T_{\text{label}} = 3.5\,\text{s}$, and a weight of $w_{\text{LMSE}} = 20\,\text{m}^{-2}$ for the lateral part of the loss. In the following, this model is referred to as the basic model and Table 5.3 shows the found optimal hyperparameters. The optimal values of the basic model are used for further trainings with the following modifications:

- A maneuver classification is performed purely and the trajectory prediction by the decoder is excluded.
- A trajectory prediction with a single mode is performed purely and the maneuver prediction is excluded.
- The duration of the cut-in labels is reduced from $T_{\text{label}} = 3.5\,\text{s}$ to $T_{\text{label}} = 2\,\text{s}$.
- The lateral mean squared error part of the loss is removed, *i.e.* $w_{\text{LMSE}} = 0\,\text{m}^{-2}$ in Equation (5.24).
- The soft constraints are applied with $w_{\text{C}} = 1\,\text{m}^{-1}$ and $v_{\text{LMAD}} = 0.75\,\text{m}^{-1}$ in Equation (5.15). Empirically, these values have been found for an optimal trade-off between the separation of the cut-in and passing trajectory and the accuracy.
- ConstraintNet and hard constraints are applied and enforce a minimum lateral movement of $b = 0.7\,\text{m}$ over 3 s for a cut-in maneuver (see Equation (5.17)). The choice of $b = 0.7\,\text{m}$ is based on an analysis of cut-in trajectories.

The modifications are applied to analyze the influences of the design choices and the constraints.

5.4.4 Results

Table 5.4 and Table 5.5 show the results of the basic and the modified LSTM-based models. The numbers result from evaluating the models on the test set with the metrics of Section 5.4.1. The mean values and the standard deviations are determined from five trainings with different and random weight initializations. For comparison, additionally the following two rule-based models are introduced. 1) For the maneuver prediction, a cut-in detection is defined if the lateral distance of the target vehicle's center to the lane marking is less than 0.2 m. Analogously to adjusting the threshold $\tilde{\alpha}$ for the cut-in detection of the LSTM-based models, the threshold of the baseline is set such that just a minimum precision of 94% is reached. 2) For the trajectory prediction, a constant velocity model is applied as baseline, *i.e.* the trajectory is extrapolated based on the last estimation of the target vehicle's velocity.

Model	Precision [%]	Recall [%]	Time Gain [s]
thresholding (0.2 m)	94.4	68.7	1.00 (0.28)
LSTM (pure classification)	**95.8 ± 0.6**	96.3 ± 0.4	2.68 ± 0.05 (2.10 ± 0.04)
LSTM (basic model)	95.3 ± 0.7	95.3 ± 0.7	2.70 ± 0.07 (2.13 ± 0.07)
LSTM ($T_{\text{label}} = 2\,\text{s}$)	95.3 ± 1.2	**96.9 ± 0.4**	2.44 ± 0.11 (1.86 ± 0.11)
LSTM ($w_{\text{LMSE}} = 0\,\text{m}^2$)	94.9 ± 0.9	96.1 ± 0.2	2.68 ± 0.07 (2.11 ± 0.07)
LSTM (ConstraintNet)	95.4 ± 0.5	95.5 ± 0.6	**2.77 ± 0.09** (2.19 ± 0.09)
LSTM (soft constraint)	94.8 ± 0.9	95.8 ± 0.5	2.74 ± 0.08 (2.16 ± 0.08)

Table 5.4: For the cut-in prediction, the table shows the evaluation metrics of the LSTM-based models on the test set. For comparison, the results of a rule-based model that thresholds the distance of the target vehicle to the lane marking are shown. Optimal values are written in bold and the time gain is denoted based on two different reference time points. The value without brackets compares the time point of detection with the time point when the target vehicle is assigned to the ego lane. The value in brackets represents the time gain *w.r.t.* the time point when the target vehicle's center has crossed the lane marking.

Table 5.4 shows the results of the cut-in prediction. All LSTM-based models achieve more than twice the time gain of the baseline and improve the recall significantly. For the LSTM-based model with hard constraints and the baseline, Figure 5.15 shows the distributions over the time gains of the true positives. Note that a true positive is defined when a cut-in is detected at least 0.6 s before the actual cut-in. For the baseline, roughly a third of all actual cutting-in vehicles have a time gain below 0.6 s, which results in a recall of 68.7%. The baseline's average time gain of the true positives is 1.0 s. Choosing the baseline's threshold above 0.2 m would improve the time gain and the recall, however, at the expense of a higher number of false positives and the precision would drop below the required value of 94%. The precison of all LSTM-based models is also adjusted to be slightly above 94% by setting the detection threshold $\tilde{\alpha}$ correspondingly. For the models with the basic label duration of $T_{\text{label}} = 3.5$ s, this precision is reached with a threshold of $\tilde{\alpha} = 0.8$. In comparison, the model with the shorter label duration of $T_{\text{label}} = 2$ s achieves the same precision with a significantly lower threshold of $\tilde{\alpha} = 0.4$. As expected and due to the lower threshold, the model with the short label duration achieves the highest recall of 96.9±0.4%. However, the models with the standard label duration achieve only slightly lower recalls between 95.3±0.7% and 96.3±0.4%. Furthermore, the label duration influences the time gain. The models with $T_{\text{label}} = 3.5$ s achieve a time gain of about 2.7±0.1s. In comparison, the time gain of the model with $T_{\text{label}} = 2$ s decreases to 2.4 ± 0.1s. Interestingly, the evaluation metrics of the model with pure maneuver prediction are comparable to the results of the models with additional trajectory prediction. Probably, the cut-in and the trajectory prediction task share most of the learned features and no trade-off is required when learning both tasks jointly.

For the trajectory prediction, the discussion of the results is twofold. On the one hand, the accuracy is measured with the Mean Squared Error (MSE), Mean Displacement Error (MDE), Final Mean Displacement Error (MDE), and Lateral Mean Displacement Error (LMDE). On the other hand, the lateral separation between the cut-in and the passing mode

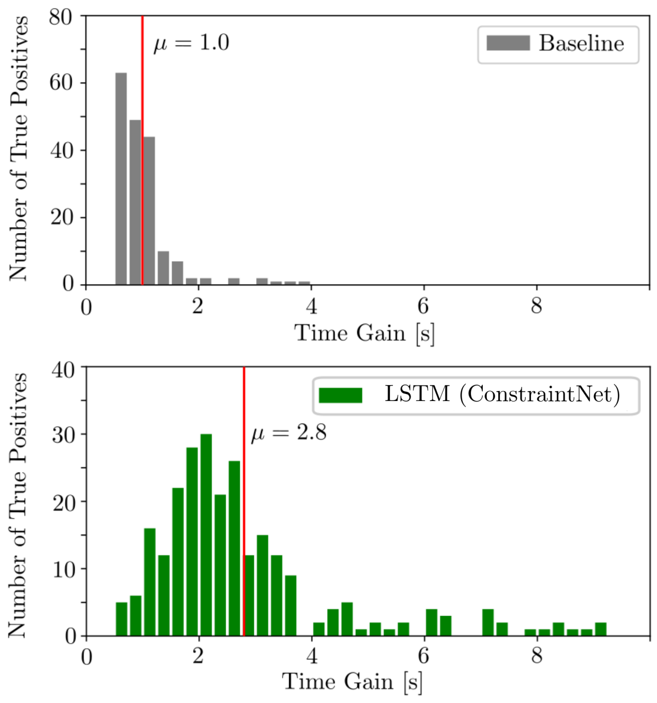


Figure 5.15: For the baseline and ConstraintNet, the figure shows the distribution of time gains on the test set. *Top*: The cut-in maneuvers are detected by thresholding the distance of the target vehicle to the lane marking with 0.2 m. *Bottom*: The cut-in maneuvers are detected with ConstraintNet, *i.e.* the LSTM-based model with hard constraints, and a threshold of $\tilde{\alpha} = 0.8$. True positives are defined as cut-in detections with a time gain of at least 0.6 s. Therefore, the histograms start above this value. ConstraintNet achieves a mean time gain of $\mu = 2.8$ s. This is more than twice as high as the baseline's time gain of $\mu = 1.0$ s.

is analyzed with the metrics Lateral Mean Displacement between Modes (M-LMDE) and Final Lateral Mean Displacement between Modes (M-FLMDE). The mode separation is discussed in the context of hard and soft constraints and the duration of cut-in labels. Regarding the prediction accuracy, the LSTM-based models outperform the constant velocity model clearly in all accuracy metrics. The best performing models achieve a Lateral Mean Displacement Error (LMDE) of 0.36 ± 0.01m, which is only half the error of the constant velocity model. Thus, the input features comprise information that indicates cut-in or passing maneuvers far beyond linear extrapolation. Without an additional weighting of the lateral part in the loss ($w_{\text{LMSE}} = 0\,\text{m}^{-2}$), the LMDE is downgraded to 0.42 ± 0.01m, which can be explained as follows. The longitudinal coordinates are of a larger order of magnitude than the lateral coordinates and therefore the loss is dominated by the longitudinal part if no additional weighting of the lateral part is included. Furthermore, a model that predicts a single trajectory is trained and evaluated. The uni- and the bimodal approach achieve compareable accuracies in all considered metrics. However, a clear advantage of the bimodal approach is the enhanced interpretability by assigning the modes semantically to a cut-in and a passing maneuver. Furthermore, it is expected that the actual but unknown real uncertainty distribution has at least two modes. Thus, the bimodal model is more realistic than the unimodal model. For analyzing the mode separation of the bimodal approaches, the Lateral Mean Displacement between Modes (M-LMDE) and

Model	MSE [m^2]	MDE [m]	FMDE [m]	LMDE [m]	M-LMDE [m]	M-FLMDE [m]
const. velocity model	6.26	1.77	3.96	0.70	-	-
LSTM (single trajectory)	4.66 ± 0.28	1.57 ± 0.04	**3.24 ± 0.10**	**0.36 ± 0.01**	-	-
LSTM (basic model)	4.62 ± 0.10	1.58 ± 0.01	3.26 ± 0.02	**0.36 ± 0.01**	0.58 ± 0.02	1.33 ± 0.04
LSTM (T_{label} = 2 s)	4.81 ± 0.32	1.62 ± 0.10	3.34 ± 0.23	**0.36 ± 0.01**	0.51 ± 0.04	0.96 ± 0.10
LSTM (w_{LMSE} = 0 m^2)	4.58 ± 0.17	1.58 ± 0.03	3.28 ± 0.05	0.42 ± 0.01	0.61 ± 0.05	1.20 ± 0.12
LSTM (ConstraintNet)	**4.56 ± 0.19**	**1.56 ± 0.03**	**3.24 ± 0.04**	**0.36 ± 0.01**	0.61 ± 0.03	**1.41 ± 0.06**
LSTM (soft constraint)	4.92 ± 0.27	1.63 ± 0.10	3.40 ± 0.25	**0.36 ± 0.01**	**0.65 ± 0.01**	1.34 ± 0.04

Table 5.5: For the trajectory prediction, the table shows the evaluation metrics of the LSTM-based models and the constant velocity model on the test set. The abbreviations of the evaluation metrics are explained in Section 5.4.1. Optimal values are written in bold.

Final Lateral Mean Displacement between Modes (M-FLMDE) are measured between the predicted cut-in and passing trajectory. Larger values of these metrics indicate that the models capture indeed bimodality and have learned the semantic assignment of a cut-in and a passing maneuver. On the one hand, the choice of the label duration T_{label} influences the mode separation. A reduced label duration of $T_{\text{label}} = 2\,\text{s}$ results in a less mean lateral separation of the modes and smaller M-LMDE and M-FLMDE. This is a logical consequence of the fact that the model is supposed to learn the separated cut-in trajectory for a shorter duration. On the other hand, imposing hard and soft constraints enhance mode separation. The model with soft constraints achieves the highest M-LMDE and ConstraintNet with hard constraints the highest M-FLMDE. Consequently, the intended effect of the constraints on the mode separation is observed in the experiments. Furthermore, ConstraintNet achieves the best total performance considering all metrics, whereas the trajectory accuracy decreases slightly with soft constraints. An advantage of the hard constraints is that the objective is unaffected as long as the hard constraint is valid. Based on empirical observations, it is reasonable to assume that the lateral movement during a cut-in maneuver over $T_L = 3\,\text{s}$ is at least $0.7\,\text{m}$ and therefore choosing a hard constraint with $b = 0.7\,\text{m}$ is a valid assumption. Contrary to the hard constraints, the loss term for the soft constraints changes the objective and results in learning a trade-off between mode separation and accuracy. Empirically, an optimal trade-off between performance and mode separation was found with the parameters $w_{\text{C}} = 1\,\text{m}^{-1}$ and $v_{\text{LMAD}} = 0.75\,\text{m}^{-1}$.

Figure 5.16 shows the bimodal prediction of ConstraintNet and the trajectory prediction of the constant velocity model for four characteristic time steps of a cut-in scenario. ConstraintNet predicts roughly equal probabilities for a cut-in and a passing scenario at time $t_k = 7.9\,\text{s}$. The mean values of the two predicted trajectories represent indeed a cut-in and passing maneuver. Additionally to the mean values, the plot shows the uncertainty of the predicted trajectory waypoints with a color encoding of the density. As expected, the model predicts a higher longitudinal than lateral uncertainty and a higher uncertainty for waypoints in the more distant future. Thus, the predicted distributions of the joint bimodal trajectory and cut-in prediction demonstrate an improved interpretability compared to a pure maneuver classification or the prediction of a single trajectory. The trajectory of

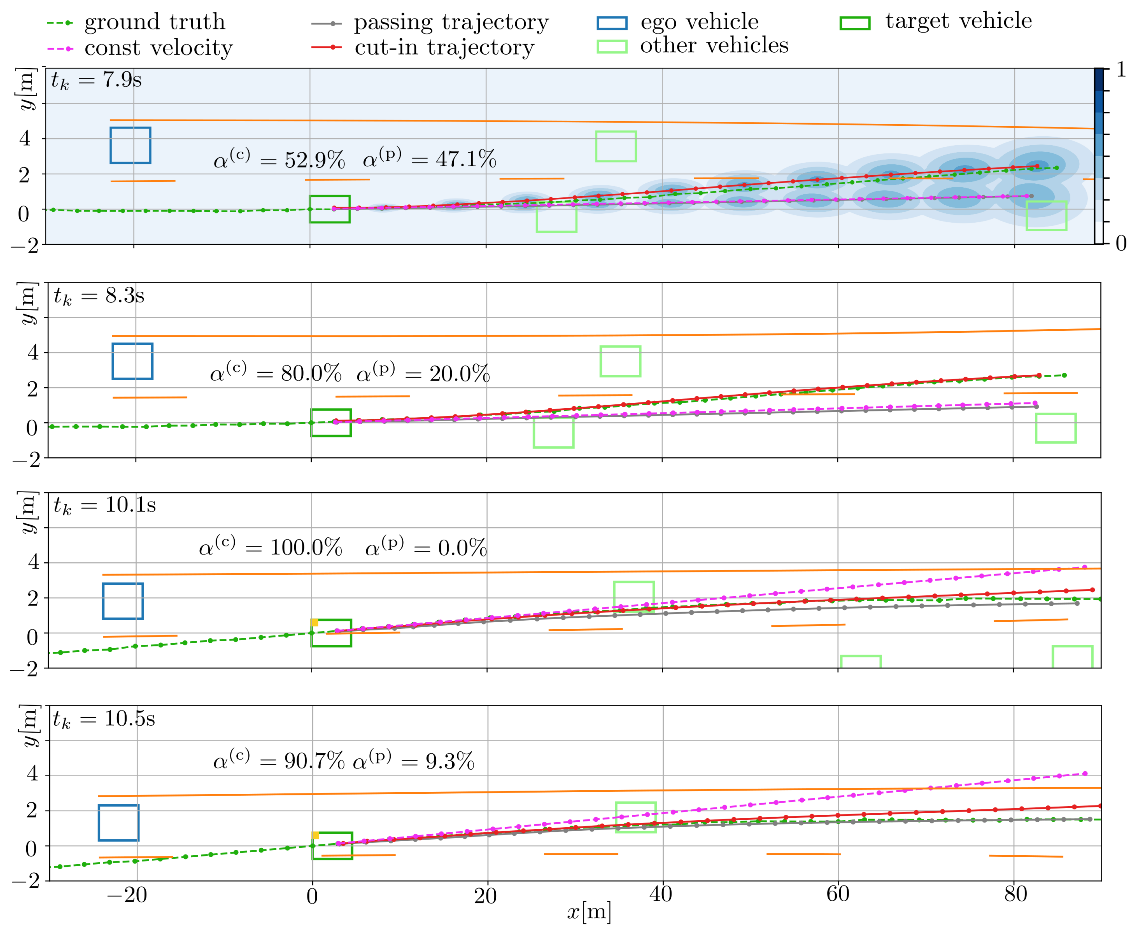


Figure 5.16: For a cut-in scenario, the predictions of ConstraintNet (red, gray) and of the constant velocity model (pink) are visualized in a bird's-eye view at four times t_k. The actual driven trajectory is shown in green. *Scene at* $t_k = 7.9$ s: For both modes, similar probabilities $\alpha^{(p/c)}$ are predicted by ConstraintNet and the heteroscedastic uncertainty of the trajectory waypoints is visualized. The color encodes the predicted Gaussian probability density of the waypoint. For illustration purposes, the density of only every third waypoint is visualized and the peak of the density is normalized. *Scene at* $t_k = 8.3$ s: For the cut-in maneuver, a probability of $\alpha^{(c)} = 80.0\%$ is predicted while the target vehicle is still clearly on the neighboring lane. The trajectory prediction of ConstraintNet is close to the ground truth, whereas the constant velocity model suffers from the simple linear interpolation. *Scene at* $t_k = 10.1$ s: The plot shows the reference point when the target vehicle's center crosses the lane marking. *Scene at* $t_k = 10.5$ s: The second reference time point t_c is defined by the time when the target vehicle is assigned to the lane of the ego vehicle. The figure is based on Brosowsky et al. [2021c].

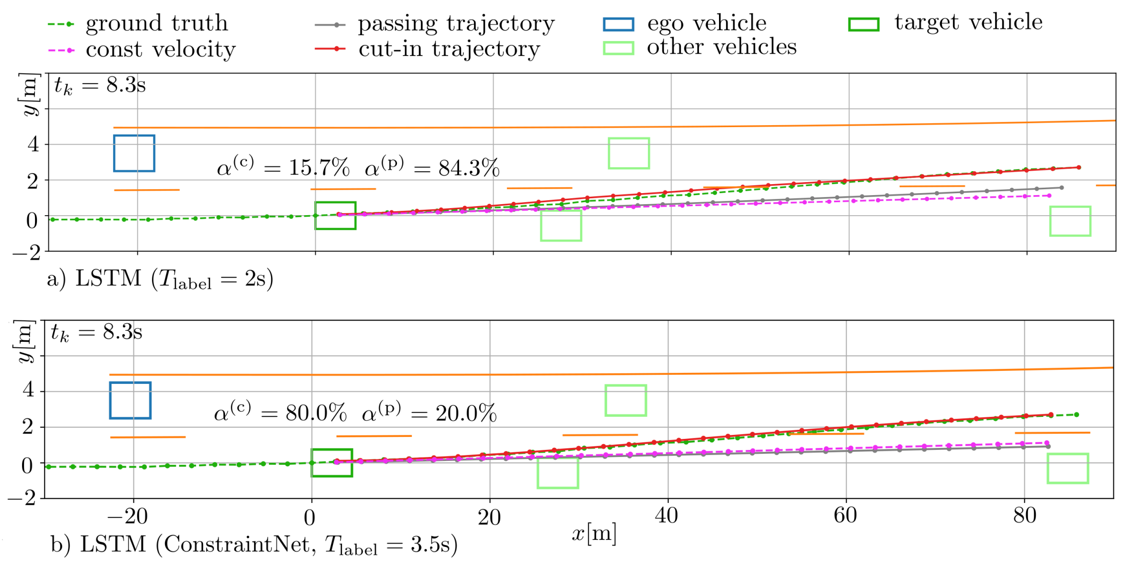


Figure 5.17: For the scene at time $t_k = 8.3$ s in Figure 5.16, the predictions of the LSTM-based model with hard constraints (ConstraintNet, $T_{\text{label}} = 3.5$ s) and the LSTM-based model with a short label duration of $T_{\text{label}} = 2$ s are compared. The depicted scene is 2.2 s before the cut-in time $t_c = 10.5$ s and therefore the model with short label duration predicts correctly a passing maneuver and ConstraintNet correctly a cut-in maneuver. Whereas both models show a consistent maneuver classification, the trajectory accuracy benefits from the larger label duration of $T_{\text{label}} = 3.5$ s. For a detailed explanation of the legend, it is referred to Figure 5.16.

the constant velocity model represents a passing maneuver at time $t_k = 7.9$ s. The second plot is 0.4 s later and ConstraintNet predicts a high probability of $\alpha^{(c)} = 80\%$ for the cut-in maneuver while the target vehicle is still clearly on the neighboring lane. Thus, the threshold of $\tilde{\alpha} = 80\%$ is reached and ConstraintNet detects the cut-in maneuver at this time. The predicted cut-in trajectory shows a non-linear shape and is close to the ground truth. Contrary, the trajectory of the constant velocity model represents still a passing maneuver and the distance to the ground truth is considerably. For the same time $t_k = 8.3$ s and scenario, Figure 5.17 compares the prediction of ConstraintNet with the LSTM-based model with a shorter label duration of $T_{\text{label}} = 2$ s. The LSTM with the short label duration predicts with $\alpha^{(p)} = 84.3\%$ a passing maneuver. This is actually the correct behavior because the time point $t_k = 8.3$ s is 2.2 s before the cut-in time $t_c = 10.5$ s and the model is supposed to predict a cut-in maneuver only $T_{\text{label}} = 2$ s before the cut-in time. Interestingly, the model compensates this fact by predicting the passing trajectory more towards a lane crossing. However, choosing a larger label time T_{label} for a higher time gain comes at the expense of downgraded precision and recall. A good balance is achieved with the label duration of $T_{\text{label}} = 3.5$ s. The lower two plots in Figure 5.16 show the reference time points for measuring the time gain. The center of the target vehicle crosses the lane marking at time $t_k = 10.1$ s and the target vehicle is assigned to the ego lane at time $t_k = 10.5$ s. Consequently, ConstraintNet achieves a time gain of 2.2 s or 1.8 s in the visualized scenario depending on the choice of the reference time point.

Table 5.6 shows the performance of ConstraintNet depending on different input feature sets. The detection threshold for the cut-in prediction is fixed to $\tilde{\alpha} = 0.8$. The best

	Cut-In Prediction $\tilde{\alpha} = 0.8$			Trajectory Prediction			
Model	Precision [%]	Recall [%]	Time Gain [s]	MSE [m^2]	MDE [m]	FMDE [m]	LMDE [m]
ConstraintNet all features	95.4 ± 0.5	**95.5 ± 0.6**	**2.77 ± 0.09** (2.19 ± 0.09)	**4.56 ± 0.19**	**1.56 ± 0.03**	**3.24 ± 0.04**	**0.36 ± 0.01**
without $l_{\text{env,l/r}}$	95.2 ± 0.3	94.5 ± 1.9	2.67 ± 0.07 (2.09 ± 0.07)	4.76 ± 0.14	1.63 ± 0.09	3.33 ± 0.09	0.37 ± 0.01
without w_{tar}	**95.8 ± 0.7**	93.3 ± 3.2	2.61 ± 0.19 (2.03 ± 0.19)	4.80 ± 0.30	1.61 ± 0.03	3.33 ± 0.08	0.37 ± 0.01
without $x_{\text{tar}}, y_{\text{tar}}$	95.2 ± 0.8	94.6 ± 1.1	2.67 ± 0.11 (2.09 ± 0.11)	4.79 ± 0.29	1.61 ± 0.05	3.31 ± 0.08	0.39 ± 0.02
without $x'_{\text{rel, CIPV}}$, $(x/y)'_{\text{rel,tar}}$	90.3 ± 0.8	87.9 ± 1.5	2.26 ± 0.05 (1.68 ± 0.05)	4.98 ± 0.18	1.62 ± 0.03	3.35 ± 0.08	0.38 ± 0.01
without $y_{\text{env,l/r,s}}$	94.1 ± 1.3	73.9 ± 12	2.67 ± 0.13 (2.11 ± 0.13)	4.75 ± 0.18	1.69 ± 0.04	3.59 ± 0.10	0.48 ± 0.03

Table 5.6: The table shows the performance of ConstraintNet on the test set for different input feature sets. The first row shows the results with all input features. In the following rows, the specified input features are excluded to analyze the impact on the performance. Optimal values are written in bold.

results for the maneuver and the trajectory prediction are achieved with the full feature set. Removing either the lane marking type feature $l_{\text{env,l/r}}$, the target vehicle's width w_{tar}, or the target vehicle's past trajectory points $x_{\text{tar}}, y_{\text{tar}}$ results in only slightly downgraded cut-in prediction metrics. Thus, the importance of these features is not as high compared to other features or the feature can be compensated by redundancy. *E.g.* the relative position of the target vehicle $x'_{\text{rel,tar}}, y'_{\text{rel,tar}}$ in combination with ego motion features may compensate the feature for the past trajectory $x_{\text{tar}}, y_{\text{tar}}$ to a certain extent. Excluding the relative position of the target vehicle $x'_{\text{rel,tar}}, y'_{\text{rel,tar}}$ and the distance of the CIPV $x'_{\text{rel,CIPV}}$ from the feature set results in a reduction of 5% in the precision, of 8% in the recall, and 0.5 s in the time gain. This significant drop in performance shows the importance of the spatial relations between the agents for the prediction of a cut-in maneuver. *E.g.* the spatial relations indicate whether a gap in traffic is available for cutting-in. Furthermore, the features for the lane geometry $y_{\text{env,l/r,s}}$ are important. Without theses features, all considered metrics are downgraded. In particular the lateral error of the trajectory prediction part (LMDE) increases significantly from 0.36 ± 0.01m to 0.48 ± 0.03m. In the used Cartesian coordinate system, the features $y_{\text{env,l/r,s}}$ are required to adapt the trajectory prediction to the lane geometry. Furthermore, the lane geometry features are important to distinguish the lateral movement of the target vehicle in curves from the lateral movement of a cut-in maneuver. In the scenario of Figure 5.18, the target vehicle passes the ego vehicle in a curve without cutting-in. Without lane geometry features, the lateral movement of the target vehicle is interpreted as a cut-in maneuver with a confidence of $\alpha^{(c)} = 77.0\%$ at the visualized time step. Furthermore, the predicted cut-in and passing trajectory do not follow the road curvature and are not interpretable. With the lane geometry features, the maneuver is correctly classified as passing with a confidence of $\alpha^{(p)} = 95.8\%$ and the predicted passing and cut-in trajectory are in line with the road geometry as expected.

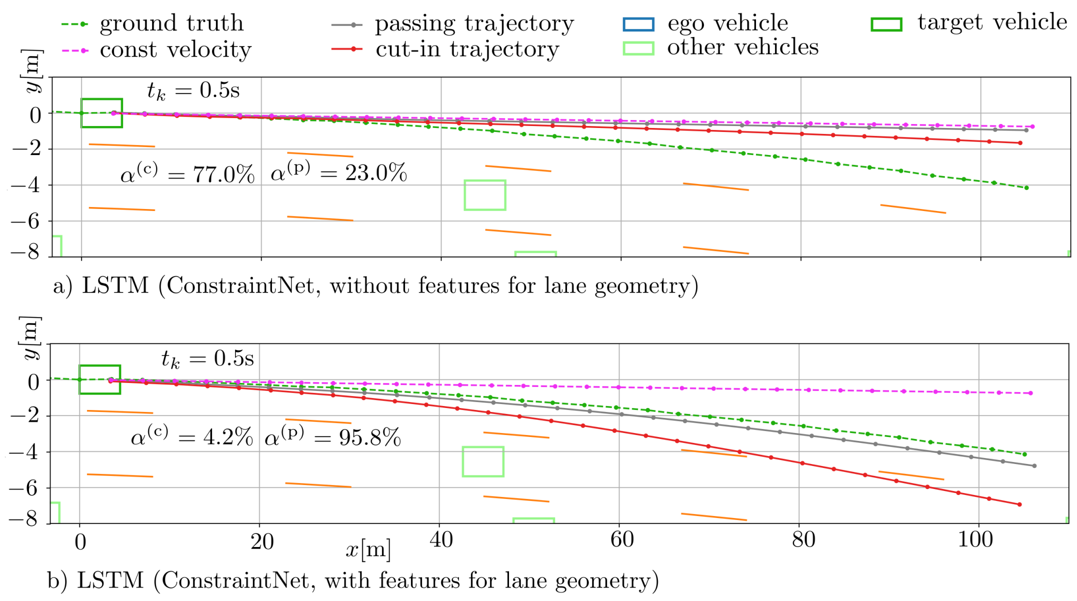
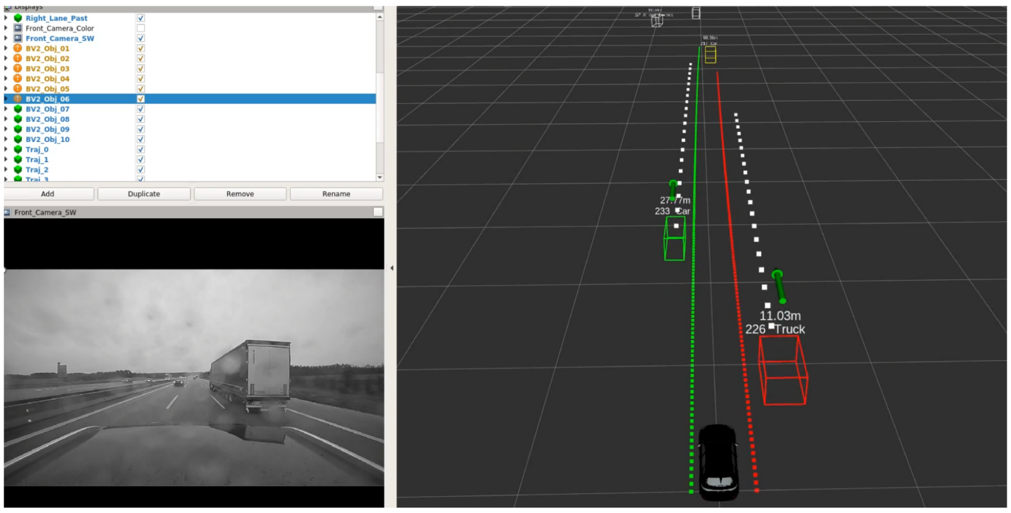


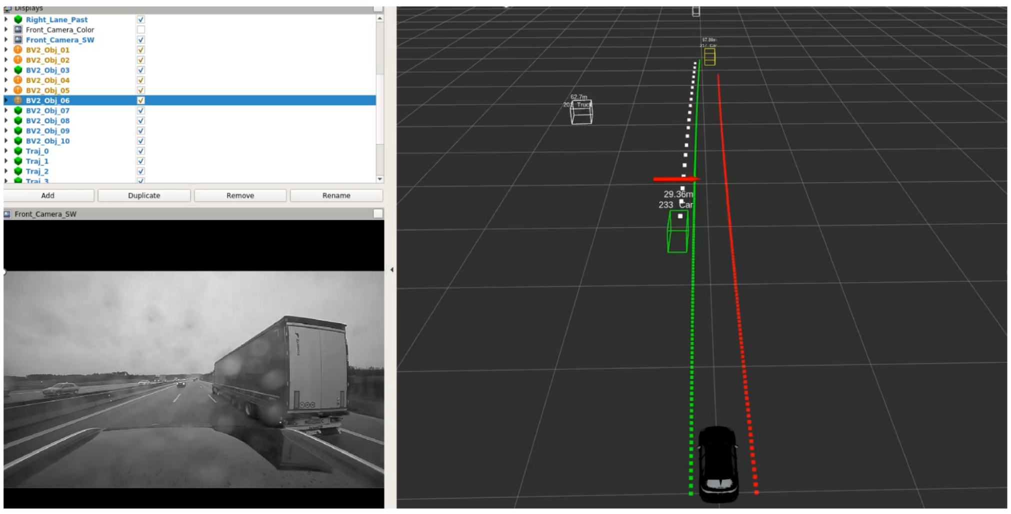
Figure 5.18: The figure shows ConstraintNet's predictions with and without features for the lane geometry. In the visualized scenario, the target vehicle passes the ego vehicle in a curve without cutting-in. The ego vehicle is at position $x = -20$ m and not visualized. Without the lane geometry features, the lateral movement of the target vehicle results in a high confidence of $\alpha^{(c)} = 77.0\%$ for a cut-in maneuver and the predicted trajectories are inconsistent with the road geometry. With lane geometry features, a passing maneuver is predicted correctly with a high confidence of $\alpha^{(p)} = 95.8\%$ and the trajectories are in line with the road geometry. For a detailed explanation of the legend, it is referred to Figure 5.16

For an in-vehicle experience of the cut-in prediction, a live prediction is implemented with the Robot Operating System (ROS) [Quigley et al., 2009]. In a Porsche Cayenne of the third generation as test vehicle, the basic LSTM-based model has been deployed. The model requires access to the output of the ego vehicle's perception and inertial measurement unit. The relevant signals are routed from the vehicle's bus to a computer in the vehicle's trunk and published as messages in Robot Operating System (ROS). Rospy is a Python client library for ROS and helps to deploy the trained model in the test vehicle. For each detected vehicle in the ODD, the input data is preprocessed, the model is applied, and the predictions are visualized live on a display in the test vehicle. Figure 5.19 shows the live visualization, which is implemented with the package RViz[2]. In the depicted highway scenario, two vehicles are in the ODD and one of them is performing a cut-in and the other a passing maneuver. The vehicle on the left-hand side overtakes the ego vehicle and is going to cut-in. The truck on the right-hand side is overtaken by the ego vehicle. The upper visualization shows the model's predictions slightly before the cut-in maneuver of the vehicle on the left lane is detected and the lower screenshot shortly after the first cut-in detection. Arrows on top of the vehicles indicate whether a passing or cut-in maneuver is predicted by the model. Furthermore, a sound is played when the arrow switches from passing to cut-in. This sound is an important feedback for the driver, who can not watch the display continuously. Additionally, the more probable of the predicted cut-in and passing trajectory is visualized with white point markers. In

[2] `http://wiki.ros.org/rviz`, accessed on 10/15/2022



a) Live prediction of LSTM-based model slightly before a cut-in maneuver is predicted



b) Live prediction of LSTM-based model with a prediction of a cut-in maneuver

Figure 5.19: For two scenes of a highway scenario, the predictions of an LSTM-based model that is deployed in a test vehicle are visualized. The visualization is implemented with the ROS-package RViz and includes three panels. The upper left panel allows to configure the displayed elements, the lower left panel shows a video stream, and the big right panel a 3D visualization of the perceived environment and the model's predictions. The detected vehicles are visualized with 3D bounding boxes. Vehicles on the left lane are visualized in green, vehicles on the right lane in red, the CIPV in yellow, and all other vehicles in white. Furthermore, the detected lane markings are depicted with a sequence of point markers with green for the left lane marking and red for the right lane marking. The arrows on top of the target vehicles point ahead if a passing maneuver is predicted and towards the lane of the ego vehicle if a cut-in maneuver is predicted. Furthermore, the more probable of the predicted cut-in and passing trajectory is visualized with white waypoints.

the upper scene of Figure 5.19, the model predicts a passing maneuver for the vehicle and the truck. The trajectory predictions follow the lane geometry and represent lane keeping. In the lower scene of Figure 5.19, the vehicle is still on the neighboring lane but slightly approaches the lane marking. Furthermore, a gap between the ego vehicle and the CIPV is available and a sufficient distance between the target and the ego vehicle is reached. Finally, the LSTM interprets the lateral movement of the target vehicle together with the context knowledge as the start of a cut-in maneuver. The predicted trajectory is oriented towards the lane marking and indicates a lane crossing. Indeed, the target vehicle is cutting-in and changes to the ego lane. The truck on the right-hand side is overtaken without any cut-in detection. Both behaviors are correct and close to how a human would interpret the scenario. In comparison, common ADASs detect a cut-in maneuver late and react only when a significant part of the target vehicle already covers the ego lane. This may result in abrupt decelerations and may even cause hazardous scenarios in the worst-case. Consequently, an early and relaiable cut-in prediction has great potential to improve comfort and safety of DASs. *E.g.* in the vehicle following mode, ACC controls the distance to the CIPV. The cut-in prediction allows to incorporate a change in the CIPV before it will actually happen. Thereby, a predictive control behavior can be achieved. Furthermore, the in-vehicle implementation demonstrates that the computational time is manageable. In future work, it would be interesting to analyze the benefits of the cut-in prediction for the control behavior of DASs and to deploy the model on an electronic control unit as a first step towards a model for series production.

5.5 Conclusion

In this chapter, LSTM-based models have been evaluated for a joint trajectory and cut-in prediction. The models are capable of predicting cut-ins far before a rule-based model and assign interpretable probabilities to the trajectory modes and waypoints. The two modes have been semantically associated with a cut-in and passing maneuver. To improve the lateral separation of the cut-in and passing trajectory, soft and hard output constraints have been modeled and evaluated.

In a supervised student thesis [Orschau, 2021], the LSTM-based models have been compared to transformers [Vaswani et al., 2017]. In the experiment, the transformers have not improved the performance and similar evaluation metrics have been measured. One potential explanation could be that the complexity of the task is moderate and therefore the capacity of LSTMs is sufficient. This argumentation is also consistent with the observation that multilayer LSTMs have not improved over single layer LSTMs. While this chapter has focused on LSTM-based models, transformers are considered especially relevant for possible extensions and enhancements of the behavior prediction.

For training and test, a semi-automatic data pipeline has been built up and a diverse and large data set has been created from recorded measurement files of a test vehicle fleet. The key part of the data pipeline is a retrospective and rule-based extraction of cut-in and passing scenarios. The data pipeline has been shown to be effective and requires a manageable and relatively low manual effort with a labeling tool.

First, the performance of the cut-in and trajectory prediction has been evaluated. The best performing LSTM-based models have achieved a time gain of 2.7 s. This is more than double the baseline's time gain of 1.0 s. The baseline detects cut-ins by thresholding the distance of the target vehicle's center to the lane marking. For a fixed precision, the data-driven approach has also improved the recall over the baseline. Furthermore, the accuracy of the LSTM-based trajectory prediction has outperformed a constant velocity model clearly. For an optimal performance of the trajectory prediction, an additional weighting of the lateral part of the loss has turned out to be important. Moreover, a separate prediction of cut-ins and a separate one for the trajectories have not shown improvements in performance compared to the joint prediction. However, the joint trajectory and cut-in prediction is beneficial for computational costs and improves interpretability by assigning maneuvers to the modes of the bimodal distribution.

For improving the lateral separation of the cut-in and passing trajectory, the impact of soft and hard constraints has been evaluated. The hard constraints have been implemented with ConstraintNet and the soft constraints with an additional loss term. Both constraint types improve mode separation. However, the best overall performance has been achieved with ConstraintNet. Hard constraints have the benefit that they guarantee constraint satisfaction, whereas soft constraints only penalize constraint violation. Furthermore, it has been found that short durations of the cut-in labels downgrade the separation of the trajectory modes. Finally, ConstraintNet has achieved a high trajectory accuracy and has predicted well-separated and interpretable modes.

Contrary to other approaches that use curvilinear coordinates, the trajectory has been predicted in Cartesian coordinates and the road geometry has been encoded in the input features. This has the benefit that the model is not directly downgraded by noisy lane marking detections and is able to reason about the noise in an end-to-end fashion. It has been demonstrated that the model is able to incorporate the road geometry from the input features and predicts trajectories that are aligned with the road geometry.

The model's input has been generated purely from the ego vehicle's sensor measurements. That has made it possible to deploy the model in a test vehicle. Moreover, all considered features have turned out to be important. In particular, features that encode the spatial relations between the agents and the road geometry have shown high influence on the model's performance.

Finally, the results of the joint trajectory and cut-in prediction have shown great potential to improve comfort and safety of planning and control systems. Furthermore, the imposing of constraints has been evaluated successfully to direct the intended behavior.

6 Conclusion and Outlook

This thesis has focused on imposing output constraints on neural networks to exclude unintended behavior in safety-critical environments of driving automation systems. Therefore, a methodological and an experimental part have been distinguished. In the methodological part, ConstraintNet—a novel and efficient neural network architecture with embedded output constraints—has been proposed. In the experimental part, constrained neural networks including ConstraintNet have been evaluated on two important tasks of driving automation systems: 1) learning a control behavior to keep safe distances to vehicles ahead with deep reinforcement learning and 2) learning a joint vehicle trajectory and cut-in prediction for safe motion planning and control.

First, the novel neural network architecture ConstraintNet has been proposed to embed hard output constraints in the architecture. ConstraintNet restricts the output in each forward pass independently to a specifiable geometry and constraint satisfaction is ensured by construction. A specific output constraint is encoded in the form of a precise tensor description and treated as an additional input. Thereby, multiple constraints for the same input are applicable and prior knowledge is incorporated in a flexible way. The more prior knowledge is known, the more specific the constraints are supposed to be chosen and *vice versa*. Contrary to projection-based approaches, ConstraintNet applies an input-dependent parametrization of the constrained output space as the final layer, the so-called constraint guard layer. Thereby, the complete interior of the constrained output space is covered and no additional optimization is required. For constraints in the form of convex polytopes, the constraint guard layer has been constructed by leveraging the vertex representation. Beyond that, ConstraintNet supports a broad class of constraints. This has been shown by suggesting concepts to constrain classification tasks, to combine constraints, and to deal with non-convex and unbounded constraints. Furthermore, the modeling of constraints and the performance of ConstraintNet have been demonstrated on several facial landmark detection tasks. Constraints in the form of bounding boxes, triangles, sectors of a circle, and relative relations between landmarks have been evaluated. ConstraintNet has shown improved performance and reduced runtimes over the projection-based approach. Thus, the results demonstrate the effectiveness of ConstraintNet's design choices. In general, this work aims to reach researchers of different domains and the focus has been set on a formal mathematical formalization. This allows to construct ConstraintNet depending on the specific problem at hand. In particular, the constraints enable the implementation of safety requirements and ConstraintNet contributes to trustfully applying neural networks in safety-critical environments. In future research, it would be interesting to evaluate more complex constraints, to constrain intermediate layers of neural networks [Brosowsky, 2021a], and to learn constraints by the network self [Brosowsky, 2020, 2021c]. The learning of constraints would weaken the safety guarantees but is promising for exploration in unknown environments.

The first application in the experimental part has focused on learning a vehicle following controller with deep reinforcement learning. The objective of a vehicle following controller is keeping a velocity-dependent distance to vehicles ahead under comfort and safety aspects. To avoid rear-end collisions, neural networks with hard output constraints have been proposed for the policy. The constrained sets of outputs are called state-specific safe sets and restrict the control input. The safe sets have been derived from the responsibility-sensitive safety model. However, the abrupt interventions in the responsibility-sensitive safety model have been modified and transformed into a continuous form. Thereby, harsh braking commands are softened and the collision avoidance guarantees remain valid. In the experiments, an unconstrained fully-connected neural network and three constrained neural networks with imposed safe sets have been compared: ConstraintNet, a neural network with clipping as post-processing, and a neural network with a projection layer. All policies have been trained with the TD3 algorithm and evaluated with respect to safety, performance, and training behavior. The results have shown the effectiveness of the safe sets. For all constrained policies, a constant crash rate of zero has been measured. Overall, ConstraintNet and the neural network with clipping as post-processing have achieved slightly improved performances compared to the projection-based approach and the unconstrained policy. Furthermore, ConstraintNet has shown the fastest and most stable training behavior. It is concluded that constrained neural networks and in particular ConstraintNet are promising for learning controllers in safety-critical environments of driving automation systems. For safe automated driving in complex scenarios, the learning of a combined longitudinal and lateral controller with constrained neural networks is of particular interest for future research.

The second application in the experimental part has studied a behavior prediction for safe driving automation systems. An early and reliable behavior prediction of surrounding vehicles is crucial to anticipate hazardous maneuvers and mitigate the risk of late detections. Addressing this, unconstrained and constrained neural networks have been leveraged for a joint vehicle trajectory and cut-in prediction. To capture the time context explicitly, LSTMs and transformers have been considered in an encoder-decoder fashion. However, in experiments the transformers have not improved the performance and achieved similar metrics compared to the LSTMs [Orschau, 2021]. The observations can be explained by the moderate complexity of the task, which may not require models with large capacities. Thus, the focus has been put on LSTM-based models. Nevertheless, transformers are considered especially relevant for possible extensions and enhancements of the model. The output of the decoder has been modeled as a bimodal probability distribution over trajectories and the modes have been semantically associated with a cut-in and passing maneuver. To improve the lateral separation of the cut-in and passing trajectory, hard and soft output constraints have been imposed. For the hard constraints, ConstraintNet has been applied. For the soft constraints, an additional loss term has been added. The proposed models are intended for real-world application on highways. Thus, the input depends only on the ego vehicle's sensors and perception and a large-scale data set has been created from recorded measurement data. Given this data set, the unconstrained LSTM, the LSTM with soft constraints, and the LSTM-based ConstraintNet have been trained. The evaluation has shown significant performance benefits of all LSTM-based models over a physical baseline. All models have achieved similar performance. However, ConstraintNet has improved the overall performance slightly with respect to the cut-in prediction, the trajectory accuracy, and the mode separation. This highlights the smooth integration of output constraints with ConstraintNet in existing architectures to address a specific intended behavior. In general, the decoders bimodal output has shown well-interpretable properties. First, the semantic association of the modes with cut-in and passing maneuvers is similar to how humans anticipate vehicle maneuvers on highways. The association with maneuvers is also important for downstream tasks like trajectory planning. Second, the predicted uncertainties of the trajectories' waypoints are interpretable. The uncertainties are larger in the lateral component compared to the longitudinal direction and increase with larger time horizons. Finally, the joint vehicle trajectory and cut-in prediction anticipates the behavior of other vehicles early, reliable, and in an interpretable way and has shown great potential to improve safety and comfort of planning and control systems. In future research, the integration of the proposed behavior prediction in downstream tasks like adaptive cruise control and trajectory planning is of particular interest to improve safety.

In conclusion, this thesis contributes to responsibly applying neural networks in safety-critical environments by imposing output constraints. Constrained neural networks have been applied on two safety-relevant tasks of driving automation systems. The experiments have shown that constrained neural networks are able to exclude unintended and hazardous behavior explicitly, do not necessarily require computational overhead, and achieve high performances. Finally, constrained neural networks are a promising step towards safe artificial intelligence and the dream of safe self-driving vehicles.

List of Figures

Figure 1.1 Research fields addressing the black box character of NNs. 2
Figure 1.2 The six levels of driving automation according to SAE-J3016 [2021]. 4
Figure 1.3 Sense-plan-act model and processing chain of DASs. 5
Figure 1.4 Overview of the chapters in this thesis. 7

Figure 2.1 The minimization of hazardous scenarios is the goal of SOTIF activities. 14
Figure 2.2 The process of the SOTIF standard. 15
Figure 2.3 Imposing constraints on the control input for safe driving policies. . 19
Figure 2.4 Illustration of an artificial neuron and a neural network. 23
Figure 2.5 A convolutional layer in 2D. 24
Figure 2.6 Visualization of an RNN. 26
Figure 2.7 Visualization of different RNN topologies. 28
Figure 2.8 Illustration of the backpropagation algorithm. 33
Figure 2.9 Interaction of an agent with the environment in RL. 36
Figure 2.10 Graphical model for an MDP. 38
Figure 2.11 Solving MDPs approximately with general policy iteration. 41
Figure 2.12 Illustration of actor-critic methods. 46

Figure 3.1 Data-driven, rule-based, and constrained data-driven models. 50
Figure 3.2 Global and sample-specific output constraints for NNs. 51
Figure 3.3 Output range of the softmax function. 52
Figure 3.4 Components of ConstraintNet. 53
Figure 3.5 Kinematic layer for predicting only kinematically feasible trajectories. 56
Figure 3.6 Projection of a point on a constrained region. 57
Figure 3.7 Illustration of the barrier method for a linear program. 59
Figure 3.8 ConstraintNet with triangle constraints and a CNN as backbone. . . . 62
Figure 3.9 Constraint guard layers for several geometric shapes. 64
Figure 3.10 ConstraintNet for a constraint in the form of a non-convex polytope. . 67
Figure 3.11 Four different output constraints for facial landmark detection. 69
Figure 3.12 Bounding box and relative constraints for facial landmark detection. . 70
Figure 3.13 Learning curves of ConstraintNet, the projection-based approach and ResNet50. 73
Figure 3.14 Streamlit app for facial landmark detection with constrained NNs. . . 76
Figure 3.15 Facial landmark detection of ConstraintNet, the projection-base approach, and ResNet50 without constraints on ambiguous images with two faces. 77

Figure 3.16 Facial landmark detection with ConstraintNet on a rotated portrait. . 78

Figure 4.1 Environment of a vehicle following controller and visualization of learning the control law with an actor-critic model. 83
Figure 4.2 Overview about the evaluated NNs for the vehicle following policy. . 84
Figure 4.3 Proper longitudinal response according to the RSS model in a vehicle following scenario [Shalev-Shwartz et al., 2017]. 91
Figure 4.4 Upper bound of the demanded acceleration for distances above the minimal safe distance. 93
Figure 4.5 Graphical user interface of the simulator for vehicle following control. 101
Figure 4.6 Reward function for comfortable and accurate vehicle following behavior [Desjardins and Chaib-draa, 2011]. 102
Figure 4.7 Average episode return over training time for ConstraintNet, the projection-based, the clipped, and the unconstrained policy. 105
Figure 4.8 Crash rate over training time of the original unconstrained policy. . . 106

Figure 5.1 The model output are parameters of a bimodal distribution over the target vehicle's trajectory. 111
Figure 5.2 Categorization of models for behavior prediction. 114
Figure 5.3 Visualization of the pipeline for the creation of a large and diverse data set with an equal number of cut-in and passing scenarios. 117
Figure 5.4 Automatic detection of cut-in and passing scenarios with regular expressions. 118
Figure 5.5 Screenshot of the labeling tool. 120
Figure 5.6 Distributions over the tracking time, the distance, and the relative velocity of the cutting-in vehicles in the data set. 122
Figure 5.7 Visualization of the input features for the behavior prediction. 124
Figure 5.8 Visualization of an LSTM cell. 126
Figure 5.9 Illustration of the LSTM-based encoder-decoder architecture. 127
Figure 5.10 Hard output constraints for enforcing a lateral separation of the cut-in and passing trajectories. 129
Figure 5.11 The predicted cut-in probability over time of a true positive. 132
Figure 5.12 Visualization of the metrics for the evaluation of the trajectory prediction. 133
Figure 5.13 Overview of the preprocessing steps for the behavior prediction. . . . 134
Figure 5.14 Computation of the target vehicle's trajectory with odometry and measurements of the relative position. 136
Figure 5.15 Distributions over the time gains of ConstraintNet and the baseline. . 141
Figure 5.16 Comparison of the trajectory prediction between ConstraintNet and the constant velocity model. 143
Figure 5.17 Comparison of the behavior prediction for two different label durations. 144
Figure 5.18 Comparison of the behavior prediction in a curve between ConstraintNet with and ConstraintNet without features for the lane geometry. . 146
Figure 5.19 Screenshot of the in-vehicle prediction with ROS. 147

List of Tables

Table 3.1 Runtimes for the training and the evaluation of ConstraintNet, the projection-based approach, and ResNet50 for facial landmark detection tasks. 74
Table 3.2 MSE metrics for facial landmark detection tasks on the test set for ConstraintNet, the projection-based approach, and ResNet50. 75

Table 4.1 Parameters of the time gap policy and the simulator. 89
Table 4.2 Parameters for the RSS-based safe sets. 92
Table 4.3 Parameters of the TD3 algorithm. 103
Table 4.4 Results of the different policies and of the ConstraintNet-based policy depending on exploration noise, network size, and reward weights. . 107

Table 5.1 Statistics of the data set with cut-in and passing scenarios. 121
Table 5.2 Input features of the behavior prediction. 123
Table 5.3 Hyperparameters of the behavior prediction. 138
Table 5.4 Results of the cut-in prediction. 140
Table 5.5 Results of the trajectory prediction. 142
Table 5.6 Analysis of the feature importance. 145

Publications, Patents, and Supervised Theses

Conference Contributions

Brosowsky, M., Keck, F., Dünkel, O., and Zöllner, M. (2021a). Sample-Specific Output Constraints for Neural Networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35:6812–6821.

Brosowsky, M., Keck, F., Ketterer, J., Isele, S., Slieter, D., and Zöllner, M. (2021b). Safe Deep Reinforcement Learning for Adaptive Cruise Control by Imposing State-Specific Safe Sets. *Proceedings of the 32nd IEEE Intelligent Vehicles Symposium*, pages 488–495.

Brosowsky, M., Orschau, P., Dünkel, O., Elspas, P., Slieter, D., and Zöllner, M. (2021c). Joint Vehicle Trajectory and Cut-In Prediction on Highways using Output Constrained Neural Networks. *Proceedings of the IEEE Symposium Series on Computational Intelligence*.

Elspas, P., Lindner, J., Brosowsky, M., Bach, J., and Sax, E. (2022). Towards a Scenario Database from Recorded Driving Data with Regular Expressions for Scenario Detection. *Proceedings of the 8th International Conference on Vehicle Technology and Intelligent Transport Systems - VEHITS*, pages 400–409.

Isele, S., Klein, F., Brosowsky, M., and Zöllner, M. (2021). Learning Semantics on Radar Point-Clouds. *Proceedings of the 32nd IEEE Intelligent Vehicles Symposium*, pages 810–817.

Patents

Brosowsky, M. (2019). DE10 2019 119 739 A1: Verfahren und System zur Erzeugung von sicherheitskritischen Ausgabewerten mittels einer Datenanalyseeinrichtung. *Deutsches Patent- und Markenamt*.

Brosowsky, M. (2020). DE10 2020 127 051 A1: Verfahren zur Bestimmung von sicherheitskritischen Ausgabewerten mittels einer Datenanalyseeinrichtung. *Deutsches Patent- und Markenamt*.

Brosowsky, M. (2021a). DE10 2021 100 765 A1: Verfahren, System und Computerprogrammprodukt zur Bestimmung von sicherheitskritischen Ausgabewerten. *Deutsches Patent- und Markenamt*.

Brosowsky, M. (2021b). US 20210027150 A1: Method and System for Generating Safety-Critical Output Values of An Entity. *United States Patent and Trademark Office*.

Brosowsky, M. (2021c). US 20220114416 A1: Method for Determining Safety-Critical Output Values by Way of a Data Analysis Device for a Technical Entity. *United States Patent and Trademark Office*.

Supervised Theses

Dünkel, O. (2019). Vehicle Trajectory Prediction Using a Neural Network With Uncertainty Estimation in the Context of a Cut-In Detection. *Bachelor Thesis, Institut für Technik der Informationsverarbeitung, Karlsruhe Institut für Technologie*.

Keck, F. (2020). Learning a Safe Follow-Up Control Using Reinforcement Learning and Implementation of a Simulation. *Bachelor Thesis, Institut für Technik der Informationsverarbeitung, Karlsruhe Institut für Technologie*.

Orschau, P. (2021). Multi-Modal Trajectory Prediction using Transformer Networks in the Context of Predicting Cut-In Scenarios on Highways. *Bachelor Thesis,, Institute for Automotive Engineering, RWTH Aachen*.

Puzzo, F. (2020). Implementation of a Follow Control with Neural Networks and Evaluation of an Integrated Safety Concept for Neural Networks. *Master Thesis, Institut für Verbrennungsmotoren und Kraftfahrwesen, Universität Stuttgart*.

Bibliography

Achiam, J., Held, D., Tamar, A., and Abbeel, P. (2017). Constrained policy optimization. *International Conference on Machine Learning*, 70:22–31.

Agrawal, A., Amos, B., Barratt, S., Boyd, S., Diamond, S., and Kolter, J. Z. (2019). Differentiable convex optimization layers. *Advances in Neural Information Processing Systems*, 32:9562–9574.

Akiba, T., Sano, S., Yanase, T., Ohta, T., and Koyama, M. (2019). Optuna: A next-generation hyperparameter optimization framework. *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2623–2631.

Alahi, A., Goel, K., Ramanathan, V., Robicquet, A., Fei-Fei, L., and Savarese, S. (2016). Social lstm: Human trajectory prediction in crowded spaces. *IEEE Conference on Computer Vision and Pattern Recognition*, pages 961–971.

Altche, F. and Fortelle, A. D. L. (2018). An lstm network for highway trajectory prediction. *IEEE International Conference on Intelligent Transportation Systems*, pages 353–359.

Altman, E. (1999). *Constrained Markov Decision Processes*. CRC Press.

Amersbach, C. and Winner, H. (2017). Functional decomposition: An approach to reduce the approval effort for highly automated driving. *8. Tagung Fahrerassistenz*.

Amos, B. and Kolter, J. Z. (2017). Optnet: Differentiable optimization as a layer in neural networks. *International Conference on Machine Learning*, 70:136–145.

Bansal, M., Krizhevsky, A., and Ogale, A. (2019). Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst. *Robotics: Science and Systems*.

Barratt, S. (2018). On the differentiability of the solution to convex optimization problems. *arXiv preprint arXiv:1804.05098*.

Barthelmes, L., Wilkes, G., Kagerbauer, M., and Vortisch, P. (2022). Ein On-Demand- und Level 4-Kleinbus auf dem Testfeld Autonomes Fahren BW – Erkenntnisse aus der begleitenden Haushaltsbefragung zu EVA-Shuttle. *Journal für Mobilität und Verkehr*, pages 36–46.

Bauer, K. L. and Gauterin, F. (2016). A two-layer approach for predictive optimal cruise control. *SAE Technical Papers*, 2016-April.

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5:157–166.

Bergstra, J., Bardenet, R., Bengio, Y., and Kégl, B. (2011). Algorithms for hyper-parameter optimization. *Advances in Neural Information Processing Systems*, pages 2546–2554.

BGBl (2017). *(Bundesgesetzblatt I) Gesetz zur Änderung des Straßenverkehrsgesetzes - Gesetz zum automatisierten Fahren*.

BGBl (2021). *(Bundesgesetzblatt I) Gesetz zur Änderung des Straßenverkehrsgesetzes und des Pflichtversicherungsgesetzes - Gesetz zum autonomen Fahren*.

Bishop, C. M. (1994). *Mixture Density Networks*. Aston University.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*, volume 4. Springer New York.

Bojarski, M., Testa, D. D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., Zhang, J., Zhang, X., Zhao, J., and Zieba, K. (2016). End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*.

Boyd, S. and Vandenberghe, L. (2009). *Convex Optimization*. Cambridge University Press, 7 edition.

Bradford, E. and Imsland, L. (2018). Stochastic nonlinear model predictive control using gaussian processes. *European Control Conference*, pages 1027–1034.

Canale, M. and Malan, S. (2003). Robust design of pid based acc s and g systems. *IFAC Proceedings Volumes*, 36:333–338.

Casas, S., Gulino, C., Suo, S., Luo, K., Liao, R., and Urtasun, R. (2020). Implicit latent variable model for scene-consistent motion forecasting. *European Conference on Computer Vision*.

Chamraz, S. and Balogh, R. (2018). Two approaches to the adaptive cruise control (acc) design. *International Conference on Cybernetics and Informatics*.

Cho, K., Merriënboer, B. V., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *Empirical Methods in Natural Language Processing*, pages 1724–1734.

Cootes, T. F., Edwards, G. J., and Taylor, C. J. (2001). Active appearance models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23:681–685.

Cristinacce, D. and Cootes, T. (2006). Feature detection and tracking with constrained local models. *British Machine Vision Conference*, pages 929–938.

Cui, H., Nguyen, T., Chou, F.-C., Lin, T.-H., Schneider, J., Bradley, D., and Djuric, N. (2020). Deep kinematic models for kinematically feasible vehicle trajectory predictions. *IEEE International Conference on Robotics and Automation*, pages 10563–10569.

Cui, H., Radosavljevic, V., Chou, F.-C., Lin, T.-H., Nguyen, T., Huang, T.-K., Schneider, J., and Djuric, N. (2019). Multimodal trajectory predictions for autonomous driving using deep convolutional networks. *IEEE International Conference on Robotics and Automation*, pages 2090–2096.

Dalal, G., Dvijotham, K., Vecerik, M., Hester, T., Paduraru, C., and Tassa, Y. (2018). Safe exploration in continuous action spaces. *arXiv preprint arXiv:1801.08757*.

Darbha, S. and Rajagopal, K. R. (1999). Intelligent cruise control systems and traffic flow stability. *Transportation Research Part C: Emerging Technologies*, 7:329–352.

Deo, N. and Trivedi, M. M. (2018a). Convolutional social pooling for vehicle trajectory prediction. *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*.

Deo, N. and Trivedi, M. M. (2018b). Multi-modal trajectory prediction of surrounding vehicles with maneuver based lstms. *IEEE Intelligent Vehicles Symposium, Proceedings*, pages 1179–1184.

Desjardins, C. and Chaib-draa, B. (2011). Cooperative adaptive cruise control: A reinforcement learning approach. *IEEE Transactions on Intelligent Transportation Systems*, 12:1248–1260.

Destatis (2022). *Verkehrsunfälle - Fachserie 8 Reihe 7*. Statistisches Bundesamt.

Djuric, N., Radosavljevic, V., Cui, H., Nguyen, T., Chou, F. C., Lin, T. H., Singh, N., and Schneider, J. (2020). Uncertainty-aware short-term motion prediction of traffic actors for autonomous driving. *IEEE Conference on Applications of Computer Vision*, pages 2084–2093.

Dontchev, A. L. and Rockafellar, R. T. (2009). *Implicit Functions and Solution Mappings*. Springer New York.

Duvenaud, D. K., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A., and Adams, R. P. (2015). Convolutional networks on graphs for learning molecular fingerprints. *Advances in Neural Information Processing Systems*, 28:2224–2232.

Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, 14:179–211.

Elspas, P., Langner, J., Aydinbas, M., Bach, J., and Sax, E. (2020). Leveraging regular expressions for flexible scenario detection in recorded driving data. *IEEE International Symposium on Systems Engineering, Proceedings*.

Engel, J. M. and Babuska, R. (2014). On-line reinforcement learning for nonlinear motion control: Quadratic and non-quadratic reward functions. *IFAC Proceedings Volumes*, 47:7043–7048.

EU2019/2144 (2019). General safety regulation. *The European Parliament and the Council*.

European-Commission (2021). *Proposal for a Regulation Laying Down Harmonised Rules on AI.*

Fujimoto, S., Hoof, H. V., and Meger, D. (2018). Addressing function approximation error in actor-critic methods. *International Conference on Machine Learning*, 80:1587–1596.

Gal, Y. (2016). *Uncertainty in Deep Learning*. University of Cambridge (PhD Thesis).

Gal, Y. and Ghahramani, Z. (2016). Dropout as a bayesian approximation: Representing model uncertainty in deep learning. *International Conference on Machine Learning*, 48:1050–1059.

Gao, J., Sun, C., Zhao, H., Shen, Y., Anguelov, D., Li, C., and Schmid, C. (2020). Vectornet: Encoding hd maps and agent dynamics from vectorized representation. *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11522–11530.

García, J., Fern, and Fernández, O. (2015). A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16:1437–1480.

Gernant, E., Vieth, B., and Roßmann, B. (2022). The autonomous logistics hub of the future. *White Paper*.

Gilpin, L. H., Bau, D., Yuan, B. Z., Bajwa, A., Specter, M., and Kagal, L. (2018). Explaining explanations: An overview of interpretability of machine learning. *IEEE International Conference on Data Science and Advanced Analytics*, pages 80–89.

Giuliari, F., Hasan, I., Cristani, M., and Galasso, F. (2021). Transformer networks for trajectory forecasting. *International Conference on Pattern Recognition*, pages 10335–10342. Code is available.

Grigorescu, S., Trasnea, B., Cocias, T., and Macesanu, G. (2020). A survey of deep learning techniques for autonomous driving. *Journal of Field Robotics*, 37:362–386.

Gros, S., Zanon, M., and Bemporad, A. (2020). Safe reinforcement learning via projection on a safe set: How to achieve optimality? *IFAC PapersOnLine*, 53:8076–8081.

Gutjahr, B. (2019). *Recheneffiziente Trajektorienoptimierung für automatisierte Fahreingriffe*. KIT Scientific Publishing (PhD Thesis).

Guzman-Rivera, A., Batra, D., Tech, V., and Kohli, P. (2012). Multiple choice learning: Learning to produce multiple structured outputs. *Advances in Neural Information Processing Systems*, 25:1799–1807.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. *IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778.

Heger, M. (1994). Consideration of risk in reinforcement learning. *International Conference on Machine Learning*, pages 105–111.

Henaff, M., Bruna, J., and LeCun, Y. (2015). Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*.

Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366.

Howard, R. A. and Matheson, J. E. (1972). Risk-sensitive markov decision processes. *Management Science*, 18:356–369.

Huang, X., Kwiatkowska, M., Wang, S., and Wu, M. (2017). Safety verification of deep neural networks. *International Conference on Computer Aided Verification*, pages 3–29.

ISO15622 (2018). Intelligent transport systems — adaptive cruise control systems — performance requirements and test procedures. *The International Organization for Standardization*.

ISO21448 (2022). Road vehicles — safety of the intended functionality. *The International Organization for Standardization*.

ISO24089 (2023). Road vehicles — software update engineering. *The International Organization for Standardization*.

ISO26262 (2018). Road vehicles — functional safety. *The International Organization for Standardization*.

ISO/SAE21434 (2021). Road vehicles — cybersecurity engineering. *The International Organization for Standardization*.

Karpatne, A., Watkins, W., Read, J., and Kumar, V. (2017). Physics-guided neural networks (pgnn): An application in lake temperature modeling. *arXiv preprint arXiv:1710.11431*.

Katz, G., Barrett, C., Dill, D. L., Julian, K., and Kochenderfer, M. J. (2017). Reluplex: An efficient smt solver for verifying deep neural networks. *International Conference on Computer Aided Verification*, pages 97–117.

KI-Absicherung (2022). Safe ai for automated driving. *https://www.ki-absicherung-projekt.de (accessed: 10/12/2022)*.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *International Conference on Learning Representations*.

Kingma, D. P. and Welling, M. (2014). Auto-encoding variational bayes. *International Conference on Learning Representations*.

Kipf, T. N. and Welling, M. (2017). Semi-supervised classification with graph convolutional networks. *International Conference on Learning Representations*.

Klingner, M. and Fingscheidt, T. (2022). Improved dnn robustness by multi-task training with an auxiliary self-supervised task. *Deep Neural Networks and Data for Automated Driving*, pages 149–170.

Koller, B. and Matawa, R. (2020). Automated driving requires international regulations. *White Paper (TÜV Süd)*.

Lathuilière, S., Mesejo, P., Alameda-Pineda, X., and Horaud, R. (2020). A comprehensive analysis of deep regression. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 2065–2081.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86:2278–2323.

Lezcano-Casado, M. and Martínez-Rubio, D. (2019). Cheap orthogonal constraints in neural networks: A simple parametrization of the orthogonal and unitary group. *International Conference on Machine Learning*, 97:3794–3803.

Li, G. and Gorges, D. (2019). Ecological adaptive cruise control and energy management strategy for hybrid electric vehicles based on heuristic dynamic programming. *IEEE Transactions on Intelligent Transportation Systems*, 20:3526–3535.

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. (2017a). Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18:6765–6816.

Li, S. E., Zheng, Y., Li, K., Wu, Y., Hedrick, J. K., Gao, F., and Zhang, H. (2017b). Dynamical modeling and distributed control of connected and automated vehicles: Challenges and opportunities. *IEEE Intelligent Transportation Systems Magazine*, 9:46–58.

Li, T. and Srikumar, V. (2019). Augmenting neural networks with first-order logic. *ACL Annual Meeting of the Association for Computational Linguistics*, pages 292–302.

Liang, M., Yang, B., Hu, R., Chen, Y., Liao, R., Feng, S., and Urtasun, R. (2020). Learning lane graph representations for motion forecasting. *European Conference on Computer Vision*, pages 541–556.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2016). Continuous control with deep reinforcement learning. *International Conference on Learning Representations*.

Lin, T. Y., Goyal, P., Girshick, R., He, K., and Dollar, P. (2017). Focal loss for dense object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42:318–327.

Lin, Y., McPhee, J., and Azad, N. L. (2021). Comparison of deep reinforcement learning and model predictive control for adaptive cruise control. *IEEE Transactions on Intelligent Vehicles*, 6:221–231.

Liu, L. and Sukhatme, G. S. (2018). A solution to time-varying markov decision processes. *IEEE Robotics and Automation Letters*, 3:1631–1638.

Liu, Z., Luo, P., Wang, X., and Tang, X. (2015). Deep learning face attributes in the wild. *IEEE International Conference on Computer Vision*, pages 3730–3738.

Madry, A. (2018). Towards deep learning models resistant to adversarial attacks. *International Conference on Learning Representations*.

Maire, F. and Bulitko, V. (2005). Apprenticeship learning for initial value functions in reinforcement learning. *IJCAI Workshop on Planning and Learning in A Priori Unknown or Dynamic Domains*.

Makansi, O., Ilg, E., Cicek, O., and Brox, T. (2019). Overcoming limitations of mixture density networks: A sampling and fitting framework for multimodal future prediction. *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7137–7146.

Mattingley, J. and Boyd, S. (2012). Cvxgen: A code generator for embedded convex optimization. *Optimization and Engineering*, 13:1–27.

McLachlan, G. J. and Basford, K. E. (1988). *Mixture Models: Inference and Applications to Clustering*. Marcel Dekker, New York.

Mertikopoulos, P., Hallak, N., Kavis, A., and Cevher, V. (2020). On the almost sure convergence of stochastic gradient descent in non-convex problems. *Advances in Neural Information Processing Systems*, 33:1117–1128.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518:529–533.

Márquez-Neila, P., Salzmann, M., and Fua, P. (2017). Imposing hard constraints on deep networks: Promises and limitations. *arXiv preprint arXiv:1706.02025*.

Nistér, D., Lee, H.-L., Ng, J., and Wang, Y. (2019). The safety force field. *White Paper (NVIDIA)*.

Opitz, D. W. and Shavlik, J. W. (1995). Generating accurate and diverse members of a neural-network ensemble. *Advances in Neural Information Processing Systems*, 8:535–541.

Pan, J., Sun, H., Xu, K., Jiang, Y., Xiao, X., Hu, J., and Miao, J. (2020). Lane attention: Predicting vehicles' moving trajectories by learning their attention over lanes. *IEEE International Conference on Intelligent Robots and Systems*, pages 7949–7956.

Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural networks. *International Conference on Machine Learning*, 28:1310–1318.

PEGASUS (2019). Project for the establishment of generally accepted quality criteria, tools and methods as well as scenarios and situations. *https://www.pegasusprojekt.de (accessed: 10/10/2022)*.

Pham, T. H., Magistris, G. D., and Tachibana, R. (2018). Optlayer - practical constrained optimization for deep reinforcement learning in the real world. *IEEE International Conference on Robotics and Automation*, pages 6236–6243.

Phan-Minh, T., Grigore, E. C., Boulton, F. A., Beijbom, O., and Wolff, E. M. (2020). Covernet: Multimodal behavior prediction using trajectory sets. *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14062–14071.

Ploeg, J., Scheepers, B. T., Nunen, E. V., Wouw, N. V. D., and Nijmeijer, H. (2011). Design and experimental evaluation of cooperative adaptive cruise control. *IEEE Conference on Intelligent Transportation Systems*, pages 260–265.

Puterman, M. L. (1994). *Markov decision processes: Discrete stochastic dynamic programming*. John Wiley and Sons.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y. (2009). Ros: an open-source robot operating system. *IEEE International Conference on Robotics and Automation Workshop on Open Source Software*.

Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You only look once: Unified, real-time object detection. *IEEE Conference on Computer Vision and Pattern Recognition*, pages 779–788.

Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 234–241.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408.

Rout, M. K., Sain, D., Swain, S. K., and Mishra, S. K. (2016). Pid controller design for cruise control system using genetic algorithm. *International Conference on Electrical, Electronics, and Optimization Techniques*, pages 4170–4174.

Rudin, C. (2019). Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1:206–215.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323:533–536.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2015). Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115:211–252.

SAE-J3016 (2021). Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles. *SAE International*.

SaFAD (2019). Safety first for automated driving. *White Paper*.

Sakhdari, B. and Azad, N. L. (2018). Adaptive tube-based nonlinear mpc for economic autonomous cruise control of plug-in hybrid electric vehicles. *IEEE Transactions on Vehicular Technology*, 67:11390–11401.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. (2009). The graph neural network model. *IEEE Transactions on Neural Networks*, 20:61–80.

Schlechtriemen, J., Wedel, A., Hillenbrand, J., Breuel, G., and Kuhnert, K. D. (2014). A lane change detection approach using feature ranking with maximized predictive power. *IEEE Intelligent Vehicles Symposium, Proceedings*, pages 108–114.

Schmidt, J., Jordan, J., Gritschneder, F., and Dietmayer, K. (2022). Crat-pred: Vehicle trajectory prediction with crystal graph convolutional neural networks and multi-head self-attention. *IEEE International Conference on Robotics and Automation*, pages 7799–7805.

Schmüdderich, J., Rebhan, S., Weisswange, T., Kleinehagenbrock, M., Kastner, R., Nishigaki, M., Kusuhara, S., Kamiya, H., Mori, N., and Ishida, S. (2015). A novel approach to driver behavior prediction using scene context and physical evidence for intelligent adaptive cruise control (i-acc). *Future Active Safety Technology Towards zero traffic accidents*.

Schuster, M. and Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45:2673–2681.

Shalev-Shwartz, S., Shammah, S., and Shashua, A. (2017). On a formal model of safe and scalable self-driving cars. *arXiv preprint arXiv:1708.06374*.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Driessche, G. V. D., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). Deterministic policy gradient algorithms. *International Conference on Machine Learning*, 32:387–395.

Simonyan, K., Vedaldi, A., and Zisserman, A. (2014). Deep inside convolutional networks: Visualising image classification models and saliency maps. *International Conference on Learning Representations Workshop*.

Smith, S., Elsen, E., and De, S. (2020). On the generalization benefit of noise in stochastic gradient descent. *International Conference on Machine Learning*, 119:9058–9067.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958.

Stage, H., Ries, L., Langner, J., Otten, S., Sax, E., Stage, H., Ries, L., Langner, J., Otten, S., Sax, E., Ries, L., Langner, J., Otten, S., and Sax, E. (2022). Analysis and comparison of datasets by leveraging data distributions in latent spaces. *Deep Neural Networks and Data for Automated Driving*, pages 107–126.

Stegmüller, S., Werner, M., Kern, M., Birzle-Harder, B., Götz, K., and Stein, M. (2019). *Akzeptanzstudie "ROBOCAB"*. Fraunhofer IAO.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. *Advances in Neural Information Processing Systems*, 27:3104–3112.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning, Second Edition: An Introduction - Complete Draft*. MIT Press.

Sutton, R. S., McAllester, D., Singh, S., and Mansour, Y. (1999). Policy gradient methods for reinforcement learning with function approximation. *Advances in Neural Information Processing Systems*, 12:1057–1063.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. (2013). Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*.

Tapani, A. (2012). Vehicle trajectory effects of adaptive cruise control. *Journal of Intelligent Transportation Systems*, 16:36–44.

Tieleman, T. and Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4:26–31.

UN-R157 (2021). Un regulation no. 157 — uniform provisions concerning the approval of vehicles with regard to automated lane keeping systems. *United Nations Economic Commission for Europe*.

UN-R79 (2018). Un regulation no. 79 — uniform provisions concerning the approval of vehicles with regard to steering equipment. *United Nations Economic Commission for Europe*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Łukasz Kaiser, and Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, 30:5998–6008.

Wachenfeld, W. and Winner, H. (2015). *Autonomes Fahren - Kapitel: Die Freigabe des autonomen Fahrens*. Springer Vieweg, Berlin, Heidelberg.

Weidl, G., Madsen, A. L., Wang, S., Kasper, D., and Karlsen, M. (2018). Early and accurate recognition of highway traffic maneuvers considering real world application: A novel framework using bayesian networks. *IEEE Intelligent Transportation Systems Magazine*, 10:146–158.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256.

Wirthmüller, F., Klimke, M., Schlechtriemen, J., Hipp, J., and Reichert, M. (2021). Predicting the time until a vehicle changes the lane using lstm-based recurrent neural networks. *IEEE Robotics and Automation Letters*, 6:2357–2364.

Wirthmüller, F., Schlechtriemen, J., Hipp, J., and Reichert, M. (2020). Teaching vehicles to anticipate: A systematic study on probabilistic behavior prediction using large data sets. *IEEE Transactions on Intelligent Transportation Systems*, 22:7129–7144.

Wissing, C., Nattermann, T., Glander, K. H., Hass, C., and Bertram, T. (2017). Lane change prediction by combining movement and situation based probabilities. *IFAC PapersOnLine*, 50:3554–3559.

Zadeh, A., Baltrušaitis, T., and Morency, L.-P. (2017). Convolutional experts constrained local model for 3d facial landmark detection. *IEEE International Conference on Computer Vision Workshops*, pages 2519–2528.

Zeiler, M. D. and Fergus, R. (2014). Visualizing and understanding convolutional networks. *European Conference on Computer Vision*, pages 818–833.