Fakultät für Elektrotechnik und Informationstechnik
Technische Universität München

Towards a cognitive automotive environment model: a novel approach based on distributed representations and spiking neural networks

Florian Mirus

Vollständiger Abdruck der von der Fakultät für Elektrotechnik und Informationstechnik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktor-Ingenieurs (Dr.-Ing.)

genehmigten Dissertation.

Vorsitzender:
Prof. Dr.-Ing. Werner Hemmert

Prüfende der Dissertation:
1. Prof. Dr. Jörg Conradt

2. Prof. Dr. Andreas Herkersdorf

Die Dissertation wurde am 01. Oktober 2019 bei der Technischen Universität München eingereicht und durch die Fakultät für Elektrotechnik und Informationstechnik am 17. November 2020 angenommen.

Abstract

The race to autonomous driving is currently one of the main forces for pushing research forward in the automotive domain. One major reason for this development in recent years is the rapid progress of Artificial Intelligence, especially the success of deep learning, which has shown remarkable results in tasks essential for autonomous driving. The focus of the young and emerging research field *neuromorphic engineering* is on biologically inspired computing systems and algorithms, aiming to close the gap in performance and efficiency between biological and artificial computing systems. Prototypes of neuromorphic computing hardware, although not technologically mature yet, show promise to be a useful, energy-efficient addition in future automotive applications. However, neuromorphic computing approaches are just beginning to draw attention in the automotive domain due to these novel spiking-neuron architectures encapsulating a drastically different computing paradigm and therefore call for alternative algorithmic approaches and new programming substrates.

In this thesis, we present a first step towards a cognitive environment model for automotive applications using distributed representations and a spiking neuron substrate. We investigate the use of vector representations, which have been previously used for problems such as cognitive modeling or natural language processing, for knowledge representation and reasoning in automotive context. This approach to information encoding is rather generic and can be applied to various different tasks with little modifications to the representation itself. Furthermore, such vector-based representations offer the opportunity to be implemented in a spiking neuron substrate, which supports efficient learning algorithms and deployment on dedicated neuromorphic hardware. This also allows us to combine the advantages of symbolization with the benefits of neural networks. We investigate varying instantiations of our vector-based scene representation applied to different tasks.

In a first sample application, we introduce a model, that learns to classify the current driving context based on a distributed representation of the current driving scene. The conceptual focus here is to capture semantics of the scene allowing conclusions about the type of environment the vehicle is currently navigating, but also investigating how varying vector vocabularies and learning architectures influence task performance. Another essential ingredient of an environment model especially in automotive context is precise knowledge about the current state and future development of all dynamic objects in the vehicle's surroundings. We focus on the task of predicting the behavior, that is, future motion of those other traffic participants around the vehicle based on a vector-based description of the current scene using convolutive vector powers to encode spatial information. We hypothesize, that these structured representations have the potential to capture mutual interactions between dynamically moving agents. Prediction of other traffic participants' behavior also offers the opportunity to explore different learning approaches. For instance, human drivers have acquired comprehension through past experience of how other cars will probably act, but adapt this knowledge continuously when encountering new situations. From this inspiration, we learn a generic model of dynamic behavior through offline training and refine this model when perceiving behavior of a particular object through a novel mixture-of-experts model employing online learning. To complement these more high-level reasoning tasks with a perspective on motor control, we also introduce a novel neuromorphic control architectures, that can be used to implement generic control algorithms in the language of SNNs (Spiking Neural Networks). This approach allows to divide larger tasks in small sub-networks combining the advantages of manual programming with neural-network-based learning. This allows a first impression of how future neural vehicle control based on our distributed, cognitive environment model could be achieved.

Zusammenfassung

Der Wettlauf in Richtung des ersten vollkommen autonom fahrenden Automobils ist aktuell eine der Hauptursachen für den rasanten Fortschritt der Forschung im Automobilsektor. Ein Hauptgrund für diese Entwicklung in den vergangenen Jahren ist der Fortschritt im Bereich der künstlichen Intelligenz und insbesondere von sogenannten tiefen neuronalen Netzen, die außergewöhnliche Erfolge in Bereichen erzielt haben, die für das autonome Fahren von entscheidender Bedeutung sind. Der Schwerpunkt des aktuell noch jungen, doch wachsenden Forschungsgebietes *Neuromorphic Computing* liegt hingegen auf biologisch inspirierten Computersystemen und Algorithmen, die darauf abzielen das Gefälle zwischen biologischen und künstlichen Systemen hinsichtlich Leistung und Effizienz zu verringern. Obwohl neuromorphe Hardware-Prototypen technologisch noch nicht so ausgereift sind wie ihre traditionelleren Gegenstücke, so zeigen sie doch vielversprechendes Potenzial zukünftige Fahrzeuge im Hinblick auf Energieffizienz zu verbessern. Allerdings ziehen diese neuromorphen Ansätze nur langsam das Interesse der Automobilindustrie auf sich, da diese neuartigen Computerarchitekturen auf sogenannten gepulsten neuronalen Netzen (kurz SNNs vom englischen Spiking Neural Networks) basieren, welche sich drastisch von klassischen neuronalen Netzen unterscheiden und daher neuartige Paradigmen hinsichtlich Algorithmik und Programmierung benötigen werden.

Im Rahmen dieser Dissertation machen wir einen ersten Schritt in Richtung eines kognitiven Umgebungsmodells für Anwendungen im Bereich des automatisierten Fahrens unter Verwendung verteilter Darstellungen und gepulster neuronaler Netze. Wir untersuchen den Einsatz von Vektor-Darstellungen, die bisher vorrangig für die Modellierung kognitiver Prozesse oder in der Sprachverarbeitung verwendet wurden, zur Wissensrepräsentation im Fahrzeugumfeld. Dieser generische Ansatz zur Informationsdarstellung erlaubt den Einsatz in verschiedenen Anwendungsbereichen ohne größere Anpassungen an der Repräsentation selbst. Darüber hinaus eignet sich eine solche Darstellung für die Implementierung in gepulsten neuronalen Netzen, welche neuartige, effiziente Lernverfahren sowie die Verwendung auf dedizierter neuromorpher Hardware erlaubt. Weiterhin lassen sich dadurch die Vorteile symbolischer Darstellungen mit den Stärken und automatisierten Lernverfahren von neuronalen Netzen kombinieren. Wir untersuchen die Eignung unserer vektor-basierten Szenen-Repräsentation an Hand von verschiedenen Anwendungsbeispielen.

In einer ersten Anwendung präsentieren wir ein Modell, welches basierend auf unserer Vektor-Darstellung der Umgebung lernt den aktuellen Fahrkontext zu klassifizieren. Der konzeptionelle Fokus liegt dabei darauf, die semantische Bedeutung der aktuellen Fahrsituation in der Darstellung zu erfassen, aber auch den Einfluss unterschiedlicher Faktoren wie beispielsweise des zugrundeliegenden Vektor-Vokabulars oder des verwendeten Lernverfahrens zu untersuchen. Ein weiterer wichtiger Bestandteil eines Umgebunsmodells im Automobilbereich ist genaues Wissen über den aktuellen Zustand sowie die Vorhersage aller sich bewegenden Objekte im Umfeld des Fahrzeugs. Wir untersuchen die Prädiktion dieser Objekte auf Basis unserer verteilten Repräsentation. Diese nutzt eine neuartige Vektor-Darstellung räumlicher Informationen unter Verwedung von Vektor-Potenzen, welche auf der zyklischen Faltung basieren. Hier untersuchen wir die Hypothese, dass eine solche strukturierte Darstellung in der Lage ist den gegenseitigen Einfluss zwischen den verschiedenen, sich bewegenden Verkehrsteilnehmern zu erfassen. Weiterhin erlaubt dieses herausfordernde Anwendungsbeispiel die Untersuchung verschiedener Lernverfahren. Ein erfahrener Autofahrer hat beispielsweise im Laufe der Zeit ein umfassendes Verständnis dafür entwickelt, wie sich andere Verkehrsteilnehmer in bestimmten Situationen verhalten werden. Gleichzeitig ist er aber auch in der Lage, dieses Vorwissen auf Basis der aktuellen Fahrsituation kontinuierlich anzupassen und sich auf das Verhalten der anderen Fahrer spontan einzustellen. Mit dieser Inspiration entwickeln wir ein generisches Modell zur Vorhersage von dynamischen Objekten, welches auf Basis von

vorab gesammelten Daten lernt. Dieses Modell verfeinern wir durch ein sogenanntes Abstimmen-von-Experten-Modell (englisch mixture-of-experts model), welches in der Lage ist die Bewegungsvorhersage zur Laufzeit auf Basis des aktuellen Fahrkontexts anzupassen. Ergänzend präsentieren wir eine neuartige neuromorphe Steuerungsarchitektur, die die Implementierung von Steuerungsalgorithmen in gepulsten neuronaler Netzen erlaubt. Diese ermöglicht es durch die Aufteilung des Gesamtproblems in kleinere Teilnetze die Vorteile manueller Programmierung mit automatisierten Lernverfahren neuronaler Netze zu kombinieren. Dadurch gewinnen wir einen ersten Eindruck, wie eine zukünftige, neuronale Fahrzeugsteuerung auf Basis unseres kognitiven Umgebungsmodells aussehen könnte.

Acknowledgment

It is not knowledge, but the act of learning, not possession but the act of getting there, which grants the greatest enjoyment

– Carl Friedrich Gauss

Scientific work such as the doctoral thesis at hand would not be possible without the support of several people. I was fortunate enough to meet a number of great people during the preparation of this thesis I would like to express my gratitude to. First and foremost, I would like to express my gratitude towards my supervisor, Prof. Jörg Conradt, who gave me the opportunity to work on this interesting topic while giving me the freedom to creatively explore the research area. Furthermore, he introduced me to his Neuroscientific System Theory Group at Technical University of Munich, where he created an atmosphere of creativity, scientific rigor but also great camaraderie. I would also like to thank the other members of my committee, Prof. Andreas Herkersdorf for agreeing to be the second adviser of my thesis, his valuable feedback and constructive criticism when I presented an intermediate state of my work to him, as well as Prof. Werner Hemmert, for acting as chairman of the committee and his excellent organization of my (virtual) oral examination.
The team of the Neuroscientific System Theory Group was truly a great mixture of amazing people. Randomizing the order, I would like to thank Cristian Axenie for his enthusiasm, encouragement, feedback on preprints and slides as well as endless nights at the robot lab, Nikolai Waniek for feedback on ideas, preprints and slides and for discussions on scientific as well as any other topics of current interest, Christoph Richter for proof-reading papers as well as his pragmatic approach to research, Lukas Everding for his deadpan sense of humor and Emeç Erçelik for his help and support with organizing practical lectures at the University as well as at the BMW Summer School. Prof. Conradt also introduced me to the neuromorphic engineering community in general and to Dr. Terry Stewart from CNRG (Computational Neuroscience Research Group) at University of Waterloo in particular, who also had a great impact on my doctoral research. I would like to thank him for his continuous support across the Atlantic Ocean and for generously sharing his wide knowledge on spiking neurons, Nengo, and machine learning in general, but moreover for becoming a great mentor and role model for me. Furthermore, I am also deeply grateful to Prof. Chris Eliasmith, the head of CNRG, as well as Terry and all the members of CNRG for hosting me during a six week research visit at University of Waterloo in summer 2017. This visit not only allowed me to push my research forward thanks to the remoteness of being an ocean and a six hour time delay apart from the "usual business". Furthermore, I was also fortunate to meet lots of great people at University of Waterloo sharing their knowledge and experience, but I also had a great time at jam sessions or simply visiting and enjoying the beauty of Canada. My research visit also allowed me to initiate a collaboration project between ABR (Applied Brain Research Inc.), a CNRG spin-off company, and BMW Group with some results of that project making their way into this thesis. As part of this collaboration project, I would like to thank Terry, Peter Blouw, Daniel Rasmussen and Eric Hunsberger on the technical side, but also Peter Suma on the administrative side for fighting with me through the pain of paperwork to make the project actually work.
I am also deeply thankful to Dr. Hans-Jörg Vögel, my supervisor at BMW Group, for giving me the opportunity to pursue my Ph.D. in the environment of BMW's research department. Furthermore, he always supported and advised me with scientific and administrative tasks, defended the idea of applying neuromorphic computing "in a car" within the company and always had an "open door", whenever something unexpected occurred. I would also like to thank Martin Arend, the leader of my department for the most time during my Ph.D. years at BMW Group for being the nicest boss a Ph.D. student could

hope for and furthermore, for not only supporting my research visit to Canada, but also joining for a couple of days and giving me the opportunity to accompany him when meeting the local start-up scene. Another great part of pursuing my doctoral studies at BMW Group was the network of fellow Ph.D. students within BMW's ProMotion program. Meeting and talking to other students with similar tasks, success stories but also problems and frustrations helped me to overcome these usual but unpleasant parts of doctoral studies when progress stagnated. In particular, I would like to express my gratitude towards Leopold Walkling and Julian Tatsch for sharing their knowledge, experience and data sets. Furthermore, I would like to thank the "Improve" group (in no particular order): Franziska Hertlein, Sascha Steyer, Nicola Hupp, Florian Roider, Jens Schulz, Marc Vogt, Daniel Knobloch, Alexander Terres, Jan Korus, Julius Riedelbauch, Michael Ponnath, Peter Rösch, Philip Kotter and Annette Böhmer. I would also like to thank my fellow Ph.D. students at our department LT-3 Maike Hartstern and Christoph Segler for sharing the ups and downs of a Ph.D. student at BMW in general and at LT-3 in particular as well as interesting scientific and daily-life discussions. Particularly, I would like to thank Christoph for our shared efforts regarding the "management" of the dev-box as well as remembering literally every single administrative problem one could run into at BMW and, more importantly, its solution. I would also like to express my gratitude to a couple of "regular" BMW colleagues for giving feedback to papers, talks or slides, helping out with administrative issues, interesting lunch-break discussions or simply for being great colleagues. Again, randomizing the order, I would like to thank Mohsen Kaboli, Tom Hubregtsen, Sebastian Wirkert, Fridolin Bauer, Viktor Rack and Suomy Jacob. Additionally, I would like to thank Angelika Schäfer and Simone Barnloher as well as Susanne Schneider for their help and patience regarding administrative tasks related to business trips at BMW or student supervision at TUM respectively. Finally, I would like to thank all the students I had the pleasure to supervise either at BMW or at TUM doing master or bachelor theses, project practicals, scientific seminars or simply lectures. I hope they have learned as much from me as I learned from them.

I am also deeply thankful to my parents, Rainer and Ingrid Mirus, for being a constant source of support and encouragement when things do not work as originally planned and for providing a basement to partly finish writing of this thesis. Thank you for supporting my life-long journey from the first steps up until this point and beyond. Furthermore, I would like to express my gratitude to Helga Wallitzer, my mother-in-law, especially for her support during the first weeks after the birth of our daughter Isabella, which allowed me to find some sleep and get at least some work done during that great yet stressful time. Last but not least, I would like to express the most heartfelt gratitude to my lovely wife Lisa for being on my side ever since I made the decision to move to Munich for the Ph.D., enduring my temper during the frustrating times, supporting me selflessly in everything that I do and for making me the greatest gift in the world: our children Isabella and Jannik. I could not have finished this thesis without you. Thank you for everything!

Florian Mirus
Munich, December 2020

Table of Contents

List of Abbreviations

ABR Applied Brain Research Inc.. v, 6
ACC Adaptive Cruise Control. 24
ACT Adaptive Control of Thought. 21
ACT-R Adaptive Control of Thought-Rational. 21
ADAS Advanced Driver Assistance System. 24, 25, 27, 53
AER Address Event Representation. 15, 16
AHSRA Advanced Cruise-Assist Highway System Research Association. 24
AI Artificial Intelligence. 1, 20, 28, 32
ANC Analog Network Core. 15
ANN Artificial Neural Network. 9, 10, 12, 13, 21, 53
APS Active Pixel Sensor. 16
ART Adaptive Resonance Theory. 10

BBP Blue Brain Project. 11
BMW Bayerische Motoren Werke. 6
BrainScaleS Brain-inspired multiscale computation in neuromorphic hybrid systems. 11, 14, 19
BSC Binary Spatter Code. 22, 35, 36, 40, 50

CCD Charge Coupled Device. 15
CNN Convolutional Neural Network. xvi, xxi, 10, 19, 28, 48, 57–60, 66, 84–86, 88, 91, 92, 155
CNRG Computational Neuroscience Research Group. v, 6
CPU Central Processing Unit. 14, 138
CUAVE Clemson University Audio Visual Experiments. 19

DARPA Defense Advanced Research Projects Agency. xv, 1, 11, 16, 24, 25
DAS Dynamic Audio Sensor. 16, 18
DAVIS Dynamic and Active Pixel Vision Sensor. 16
DBN Deep Belief Network. 18, 19, 48
DDR Double Data Rate. 15
DFT Discrete Fourier Transform. 37, 41–43, 51, 70
DNN Deep Neural Network. 2, 10, 13, 19, 22, 28, 31, 53, 57
DVS Dynamic Vision Sensor. xv, 16–20, 137, 138, 143, 144

eDVS embedded Dynamic Vision Sensor. 138
EPIC Executive-Process/Interactive Control. 21
EU European Union. 11

FACETS Fast Analog Computing with Emergent Transient States. 11, 14
FCN Fully Convolutional Neural Network. xv, 28
FET Future Emerging Technologies. 11
FLOPS Floating Point Operations per Second. 15

GloVe Global Vectors. 22, 49
GOFAI Good Old-Fashioned Artificial Intelligence. 21
GPS

1) General Problem Solver. 21
2) Global Positioning Systems. 24, 25, 27, 79
GPU Graphics Processing Unit. 1, 10, 14, 15, 28, 138
GTSRB German Traffic Sign Recognition Benchmark. xvi, 31, 58, 63, 90

HBP Human Brain Project. 11
HICANN High Input Count Analog Neural Network. 11, 15
HMM Hidden Markov Model. 19
HOG Histogram of Oriented Gradients. 28
HRL Hughes Research Laboratories. 11, 16
HRR Holographic Reduced Representation. 22, 35, 37, 40
HTM Hierarchical Temporal Memory. 22

IDFT Inverse Discrete Fourier Transform. 37, 41–43, 51, 70
IF Integrate-and-Fire. 15
IMU Inertial Measurement Unit. 138

LIDAR Light Detection and Ranging. 3, 25, 26, 28, 31, 79, 94
LIF Leaky-Integrate-and-Fire. xv, 12–14, 83, 85, 115
LSTM Long Short-Term Memory. viii, xv, xviii, 5, 6, 30, 93, 98–117, 120, 121, 123, 127, 129, 135, 156

MAP Multiply-Add-Permutate. 22, 35–37
MNIST Mixed National Institute of Standards and Technology. 17–19, 31
MS COCO Microsoft Common Objects in COntext. 31

NAHSC National Automated Highway System Consortium. 24
NEF Neural Engineering Framework. viii, ix, xvi, xix, 5, 13, 14, 16, 19, 22, 33, 35, 43–47, 50–53, 83, 99, 102–104, 115–117, 120, 121, 125, 150, 155
Nengo Neural Engineering Objects. xvii, xx, 13, 14, 19, 43, 45, 83, 85–89, 91, 101–103, 115, 139, 141, 145, 147, 148
NEST NEural Simulation Tool. 14
NGSIM Next Generation Simulation. viii, xvii–xix, 31, 93, 95–97, 101, 104, 108, 110, 112, 115–117, 119–121, 129, 130
NLP Natural Languange Processing. 49, 50
NPU Neuromorphic Processing Unit. 138
NST Neuroscientific System Theory Group. v

PACMAN Partitioning And Configuration MANager. 14
PES Prescribed Error Sensitivity. 150
PID Proportional-Integral-Derivative. 138
PROMETHEUS PROgraMme for a European Traffic of Highest Efficiency and Unprecedented Safety. 24
PyNCS Python Neuromorphic Cognitive Systems. 14
PyNN Python Neural Networks. 11, 13, 14, 19

RADAR Radio Detection and Ranging. 3, 25, 26, 79
RBF Radial Basis Functions Network. 10
RBM Resctricted Boltzmann Machine. 10, 13, 18, 19, 48
ReLU Rectified Linear Unit. 85, 86
RMSE Root-Mean-Square Error. xvi, xviii–xx, 70, 103, 105–116, 130–135
RNN Recurrent Neural Network. 10
ROLLS Reconfigurable On-Line Learning Spiking. 15
ROS Robot Operating System. 148, 149

RPM
1) Raven's Progressive Matrix. 50, 51
2) Rotations per minute. 149

SAE Society of Automotive Engineers. 23
SDR Sparse Distributed Representation. 22
SDRAM synchronous dynamic random-access memory. 15
SIMD single instruction multiple data. 14
SLAM Simultaneous Localization and Mapping. 18, 26
SNN Spiking Neural Network. i, iii, vii, viii, xx, 2, 4–6, 12–15, 18, 31, 32, 35, 47, 50–53, 91, 120, 137, 147, 148, 150–152, 156–158
SOM Self-Organizing Map. 10, 48
SPA Semantic Pointer Architecture. vii, xvii–xix, 4, 22, 32, 33, 35, 40–42, 47, 50–54, 57, 63–66, 68, 72–74, 79, 97, 101, 104, 109–117, 120, 121, 127, 129, 155
Spaun Semantic Pointer Architecture Unified Network. 13, 14, 43, 48
SpiNNaker Spiking Neural Network Architecture. xv, 2, 11, 14, 15, 19, 20, 159
STDP Spike Timing Dependant Plasticity. 13, 14, 19
STM synaptic time multiplexing. 16
SVM Support Vector Machine. 10
SyNAPSE Systems of Neuromorphic Adaptive Plastic Scalable Electronics. xv, 1, 11, 16

TORCS The Open Racing Car Simulator. 137, 148, 149
TUM Technical University of Munich. 6

US Ultrasonic Sensors. 25

VLSI Very-Large-Scale Integration. 11, 15
VSA Vector Symbolic Architecture. vii, viii, 3, 4, 9, 22, 32, 35–40, 47, 49–53, 56, 57, 62, 68–70, 72, 73, 103, 155

List of Figures

List of Tables

1 Introduction

1.1 Preamble

The race to autonomous driving is currently one of the main forces pushing research forward in the automotive domain. With highly automated vehicle prototypes gradually making their way to our public roads and fully-automated driving on the horizon, it seems to be a matter of time until we see robot taxis or cars navigating us through urban traffic or heavy stop-and-go on highways. One major reason for this development in recent years is the rapid progress of AI (Artificial Intelligence), especially the success of deep learning, which has shown remarkable results in tasks essential for autonomous driving such as object detection, classification (Ciresan et al., 2012a) and control (Bojarski et al., 2016). Although more and larger data sets (Geiger et al., 2013; Cordts et al., 2016) become available and powerful, parallel computing hardware like GPUs (Graphics Processing Units) facilitates training of increasingly deep network architectures (Simonyan and Zisserman, 2014), power consumption remains a critical issue. Especially in the automotive domain as a mobile application, energy-efficiency is of crucial importance. Furthermore, current automated vehicle prototypes rely on a rich setup of redundant sensory systems to perceive sufficient information about the outside world (Aeberhard et al., 2015). These sensor setups are estimated and expected to grow even further with increasing level of automation of future vehicles. In contrast, human drivers are capable of handling challenging driving situations under changing environmental conditions by using mainly the human eyes as primary sensor input and the brain for information processing. Furthermore, the human brain is also comparably small and efficient consuming only 20 W of power, which is equivalent to a compact fluorescent light bulb, while comprising only 2 % of the body weight (Eliasmith, 2013, Chap. 2.1). Therefore, the focus of the young and emerging research field neuromorphic engineering is on biologically inspired computing systems and algorithms, aiming to close the gap in performance and efficiency between biological and artificial computing systems. With researchers expecting a growing discrepancy between the energy and efficiency requirements of future applications, e.g., see Fig. 1.1 as well as Marr et al. (2013), Farahini (2016), and Akopyan et al. (2015), the demand for alternative approaches regarding computing hardware and algorithms is likely to increase. Although

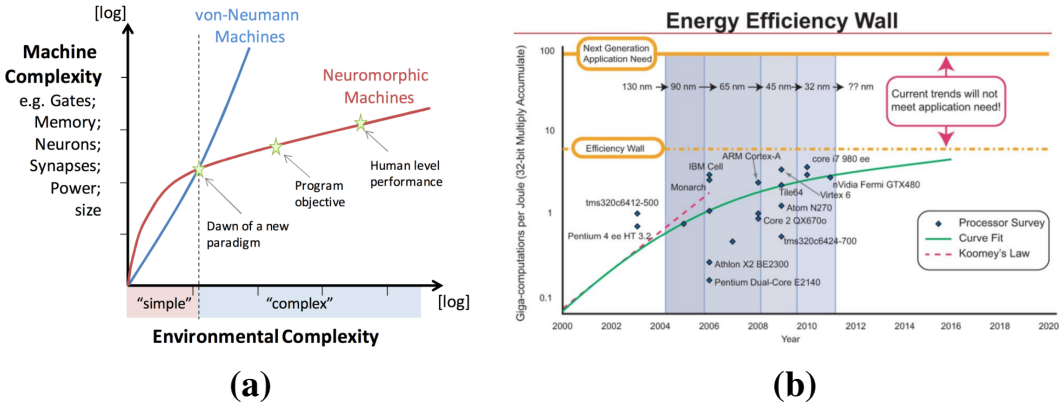


Figure 1.1: Expected discrepancy between energy-efficiency requirements of future applications and the development of computing hardware. (a) Image source: Farahini (2016), adapted from the DARPA SyNAPSE program. (b) Image source: Marr et al. (2013).

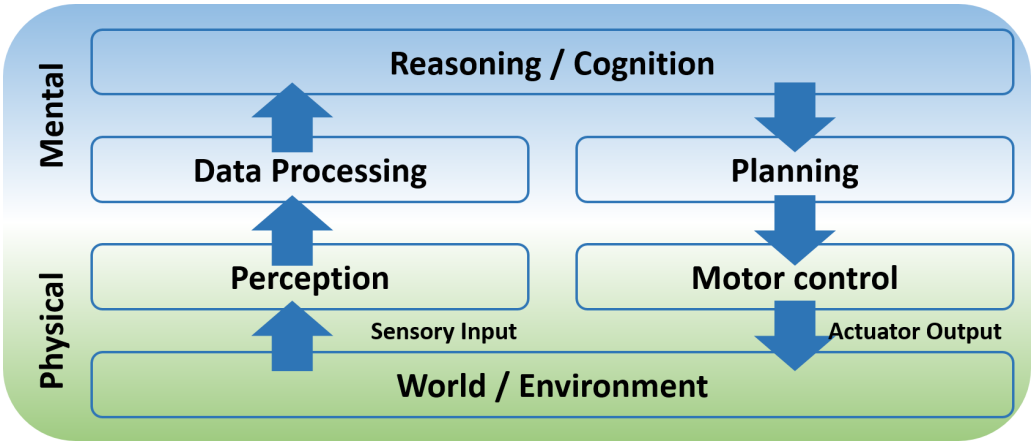


Figure 1.2: Schematic visualization of the perception-action-cycle

the neuromorphic prototypes of computing hardware such as SpiNNaker (Furber et al., 2014) or IBM's TrueNorth (Akopyan et al., 2015) dedicated to processing so-called SNNs are not technologically mature nor available as commercial products yet, they show promise to be a useful addition in future automotive applications. Since these novel spiking-neuron architectures encapsulate a drastically different computing paradigm compared to the conventional von-Neumann architecture (Neumann, 1993), they also call for alternative algorithmic approaches and new programming substrates (Amir et al., 2013).

In this thesis, we investigate potential applications of neuromorphic approaches in automotive context. Given the complexity of driving a car autonomously in all possible real-world situations, tackling the complete set of all necessary tasks is out of scope of a single thesis. Therefore, we need to focus on certain sub-tasks. Regarding this selection process, we contemplate two concepts. We categorize tasks necessary for highly automated driving using the perception-action-cycle and rate their level of complexity using Moravec's paradox (Moravec, 1988). Moracev's paradox postulates the observation, that tasks or skills that seem effortless to humans are harder to reverse-engineer than tasks we experience or expect to be difficult. Moracev argues, that we learned those seemingly effortless skills over billions of years of evolution through experience about the nature of the world such that they became unconscious to us. A good example of this paradox is the fact that artificial systems are already able to defeat the world's best human players in games like Chess (Hsu, 2002) or Go (Silver et al., 2016), which are considered particularly difficult by humans. On the other hand, we are still struggling to create robots that solve seemingly effortless sensorimotor tasks like climbing stairs or opening doors (Guizzo and Ackerman, 2015; Norton et al., 2017).

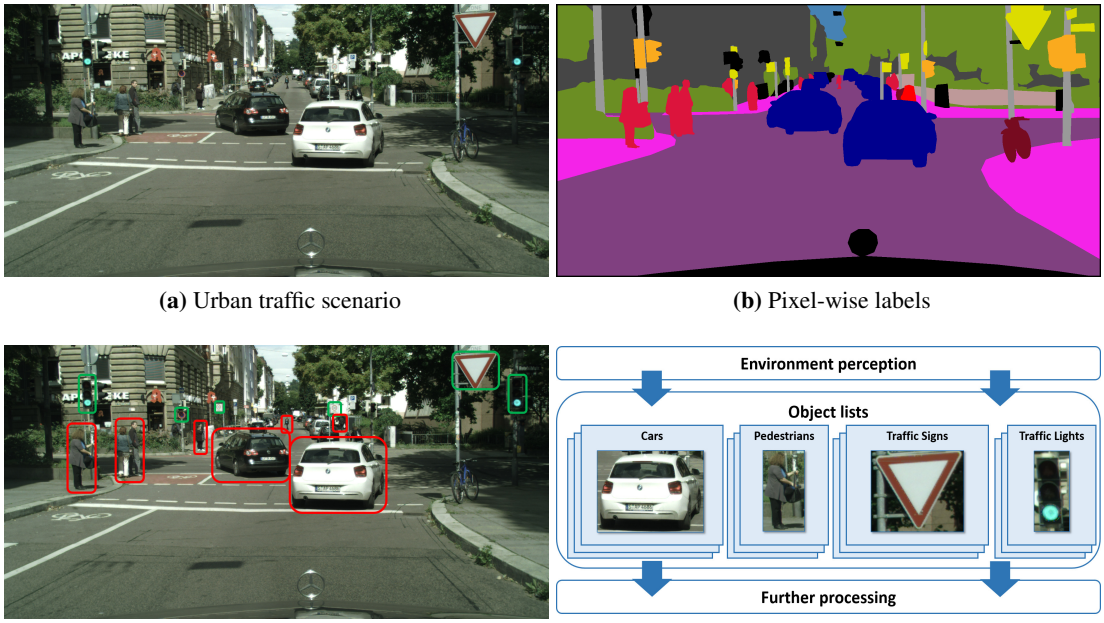
The perception-action-cycle (Fuster, 2004) is the concept of a circular flow of information between an organism or agent (living or artificial) when interacting with its environment through goal-directed behavior (Fig. 1.2). Instead of considering the directional aspects of the cycle in the opposite categories of perception and action, it is also common to separate tasks in terms of hierarchy (Loeb and Fishel, 2014). Interaction with the physical world through sensing and manipulation is considered on a lower hierarchical level within the perception-action-cycle while formation of mental representations and reasoning about them reside on a higher level. We refer to those different hierarchical levels as *physical* and *mental* or *lower* and *upper* interchangeably.

Considering the physical/lower level of the perception-action-cycle, DNNs (Deep Neural Networks) have recently shown significant progress and success at perception tasks like object detection and classification. Therefore, we believe a lot more research is necessary until neuromorphic approaches using SNNs are sufficiently mature to compete with those traditional approaches, although current work in this directions shows promising results (Hunsberger and Eliasmith, 2015). Similar arguments hold true for automated learning approaches regarding vehicle control. Although sophisticated learning techniques show promise to improve motor control, to date most controllers for automated vehicles or robots in general

are "designed and tuned by human engineers" (Deisenroth et al., 2013). One of the main reasons is the fact that control of an automated vehicle is an extremely safety-critical domain, while at the same time machine learning approaches in general pose additional challenges regarding safety validation (Koopman and Wagner, 2016). Furthermore, those tasks on the physical level of the perception-action-cycle are solved effortlessly by humans and therefore, according to Moravec's paradox, we should expect them to be hard to master for artificial learning systems. Therefore, we decided to focus on the mental/upper part of the perception-action-cycle in this thesis. On the one hand, we believe that tasks in this part of the cycle show promise to benefit most from neuromorphic approaches. On the other hand, "mental" tasks do not necessarily need to be performed and evaluated in closed-loop systems, which make them less problematic regarding safety issues, and therefore ideal candidates for further investigations. It should be clear however, that "perception/action and higher level cognition are not two independent parts of one systems, but rather two integrated aspects of cognitive beings such as as ourselves" (Eliasmith, 2013). Current artificial systems are still far away from integrating both aspects effectively and more research will be necessary to close this gap.

Returning to the application domain of automated driving, precise knowledge about the current state of the environment is essential for autonomous agents and biological organisms alike to plan a secure path for navigation and to safely interact with the world. In case of highly automated vehicles, perception of the outside world usually happens through a variety of different sensor systems such as cameras, RADAR and LIDAR sensors (Aeberhard et al., 2015). This observed information needs to be collected and combined into a central environment model, which is the (mental) basis for further reasoning and decision making. One essential ingredient for such a model of the environment, or mental tasks in general, is knowledge or information representation. It is an open research question to date, how the human brain represents knowledge and what underlying neural or computational substrate it uses to encode information (Wang and D. Liu, 2003; Samsonovich, 2012; Handjaras et al., 2016). Most modern approaches to reasoning or cognitive tasks in context of robotics or automated driving are built upon Bayesian probability theory and use "computer-scientific" approaches to knowledge representation. This could be lists of objects (cf. Fig. 1.3d) with a numerical encoding of properties when working on a higher level of abstraction. On a lower level, another common approach especially in context of neural-network learning is to label raw sensory data, for example individual pixels of images (cf. Fig. 1.3b). However, when we as humans observe a particular scene (e.g., while driving), our mental representation will probably be very different from those aforementioned approaches. A human observer's representation/description of Fig. 1.3a would probably be in a semantic/linguistic form, for instance *a blue car turning left, a white car turning right, three pedestrians waiting on the left side of the road, green traffic lights, a yield way sign*. One common approach to encode such conceptual, semantic information or, more generally, natural language in computer-readable fashion is by using vector representations. VSAs (Vector Symbolic Architectures) is a term coined by Gayler (2003) to cover this family of modeling approaches that represent symbols or structures by mapping them to (high-dimensional) vectors.

In this thesis, we present a first step towards a cognitive environment model for automotive applications using distributed representations. We investigate the use of these vector representations for knowledge representation and reasoning in automotive context. This approach to information encoding is rather generic and can be applied to various different tasks with little modifications to the representation itself. Furthermore, VSAs (Vector Symbolic Architectures) offer the opportunity to be implemented in a spiking neuron substrate (Eliasmith, 2013), which support efficient learning algorithms and deployment on dedicated neuromorphic hardware. This allows us to combine the advantages of symbolization with the benefits of neural networks. We investigate varying instantiations of our representation applied to different tasks. In a first sample application, we introduce a model learning to classify the current driving context based on a distributed representation of the current driving scene. The conceptual focus here is to capture semantics of the scene allowing conclusions about the type of environment the vehicle is currently navigating. Another essential ingredient of an environment model especially in automotive context is precise knowledge about the current state and future development of all dynamic objects in the



(a) Urban traffic scenario

(b) Pixel-wise labels

(c) Bounding boxes indicating objects of interest

(d) Exemplary representation using lists of objects

Figure 1.3: Example of urban driving scene with different approaches to representation. Images 1.3a, 1.3b and 1.3c are (adapted) from the Cityscapes data set (Cordts et al., 2016).

ego-vehicle's surroundings. We focus on the task of predicting the behavior of those other traffic participants around the ego-vehicle based on a vector description of the current scene. We hypothesize, that these structured representation have the potential to capture mutual interactions between dynamically moving agents. Prediction of other traffic participants behavior also offers the opportunity to explore different learning approaches. Human drivers have acquired comprehension through past experience of how other cars will probably act, but adapt this knowledge continuously when encountering new situations. From this inspiration, we learn a generic model of dynamic behavior through offline (i.e., batch) training and refine this model when perceiving behavior of a particular object through online learning. To complement the more high-level reasoning tasks with a perspective on motor control, we also introduce a novel neuromorphic control architecture, that can be used to implement generic control algorithms in the language of SNNs. This approach allows to divide larger tasks in small sub-networks combining the advantages of manual programming with neural network learning.

1.2 Outline of the thesis

This section provides a brief overview of the thesis structure as well as a short summary for each chapter. Chapter 2 summarizes the state-of-the-art in several areas related to the core of the thesis at hand spanning from biologically inspired hardware and software over cognitive modeling techniques to automated driving research. We give an overview over all of these sub-topics providing a detailed description where necessary while putting these details in context of a bigger picture.

Chapter 3 introduces the key ingredients for the models developed in later chapters, distributed representations and SNNs, and establishes the mathematical apparatus and the essential theoretical properties of these ingredients. After proposing a more rigorous mathematical formalism to describe the general theory of VSAs, we proceed to the SPA as a special case of VSAs with additional features and theory to be presented. Furthermore, we give an introduction to the general ideas for cognitive modeling based on VSAs like vocabularies and structured representation generation. Furthermore, we present a brief

description of the NEF and how it can be used to implement cognitive architectures based on vector representations in a spiking neuron substrate.

In chapter 4, we proceed to present the general approach to encode automotive scenes in semantic vectors as representational substrate. We show different approaches to generate vector vocabularies, which are the basic ingredients to built more complex, structured representations from. We demonstrate successful embedding of several similarity structures into vector vocabularies designed for automotive applications. Additionally, we present several approaches to representing numerical values in our vector substrate introducing a novel approach based on a convolutive power, generalizing exponentiation to circular convolution. We conclude the chapter with a thorough analysis regarding the systematic limitations regarding the vectors' capacity, i.e., amount of information such vector representations can effectively store.

Chapter 5 introduces the first sample instantiation of our vector representation applied to the task of driving context classification based on a scene vector encapsulating the current driving situation. We establish the vector-based scene representation for the context classification task and use it as input data to a learning model implemented in a spiking neuron substrate. To evaluate the model's performance, we present several reference learning systems using either the vector representation or visual information as input and compare them to human level performance. Finally, we analyze the influence of the underlying vocabularies encoding different similarity structures on the learning model's classification performance. All evaluations are conducted using real-world driving data.

In chapter 6, we introduce a second instantiation of our scene representation to predict the behavior of other objects in the vehicle's surroundings. In the context of vehicle trajectory prediction, we employ our novel approach to encode spatial information in distributed representations to encode the positions of several vehicles in vectors of fixed length. We employ neural networks built from LSTM units as well as simpler single-layer SNNs to predict vehicle trajectories based on past motion. We analyze the influence of hyperparameters, information provided to the models as well as the composition of the training data on the models' prediction accuracy. Furthermore, we compare the models with respect to situations in which one particular model outperforms the others. Finally, we show that it is generally possible to detect abnormal samples indicating potentially dangerous situation based solely on the distributed vector representation and unsupervised learning. The evaluation conducted in the chapter uses data from two different data sets containing real-world driving data.

Chapter 7 introduces an extension of the proposed trajectory prediction models for online adaptation through incremental learning. Building on the findings in chapter 6, we present a novel mixture-of-experts model implemented in a spiking neuron substrate meant to be trained at run time to improve the prediction performance over several individual predictors. One of the strengths of this model is that, instead of having to start from a completely blank state, it relies on several expert models, which have been previously trained and validated offline. This model, like all models making predictions about the future, faces the issue that the actual motion of the predicted vehicle needed to compare the model's anticipated values to is future data and thus not available at the time the model actually makes its predictions. To avoid potentially long delays in the online learning process, we propose a novel approach to spread the error signal of earlier predictions over later predictions. Revisiting the data sets used in chapter 6, we evaluate two different variants of the mixture models adapting their weights either solely based on the prediction error or the current context of the scene. We show, that our online learning model is able to improve over the individual predictors already after being exposed to a small set of example vehicles.

Chapter 8 presents a first step towards a neuromorphic control architecture by developing two sample instantiations of neurally-inspired control algorithms. We establish neuromorphic control architectures, that can be used to implement generic control algorithms in the language of SNNs with the advantage, that the overall task can be divided into several sub-networks. This approach allows us to combine manual programming with the advantages of neural network learning. Manually programmed sub-tasks can either complement learning networks within the system or serve as an initial approximation of the desired function, which allows more directed learning to improve task performance avoiding the need to

start learning from a completely blank state. As mentioned earlier, control of an automated vehicle is an extremely safety-critical application. Hence, we present two sample applications in simplified setups: one on mobile robot manipulation demonstrating initial manual programming of a non-trivial robotic task in a spiking neuron substrate and one on vehicle trajectory control demonstrating how manually programmed modules can be complemented by learning networks.
Chapter 9 summarizes the work and interprets the results achieved in this thesis. We discuss the main advantages of the representations and models proposed in this work while also addressing limitations indicated by either systematic or experimental analysis. We conclude the thesis by proposing a series of extensions and improvements, which potentially contribute to adopting the principles investigated in this thesis on a wider scale to obtain a more mature representation framework for automotive applications.

1.3 Contributions of and to this thesis

This thesis presents a first step towards a cognitive environment model for automated vehicles and thereby provides a novel perspective to knowledge representation in automotive context. The two key ingredients for our approach are distributed representations and SNNs, which in combination offer a promising modeling substrate for cognitive tasks as shown in Eliasmith (2013) and Eliasmith et al. (2012). The possibility to implement distributed representations in a spiking neuron substrate offers the potential to combine symbol-like processing with the strengths of neural network learning. Additionally, SNNs are one option to tackle increasing energy-efficiency requirements in future automated vehicles equipped with a plethora of sensors and computing systems. This thesis presents a strict mathematical formalism regarding the theory of distributed representations, summarizing their key features in a generic framework. Thereby, we lay the foundation for our novel approach to build structured representations of automotive scenes. We demonstrate the general feasibility of our approach in two sample instantiations. For all the models and representation approaches investigated in this thesis, we present a thorough and detailed analysis regarding parameters and accuracy including a comparison to several baseline models. While the models employing our structured representations achieve promising results in terms of accuracy without clearly outperforming more traditional approaches, we expect the critical benefits of our approach to be revealed when being actually deployed on specialized computing hardware. As shown by Hunsberger and Eliasmith (2016), implementing learning models in SNNs is often a trade-off between energy-efficiency and accuracy. Finally, we establish neuromorphic control architectures, that can be used to implement generic control algorithms in the language of SNNs. This approach can benefit from splitting larger tasks in small sub-networks, that can be manually programmed to complement or bootstrap learning networks.
This dissertation was supported by the BMW (Bayerische Motoren Werke) AG, where I was employed as doctoral candidate and later in a permanent position during the preparation of this thesis. Therefore, parts of the literature review and text that presents the research context in chapter 2, especially the section on neuromorphic computing hardware, was conducted in collaboration with BMW colleagues. The research on the mobile manipulation task was conducted during a collaboration project between TUM (Technical University of Munich) and University of Waterloo while the work on vehicle trajectory control was supported by Benjamin Zorn during his internship at BMW. The approaches, principles and models presented in the sections on structured vocabularies in chapters 4 and 5 were developed in collaboration with Robert Darius during the preparation of his Master's thesis (Darius, 2018) under my supervision at BMW Group. Many of the novel approaches presented in this thesis arose from discussions with researchers from the CNRG at University of Waterloo, where I spent a six week research visit in summer 2017. This research visit spawned a follow-up collaboration project between BMW and ABR (Applied Brain Research Inc.), during which parts of chapters 6 and 7 were developed in collaboration. This collaboration covered mainly the implementation of the LSTM model based on numerical input in chapter 6 as well as discussions about the online learning approach presented in chapter 7. Researchers involved in this collaboration co-authored some of the publications listed in 1.3.1.

Finally, figures displayed in this thesis, which have been reprinted or adapted from others or quotations are clearly marked as such. Anything not indicated as quotation or external source is the author's original work.

1.3.1 List of Publications

The following list gives an overview of the publications written and submitted during the preparation and work on this thesis.

Published peer-reviewed journal papers

1. F. Mirus, P. Blouw, T. C. Stewart, and J. Conradt (2019a-10). "An Investigation of Vehicle Behavior Prediction Using a Vector Power Representation to Encode Spatial Positions of Multiple Objects and Neural Networks". In: *Frontiers in Neurorobotics* 13, p. 84. ISSN: 1662-5218. DOI: 10.3389/fnbot.2019.00084. URL: https://www.frontiersin.org/article/10.3389/fnbot.2019.00084

2. F. Mirus, C. Axenie, T. C. Stewart, and J. Conradt (2018a). "Neuromorphic sensorimotor adaptation for robotic mobile manipulation: From sensing to behaviour". In: *Cognitive Systems Research* 50, pp. 52–66. ISSN: 1389-0417. DOI: 10.1016/j.cogsys.2018.03.006. URL: http://www.sciencedirect.com/science/article/pii/S1389041717300955

Published peer-reviewed conference papers

1. F. Mirus, T. C. Stewart, and J. Conradt (2020b-10-02). "Detection of abnormal driving situations using distributed representations and unsupervised learning". In: *28th European Symposium on Artificial Neural Networks, ESANN 2020, Bruges, Belgium*

2. F. Mirus, T. C. Stewart, and J. Conradt (2020c-07-19). "The Importance of Balanced Data Sets: Analyzing a Vehicle Trajectory Prediction Model based on Neural Networks and Distributed Representations". In: *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, pp. 1–8. DOI: 10.1109/IJCNN48605.2020.9206627

3. F. Mirus, T. C. Stewart, and J. Conradt (2020a-07-19). "Analyzing the Capacity of Distributed Vector Representations to Encode Spatial Information". In: *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, pp. 1–7. DOI: 10.1109/IJCNN48605.2020.9207137

4. F. Mirus, T. C. Stewart, C. Eliasmith, and J. Conradt (2019c). "A Mixture-of-Experts Model for Vehicle Prediction Using an Online Learning Approach". In: *Artificial Neural Networks and Machine Learning – ICANN 2019: Image Processing*. Ed. by I. V. Tetko, V. Kůrková, P. Karpov, and F. Theis. Vol. 11729. Lecture Notes in Computer Science. Springer International Publishing, pp. 456–471. ISBN: 978-3-030-30508-6. DOI: 10.1007/978-3-030-30508-6_37

5. F. Mirus, P. Blouw, T. C. Stewart, and J. Conradt (2019b). "Predicting vehicle behaviour using LSTMs and a vector power representation for spatial positions". In: *27th European Symposium on Artificial Neural Networks, ESANN 2019, Bruges, Belgium*, pp. 113–118

6. F. Mirus, B. Zorn, and J. Conradt (2019d). "Short-term trajectory planning using reinforcement learning within a neuromorphic control architecture". In: *27th European Symposium on Artificial Neural Networks, ESANN 2019, Bruges, Belgium*, pp. 649–654

7. F. Mirus, T. C. Stewart, and J. Conradt (2018b). "Towards cognitive automotive environment modelling: reasoning based on vector representations". In: *26th European Symposium on Artificial Neural Networks, ESANN 2018, Bruges, Belgium*, pp. 55–60

2 Research Context

Highly automated driving is currently an immensely attractive field for both academic and industrial research groups. A fully autonomous vehicle, which is able to tackle challenging driving situations without external input comparable to a human driver's performance, is yet to be build. In this thesis, we propose a novel approach to knowledge and information representation for automotive environment models using cognitive modeling techniques. More precisely, we adopt VSAs (Vector Symbolic Architectures), which are commonly used in tasks like cognitive modeling and natural language processing, for the specific application of automotive environment modeling. To our knowledge, VSAs have not been applied in this particular context. In order to put our work in context of the current research landscape and to circumscribe this thesis, we need to review related work from several areas like automated driving and mobile robotics, computational neuroscience, artificial intelligence and neuromorphic engineering. A comprehensive overview for all of these research areas is out of scope of a single thesis. However, we aim to cover the most significant results from all areas at least briefly, whereas we present an in-depth review of relevant work related to the thesis at hand, where necessary.

2.1 Biologically-inspired Systems

To fully understand biological systems like the brain, which evolved over millions of years, is an ongoing yet unsolved challenge in biology and neuroscience. Even small animals like insects or rodents show remarkable behavioral flexibility and the ability to constantly adapt to a rapidly changing and noisy world, which is unmatched by modern computing machines. Mammals and primates are able to perform more sophisticated behaviors culminating in complex cognitive computations humans are capable of doing: thinking, problem solving, memory, reasoning, decision-making, strategic planning, knowledge representation, learning etc. The "biological computer" enabling these behaviors and cognitive abilities is the brain consisting of large networks of neural cells (or neurons), which communicate by sending and receiving electric signals via synapses. At the same time, brains are comparably small and efficient: the human brain for example consumes only 20 W of power (equivalent to a compact fluorescent light bulb) and comprises 2 % of the body weight (Eliasmith, 2013, Chap. 2.1).
Several research fields like computational neuroscience, neuromorphic engineering and neurorobotics try to reverse engineer biological systems to achieve similar performance and computational power. During the last decades, researchers and engineers strived to close the gap in performance and efficiency between biological and artificial computing systems by mimicking neuro-biological architectures in hardware and implementing models of neural systems in software. This biologically inspired, *neuromorphic* approach promises not only to perform computations in a more efficient way, but also to tackle problems unsolvable with current computing machines.

2.1.1 A brief history

The research field of ANNs (Artificial Neural Networks) goes back to the 1940s when McCulloch and Pitts (1943) introduced artificial neurons as computational units, which embody a simplified model of biological neurons. These first simple networks were able to calculate compositions of basic logic functions (McCulloch and Pitts, 1943; Rojas, 1996). Rosenblatt (1958) proposed the first neural network, which was capable of learning, by adding numerical weights to the connections of the network with threshold functions as activation functions: the *perceptron*. Minsky and Papert (1969) showed that

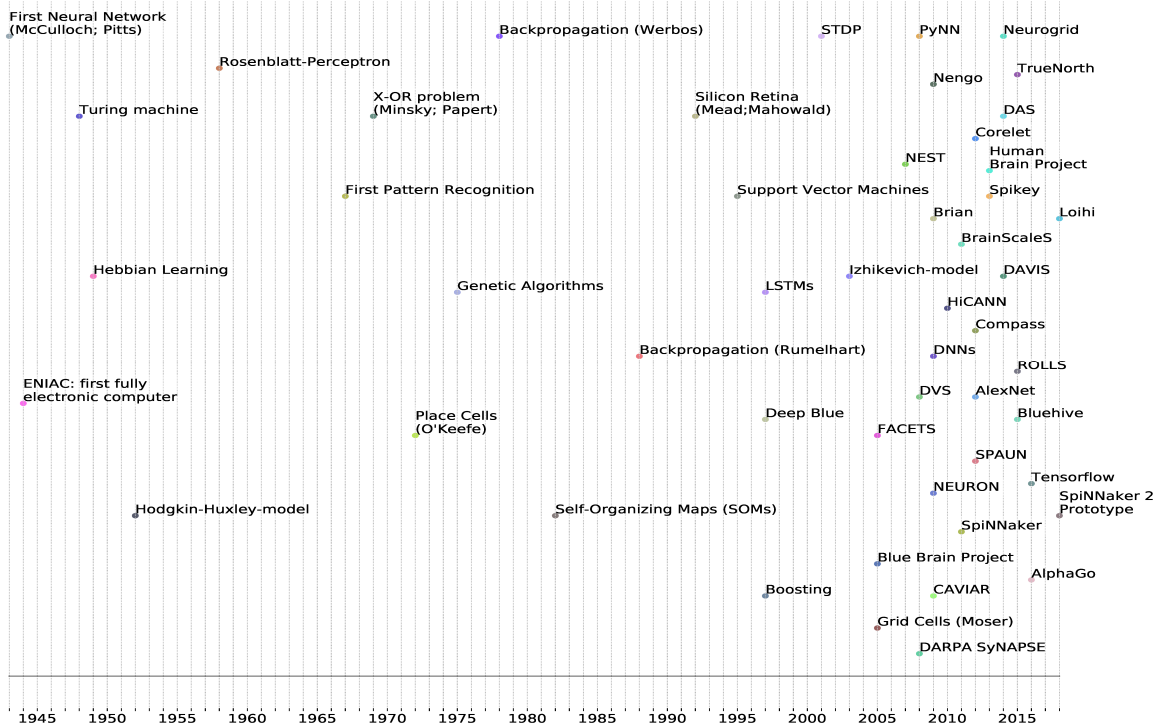


Figure 2.1: A selection of historical milestones in artificial intelligence, neuromorphic engineering and computational neuroscience. There is a significant boost of research and technologies in recent years.

single-layer perceptrons are not able to calculate the XOR-function or, more generally, are only capable of learning linearly separable patterns. This caused a decreased interest in neural network research until the rediscovery of the backpropagation algorithm (Werbos, 1974) in the 1980s, when Rumelhart et al. (1986) introduced a practically feasible method to optimize the network weights using gradient descent, which led to a resurgence of neural network research. Since then, various different network architectures such as feed-forward, CNNs, RNNs (Recurrent Neural Networks), RBFs (Radial Basis Functions Networks), RBMs (Resctricted Boltzmann Machines), SOMs (Self-Organizing Maps) and ART (Adaptive Resonance Theory) just to name a few (Schmidhuber, 2015) have been proposed for different learning paradigms. Although several simpler methods such as Boosting (Freund and Schapire, 1997) or SVMs (Support Vector Machines) (Vapnik, 1995) have been developed and achieved noteworthy results, the availability of powerful, parallel computing hardware like GPUs as well as the advent and success of deep learning (partly achieving better-than-human accuracy) made ANNs (Artificial Neural Networks) (Rojas, 1996) and especially DNNs (LeCun et al., 2015) the state-of-the-art for several machine learning tasks like visual digit (Ciresan et al., 2012b) and traffic sign (Ciresan et al., 2012a) recognition in recent years. Another great achievement in the field of deep learning was the victory of AlphaGo (Silver et al., 2016) over the world's best Go player Lee Sedol in March 2016, which was considered to be at least a decade away due to the complexity of Go. Compared to Deep Blue, the system that beat former chess world champion Garri Kasparov in 1997 (Hsu, 2002) with sheer computational power by brute forcing through a large number of possible moves in advance to find the best one, this strategy is not feasible for Go due to its higher complexity (larger board, more options to consider per move). In contrast, modern DNNs trained by a combination of supervised learning from human expert games and reinforcement learning from self-play have been used for the evaluation of board positions and selection of moves to avoid expensive look-ahead search (Silver et al., 2016). A comprehensive and historical overview of relevant literature concerning ANNs and especially DNNs can be found in Schmidhuber (2015) and LeCun et al. (2015).

Neuromorphic systems

The term neuromorphic itself was first introduced by Mead (1990), when describing one of the first silicon retinas. He called artificial systems that share organization principles with biological nervous systems neuromorphic. An interesting prototype of a silicon retina, which is now considered a milestone, was implemented by Misha Mahowald, a PhD student of Carver Mead. Her thesis received Caltech's Milton and Francis Clauser Doctoral Prize for its originality and potential for opening up new avenues of human thought and endeavor.

Since these early days of neuromorphic engineering, the term has widely been used to describe VLSI (Very-Large-Scale Integration) systems (Mead, 1989), novel computing devices (Schemmel et al., 2010), sensory systems (Lichtsteiner et al., 2008; S.-C. Liu and Delbruck, 2010), software (Davison et al., 2008; Bekolay et al., 2014) and algorithms (Reverter Valeiras et al., 2016). Considering the number of scientists, neuromorphic engineering is still a comparably young field of research but received an increased interest during the last decade from both academic and industrial research groups caused by the funding of large, ambitious projects. Although there have been several achievements in the field during the 1990s (Mead, 1989; Mahowald, 1992; Indiveri, 1997; Cauwenberghs, 1998) and early 2000s (S.-C. Liu et al., 2002), the FACETS (Fast Analog Computing with Emergent Transient States) project[1] and the BBP (Blue Brain Project)[2], both starting in 2005 and mainly funded by the EU (European Union) under the FP6-FET (Future Emerging Technologies) program, were among the first big-budget neuromorphic projects. The follow-up project BrainScaleS (Brain-inspired multiscale computation in neuromorphic hybrid systems)[3] (Schemmel et al., 2010), which was conducted from 2011 until 2015 built on and extended the research conducted during the FACETS project. The main developments of the FACETS and BrainScaleS projects are the HICANN (High Input Count Analog Neural Network) chip (Schemmel et al., 2010) and the Python-based simulator-independent language PyNN (Python Neural Networks) (Davison et al., 2008) for building neural network models. Building on the Blue Brain Project, the BrainScaleS hardware development is currently continued in the neuromorphic computing platform of the HBP (Human Brain Project)[4], a large ten-year research project, which was selected as one of the two EU-FET flagships in 2013 and is granted around one billion euros funding (Calimera et al., 2013). Another project starting in 2005, initially funded by the UK government until 2014 and now also part of the neuromorphic computing platform of HBP, is the SpiNNaker project (Furber et al., 2014) during which the neuromorphic computing hardware of the same name was developed. The HBP is organized in thirteen platforms in total, which focus on different research fields related to the brain like for example theoretical neuroscience, neurorobotics, cognitive architectures, high performance computing, brain simulation and the aforementioned neuromorphic computing platform.

Beside these research activities in Europe, the DARPA funded another big-budget neuromorphic project: the SyNAPSE (Systems of Neuromorphic Adaptive Plastic Scalable Electronics) program[5] (Srinivasa and Cruz-Albrecht, 2012), which started in 2008 and ran until 2016, has received 102.6 million US dollars in funding as of January 2013. The program aims to build an electronic microprocessor system that matches a mammalian brain in function, size, and power consumption. Achievements during the SyNAPSE program, which is primarily contracted to IBM Research and HRL, so far are include brain simulations, design of brain-inspired neuromorphic architectures (Nere et al., 2012) and the development of a digital neuro-synaptic core (Merolla et al., 2011), which is a building block of IBM's recently pub-

[1] Kirchhoff-Institute for Physics, Heidelberg University (2018b). *FACETS-project*. URL: https://facets.kip.uni-heidelberg.de (visited on 2018-04-05).

[2] École Polytechnique Fédérale de Lausanne‘ (2017). *Blue Brain project webpage*. URL: http://bluebrain.epfl.ch (visited on 2017-12-20).

[3] Kirchhoff-Institute for Physics, Heidelberg University (2018a). *BrainScaleS-project*. URL: https://brainscales.kip.uni-heidelberg.de (visited on 2018-04-05).

[4] Human Brain Project (2018). *Human Brain Project webpage*. URL: https://www.humanbrainproject.eu (visited on 2018-04-05).

[5] DARPA (2017). *DARPA SYNAPSE webpage*. URL: https://www.darpa.mil/program/systems-of-neuromorphic-adaptive-plastic-scalable-electronics (visited on 2017-12-20).

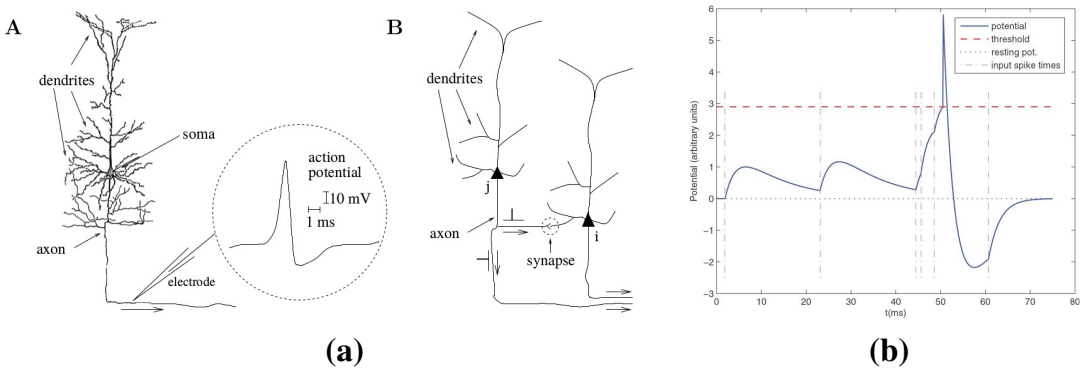


Figure 2.2: Visualization of different aspects of neuron models. (a) depicts the structure and functioning of biological neurons in a schematic visualization. Image source: Gerstner and Kistler (2002). (b) visualizes the membrane potential's sub-threshold behavior of a LIF neuron model. Image source: Masquelier et al. (2007).

lished TrueNorth chip (Akopyan et al., 2015). Further project results are the Corelet language (Amir et al., 2013) and the simulator Compass (Preissl et al., 2012), which enable dedicated software development as well as simulation and testing of TrueNorth algorithms on standard hardware respectively.
Beside these projects, the neuromorphic community is coming together at two annual workshops in Telluride and CapoCaccia, which have been established in 1994 and 2007 respectively, to discuss the current state of research in lectures and interactive talk sessions, to forge new ideas and to work on hands-on projects in small work groups.

2.1.2 Spiking Neural Networks

Figure 2.2a depicts the structure and functioning principles of biological neurons. They exchange information by sending short and sudden pulses, so-called action potentials or spikes, via synaptic connections. Whenever the membrane potential of a neuron, which can be increased or decreased by incoming spikes depending on the synaptic weight, reaches a certain threshold, the neuron produces a spike itself and resets its membrane potential afterwards (Gerstner and Kistler, 2002; Paugam-Moisy and S. Bohte, 2009). Recent neuroscientific research suggests that the exact timing of those spikes encodes information rather than just average firing rates (S. M. Bohte, 2004). While traditional ANNs used in machine learning neglect these biological details, SNNs embody these spike times and are therefore often referred to as the third generation of neural networks (Maass, 1997; Paugam-Moisy and S. Bohte, 2009). Maass (1997) showed, that SNNs have at least the same computational power as threshold and sigmoidal neural networks of similar size.
The simplest spiking neuron model is the LIF (Leaky-Integrate-and-Fire) model with

$$\frac{\partial V}{\partial t}(t) = -\frac{1}{\tau_m}\left(V(t) - R \cdot I(t)\right) \tag{2.1}$$

describing the sub-threshold behavior of the neuron, where V is the voltage across the membrane, $I(t)$ is the input current, R is the passive membrane resistance and τ_m is the membrane time constant. In other words, equation (2.1) states as follows: the membrane voltage increases in the presence of input current $I(t)$ depending on the membrane resistance R while at the same time, especially in the absence of input current ($I(t) = 0$), the voltage decreases or "leaks out" depending on the membrane time constant τ_m. When the voltage $V(t)$ passes a certain threshold ϑ, the neuron produces a spike and the voltage is reset to a resting state c for a certain refractory time interval τ_{ref} during which incoming spikes have no impact on the membrane potential. Figure 2.2b visualizes the spiking behavior of the LIF model. It shows an example curve of the membrane potential of one LIF neuron based on six incoming spikes whereas

only the third, fourth and fifth incoming spike are appearing closely enough for the membrane potential to surpass its threshold and, therefore, cause the neuron to emit a spike itself. The LIF model, despite its biological simplifications, is maybe the most widely used neuron model for simulations due to its simplicity and comparably low computational complexity (Izhikevich, 2004), which allows simulations of large networks of neurons in reasonable time. In contrast, the famous Hodgkin and Huxley (1952) model with its four differential equations and dozens of (biologically meaningful) parameters is a model of high biological plausibility but also computationally challenging regarding large simulations (Izhikevich, 2004). Izhikevich (2003) proposed a neuron model as compromise between biological plausibility and computational feasibility. In Izhikevich (2004), he showed that his simple model, described by two differential equations with four parameters, is able to produce all known spiking behaviors observed in cortical neurons.

One major hindrance for the widespread adoption of SNNs has been the problem, that standard learning algorithms for traditional ANNs like backpropagation (Werbos, 1974) can not be directly applied to SNNs. Although an analogon, the so-called SpikeProp algorithm (S. M. Bohte et al., 2002) for SNNs has been developed, the more natural approach is to transfer and mimic biologically inspired learning approaches like Hebbian learning (Hebb, 1949) or STDP (Spike Timing Dependant Plasticity) (Bi and Poo, 2001). An overview of several learning approaches for SNNs possibly applied with neuromorphic hardware can be found in Walter et al. (2015). Another possibility is to train a traditional ANN and convert the resulting network into a SNN as demonstrated in Diehl et al. (2015) and Hunsberger and Eliasmith (2015). An example for this approach is the network performing the visual digit recognition task as part of the larger Spaun (Semantic Pointer Architecture Unified Network) model (Eliasmith et al., 2012), which was derived by training a DNN consisting of four RBM layers and converting this network using the principles of the NEF (Eliasmith and C. H. Anderson, 2003). Although theoretically superior (Maass, 1997), SNNs have not yet outperformed state-of-the-art DNNs in terms of accuracy in practical machine learning applications (Schmidhuber, 2015).

Beside the aforementioned procedures to solve traditional machine learning tasks with SNNs and thereby encode artificial functions in spiking neurons, a different approach is to try to understand how complex cognitive behaviors and the underlying neural functions are performed in the brain. Therefore, the question how the brain encodes complex information and behavior in trains of spikes and also how to decode these spike trains to reconstruct the encoded information needs to be answered. Although modern research has shed some light on this question regarding the neural code, it is still mainly unanswered since we do not fully understand the anatomical and neurophysiological processes within the brain (Stanley, 2013). Currently, there exist several approaches to code information as spike trains, which can be summarized by the categories rate coding, temporal coding (Gerstner et al., 2014, Chap. 7.6), population coding (Gerstner and Kistler, 2002, Chap. 1; Ponulak and Kasinski, 2011; Boerlin and Denève, 2011) or sparse coding (Olshausen and Field, 1996). Except for the biologically unrealistic rate coding approach, there are cues for all of these coding schemes and even combinations (Gupta and Stopfer, 2014) of them to appear in biological systems.

Software tools

There exist several different programming languages, simulators and software libraries specifically designed for modeling SNNs varying from tools like Corelet (Amir et al., 2013) and Compass (Preissl et al., 2012) working with one specific hardware component (see also Sec. 2.1.3), in this case IBM's TrueNorth chip (Akopyan et al., 2015), to libraries like PyNN (Davison et al., 2008), which aim for universality to work with different simulators and hardware components as back-end.

The simulation tool offering maybe the highest level of abstraction is Nengo (T. C. Stewart et al., 2009; Bekolay et al., 2014), which implements the principles of the NEF (Eliasmith and C. H. Anderson, 2003) and was used to build the Spaun model presented in Eliasmith et al. (2012). Originally written in Java, Nengo was re-implemented in Python (Bekolay et al., 2014) and improved by incorporating the lessons learned from the creation of Spaun. Nengo allows the user to describe a model on a high level of

abstraction by defining groups of neurons to simulate different functional blocks while Nengo takes care of neural properties and synaptic weights using the NEF (see Sec. 3.3 for further details). Nengo models can be run using the internal simulation back-end, but also simulation on some neuromorphic hardware components such as Neurogrid (Dethier et al., 2011; Choudhary et al., 2012), Braindrop (Neckar et al., 2019) or SpiNNaker (Mundy et al., 2015), which are currently used in developments aiming to run the Spaun model in real-time, is supported.
Another Python-based tool is PyNN (S. Davies et al., 2010), which was developed during the FACETS[6] and BrainScaleS[7] projects and aims for building SNN models independent of actual simulation tools. The level of abstraction is lower than Nengo, but therefore it allows the creation of arbitrary neuron populations and connections, while the properties and synaptic weights need to be specified by the user or acquired using a learning algorithm. PyNN implements a number of standard neuron models like LIF or Izhikevich, connection algorithms like *one-to-one*, *all-to-all* or connection matrices, static and plastic synapse types as well as several STDP rules supported by the simulation and hardware back-ends. Furthermore, PyNN enables the user to implement custom models, connections and learning rules for advanced simulations and thereby extend the neural modeling toolkit. PyNN currently supports several different simulators such as NEST (NEural Simulation Tool) (Gewaltig and Diesmann, 2007), Brian (Goodman and Brette, 2009) and NEURON (Carnevale and Hines, 2009) as well as neuromorphic hardware as back-ends. PyNN is currently the preferred development environment for the creation of SNN models to run on the SpiNNaker system (Furber et al., 2014). The mapping of the network structure, neurons and synapses to actual cores on the chip is done with a separate software package called PACMAN (Partitioning And Configuration MANager) (Galluppi et al., 2012).
Another Python-based software package which, in contrast to PyNN and Nengo, mainly aims at modularity and flexibility in terms of supporting as many different neuromorphic hardware systems as possible as a front-end is PyNCS (Python Neuromorphic Cognitive Systems) (Stefanini et al., 2014).
A comprehensive overview of several other simulation tools for neural modeling can be found in Brette et al. (2007).

2.1.3 Neuromorphic Hardware

In this section, we give a brief overview of recent neuromorphic prototypes and hardware developments. Although we do not integrate the models and approaches developed in this thesis with this kind of dedicated hardware platforms, deployment on specialized hardware components is an interesting future possibility promising increased energy-efficiency and scalability. The neuromorphic prototypes described here are still comparatively young and not technologically mature yet, especially compared to traditional computing systems. Therefore, most of the hardware and sensors described are mainly developed and used in academic research and are not standardized, commercial products yet. However, this technology is gradually becoming available to a broader community and draws increased attention in industrial research groups.

Digital Neurochips

GPUs provide a SIMD (single instruction multiple data) architecture, which processes data in parallel at the cost of increased power consumption (Krichmar et al., 2011; Carlson et al., 2014) compared to CPUs (Central Processing Units). Allowing fast matrix and vector multiplication, GPUs provide an appealing platform for training and executing deep learning techniques (Schmidhuber, 2015). For specific applications, which take several months of training on a traditional CPUs, GPU-based systems can significantly accelerate processing by several orders of magnitude to decrease the computation time to a number of

[6] Kirchhoff-Institute for Physics, Heidelberg University (2018b). *FACETS-project*. URL: https://facets.kip.uni-heidelberg.de (visited on 2018-04-05).
[7] Kirchhoff-Institute for Physics, Heidelberg University (2018a). *BrainScaleS-project*. URL: https://brainscales.kip.uni-heidelberg.de (visited on 2018-04-05).

days (Edwards, 2015). A collaboration between NVIDIA and Stanford researchers demonstrated a cluster of GPU servers, which was able to train a network with billion parameters (scalable to 11 billion using 16 machines) on just three computers in two days. Each machine contained 4 NVIDIA GTX680 GPUs with 4 GB at 1 TFLOPS (Floating Point Operations per Second) each.
One of the recent developments in digital neurochips is IBM's TrueNorth (Akopyan et al., 2015). It contains a network of 4096 cores with 256 digital neurons each following a digital, re-configurable spiking neuron model (Cassidy et al., 2013) yielding the capability to implement over one million neurons and 256 million synapses in total while consuming only 65 mW of power. The cores are connected internally by a two-dimensional grid. The intersections of this grid contain routers, which control the signal transmission within the network of cores inside a chip. For programming this novel hardware implementation, a specialized programming language named Corelet was developed (Amir et al., 2013).
Another example of a digital neuromorphic hardware implementation is University of Manchester's SpiNNaker system (Furber et al., 2014). It contains 18 synchronously connected ARM968 microprocessors and 128 MB DDR (Double Data Rate) SDRAM (synchronous dynamic random-access memory). Communication is carried out by a packet-switched on-chip network with all chips having their own router. This architecture scales well to larger application by allowing to place more chips on a board (Painkras et al., 2013; Navaridas et al., 2009). A SpiNNaker system forms a toroidal mesh, which, despite having fixed connections, allows the simulation of neural networks of arbitrary connectivity due to the protocol implemented by the routers on the individual chips.
Another recent digital neuromorphic hardware platform is Intel's Loihi chip (M. Davies et al., 2018). This chip consists of a fully asynchronous many-core mesh of 128 neuromorphic cores, each containing 1024 primitive units implementing spiking neuron behavior in a tree-like structure. In total, it contains 130 000 artificial spiking neurons and 130 million synapses. One of Loihi's key features is the ability of on-chip learning, which is realized through a learning engine integrated in each core that enables the implementation of learning rules to adapt parameters of the core's SNN.

Analog Neurochips

Most analog chips prohibit on-chip training due to limited adaptability of implementation techniques like capacitors (Schwartz, 1990), floating gate transistors (Holler et al., 1989) or CCDs (Charge Coupled Devices) (Agranat et al., 1990). On the other hand, analog techniques use less components and offer high-speed operation. To exploit the benefits of analog semiconductor technologies and neural adaptivity, the learning algorithms have to be implemented on-chip. However, on-chip-implementation limits the platform's re-configurability and complicates the implementation of most learning rules directly into analog VLSI at the same time. These limitations restrict the flexibility of analog designs compared to their digital counterparts.
The HICANN chip (Schemmel et al., 2008) has the capability to simulate 131 072 synapses and 512 neurons residing inside an ANC (Analog Network Core). To simulate large networks, a wafer-scale integration method was used, which allows 384 HICANN chips to be interconnected on a wafer of 20 cm diameter (Schemmel et al., 2010). The spikes between various ANCs on a single wafer and between several interconnected wafers are carried out by a network of horizontal and vertical grid-like structures. The horizontal pathway consists of 64 bus lines which carry spikes from 64 neurons. The 256 vertical lines collect spikes for all connected ANCs. The spikes are represented as data packets of 6 bit for groups of 64 neurons and transmitted using AER (Address Event Representation).
The ROLLS (Reconfigurable On-Line Learning Spiking) neuromorphic processor consists of 256 neurons and 128 000 synapses. It consumes only 4 mW of power and uses exponential IF (Integrate-and-Fire) modeling (Qiao et al., 2015). The latest processor contains different configuration options. Each neural circuit is connected to three different types of synaptic circuits: the first is an array of 256×256 synapses, which can be excitatory or inhibitory. The synapses in the second 256×256 array offers only excitatory mode. In the third array, there are 256×2 virtual synapses with weights being configurable as both, excitatory as well as inhibitory. The virtual synapses operate in shared mode and occupy less space

compared to individual circuits. The neural spikes are generated and transmitted using an AER protocol with 8 bit neuron addresses (Qiao et al., 2015).

Another example of analog neural hardware is a chip (Srinivasa and Cruz-Albrecht, 2012) developed under DARPA-funded HRL (Hughes Research Laboratories) SyNAPSE project, which uses STM (synaptic time multiplexing) for computation and memristors for synaptic weights storage. It contains a total of 576 neurons and 73 728 synapses, arranged in an array of 24×24 neurons each with 128 synapses and consumes 130 mW. In STM methodology, a single synaptic circuit on the chip calculates multiple logical synapses of the simulated neural network since the electrical circuitry can function at higher speeds than biological neurons. Hence, a single synaptic circuit allows emulating multiple logical synapses by switching between the corresponding parameter sets. There is a point-to-point routing mechanism avoiding the necessity of the AER protocol (Walter et al., 2015).

The Brain in Silicon group at Stanford University developed the custom board Neurogrid, which contains 16 Neurocores (Benjamin et al., 2014; Choudhary et al., 2012). The neuron circuits are arranged in a 256×256 grid with every neuron having separate circuits for its soma and dendrite and combining analog computations with digital communication. The system is capable of simulating a million neurons and eight billion synapses in real-time while consuming 3.1 W of power. Neurogrid employs the AER protocol to transmit spikes. It additionally implements a deadlock-free wormhole routing protocol to ensure free transmission of packets. Finally, Neurogrid allows to explore different cortical areas of the brain by programming each of the Neurocores with a different model (Merolla et al., 2014). Its successor BrainDrop (Neckar et al., 2019) is inspired by NeuroGrid's architecture and emerged from continued development on that architecture. It is designed as one of the first neuromorphic systems to be programmed at a high level of abstraction using the NEF (Eliasmith and C. H. Anderson, 2003).

Neuromorphic Sensors

To integrate neuromorphic hardware in real-world applications such as neurorobotics, it needs to be able to process sensors signals. The neurally inspired, non von-Neumannian architecture requires either a new generation of sensors such as the DVS (Lichtsteiner et al., 2008), DAVIS (Dynamic and Active Pixel Vision Sensor) (Brandli et al., 2014) or DAS (Dynamic Audio Sensor) (S.-C. Liu et al., 2014), which already embody the spiking neuron signal processing, or a way of translating traditional sensor signals to sequences of spikes. Several approaches to encode signals and stimuli with trains of spikes, e.g., by rate coding, temporal coding (Gerstner et al., 2014, Chap. 7.6) or population coding (Gerstner and Kistler, 2002, Chap. 1; Ponulak and Kasinski, 2011) have been investigated. Neuromorphic sensors on the other hand directly emit a sequence of events without the need of such a translation process. The DVS for example, unlike traditional frame-based cameras, generates spike events (Lichtsteiner et al., 2008) asynchronously for individual pixels when perceiving relative illumination changes (see Fig. 2.3), which are key features of biological vision. This event-based approach offers several advantages compared to conventional frame-based cameras like the ability to perceive very fast movements (sub-millisecond time precision) without the need to wait for the next frame and the time to process it. Furthermore, the rate of the output data depends on the dynamic content of the scene (Fig. 2.3a) and thus reduces the output of redundant information at a fixed frame-rate.

The DVS-pixels were designed to cover wide dynamic range and to offer low latency and mismatch. This is achieved by a fast photoreceptor circuit with logarithmic response whose output is fed into a high precision difference amplifier circuit followed by a two-transistor comparator circuit (see Fig. 2.3b and 2.3c). The comparator circuit produces ON and OFF signals for each pixel, which are passed to the AER interface for transmission (Lichtsteiner et al., 2008). On the other hand, the DAVIS employs an APS (Active Pixel Sensor) circuit to output synchronous global shutter frames, which helps in persevering absolute intensity information needed for classification and object recognition tasks while asynchronous DVS events are beneficial for the tracking of fast moving objects (Brandli et al., 2014).

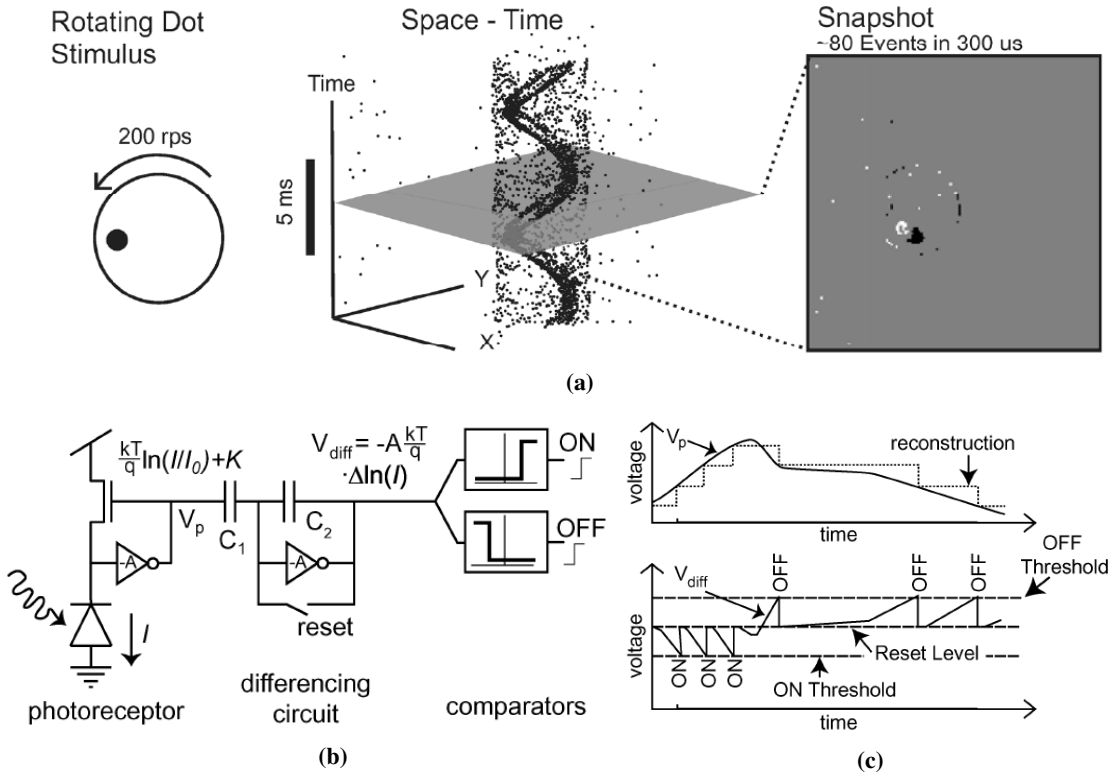


Figure 2.3: (a) Space-time representation of the event stream generated by a rotating dot on a spinning disk and a snapshot of the events (b) abstracted pixel core schematic (c) principle operation of a single DVS pixel. Image source: Lichtsteiner et al. (2008)

2.1.4 Neuromorphic Applications

In this section we present some examples of applications of neuromorphic hardware and sensors described in Sec. 2.1.3, software, algorithms and neural modeling depicted in Sec. 2.1.2 as well as combinations of both. First, we describe applications using either neuromorphic sensors or hardware in combination with traditional computing hardware. Then, we focus on purely neuromorphic systems, where the spiking signals of neuromorphic sensors, mainly the DVS, is processed by neuromorphic hardware. Although there are currently - to the knowledge of the author - no neuromorphic actuators working directly on the basis of spikes, we refer to the robotic systems described here as purely neuromorphic.

Mixed systems

Neuromorphic vision (Tan et al., 2015; Gallego et al., 2019) is an emerging field of research, which aims to transfer approaches from traditional computer vision and also establish new methods incorporating the characteristics of the DVS. To perform pattern recognition tasks with neuromorphic cameras and at the same time use traditional methods as a benchmark, existing image data bases like the MNIST (Mixed National Institute of Standards and Technology) data set (LeCun et al., 1998) are translated to neuromorphic data sets by presenting the images on a computer screen to a DVS sensor moving minimally back and forth (Orchard et al., 2015) (mimicking saccade movements of the human eye), which gives better results than moving the images themselves on the screen (Serrano-Gotarredona and Linares-Barranco, 2013).

As the DVS naturally captures moving objects when held statically and cues of the ego-motion when moving, tracking of these movements are suitable applications making use of the sensor's characteristics. These characteristics along with the DVS's advantages compared to traditional frame-based cam-

eras, which have been described in Sec. 2.1.3, have been demonstrated in applications such as pencil balancing (Conradt et al., 2009) and a robotic goalie (Delbruck and Lang, 2013) interfacing the DVS with traditional computing hardware and actuators. Keeping track of (geometric) features (Lagorce et al., 2015) and contours (Barranco et al., 2014) observed by the DVS lays the foundation for more sophisticated algorithms. Researchers recently proposed several approaches for tracking people (Schraml et al., 2010; Piatkowska et al., 2012) and other geometric objects (Reverter Valeiras et al., 2016) using stationary sensors. Especially tracking at high velocities (Saner et al., 2014) benefits largely from the sub-millisecond time precision of the DVS even enabling tracking of particles in fluid flows (Drazen et al., 2011), which would require a high-speed PC, lots of disk space and high-intensity laser strobe lighting to illuminate the fluid in a conventional setting.

A traditional application of computer vision in robotics is the estimation of the ego-motion or odometry based on optical flow. An event-based approach to optical flow can be found in Benosman et al. (2014), which can also be obtained by combining traditional cameras with the DVS on a small wheeled robot (Censi and Scaramuzza, 2014). Another suitable application for the DVS is the estimation and tracking of the whole six degree-of-freedom pose of a flying robot, especially when performing high-speed maneuvers, where traditional cameras suffer from motion blur (Mueggler et al., 2014). Another problem in robotics closely related to tracking is self-localization of the robot using a given map of the environment or building a map online and localizing within this map at the same time, which is known as the SLAM (Simultaneous Localization and Mapping) problem (Thrun et al., 2005). Localization performed within a given map using the DVS on ground vehicles and by tracking markers from a flying robot is presented in Gallego et al. (2015) and Censi et al. (2013) respectively. There are also several papers such as Weikersdorfer and Conradt (2012) and Weikersdorfer et al. (2014) treating the SLAM problem or the related problem of tracking the camera pose and simultaneously reconstructing the observed scene by mosaicing (Kim et al., 2014) with neuromorphic vision sensors.

Most of the tracking algorithms mentioned here use variations of traditional Bayesian filters like (extended) Kalman- or particle-filters (Thrun et al., 2005) for consecutive estimation of the quantity of interest. However, due to the asynchronous information processing of the DVS or neuromorphic systems in general, these filters need some modifications to work with this event-based approach (Weikersdorfer et al., 2013) like taking several past measurements into account (in contrast to the traditional Markov assumption) or updating only after a certain number of events have occurred to avoid computational overhead.

Axenie and Conradt (2015) describe a biologically inspired system for sensor-fusion of several traditional sensors like wheel encoders, magnetometer and gyroscope for ego-motion estimation demonstrated in ground and flying robots. Another example for sensor fusion is presented in O'Connor et al. (2013), where the biologically inspired sensors DVS and DAS are used to recognize hand-written digits from the MNIST data set (LeCun et al., 1998). To incorporate the neuromorphic audio sensor, each digit is assigned one tone frequency in the A harmonic minor scale, while the actual fusion is performed by a DBN (Deep Belief Network) consisting of several, pre-trained (unsupervised) RBM layers, which was traditionally trained and afterwards transferred to an event-based network. Although the authors state that the actual implementation in O'Connor et al. (2013) is just a proof-of-concept in software, their approach shows promise to translate fully trained DBNs to SNNs, deploy them on efficient neuromorphic chips and thereby making this technology available for mobile and/or real-time applications.

To demonstrate the functionality and applicability of their neuromorphic chip TrueNorth, IBM implemented several proof-of-concept applications ranging from virtual robots, game simulations (Arthur et al., 2012), digit recognition (Arthur et al., 2012; Esser et al., 2013) to classical machine learning tasks like object detection (Akopyan et al., 2015). Arthur et al. (2012) present four example applications of neural algorithms implemented on TrueNorth using the Corelet language: a virtual robot driver aiming to keep the simulated robot on a virtual road based on visual cues, a neural algorithm controlling an autonomous player performing the classical video game Pong, a neural implementation recognizing hand-written digits of the MNIST data-base using RBMs as well as a Hopfield network, which performs auto-association.

Seven example algorithms and applications for TrueNorth are presented in Esser et al. (2013): speaker recognition using CNNs on the CUAVE (Clemson University Audio Visual Experiments) data set (Patterson et al., 2002), composer recognition distinguishing between classical composers Bach and Beethoven using liquid state machines, recognition of hand-written digits of the MNIST data set using population coding and RBMs, a neural implementation of a HMM (Hidden Markov Model), collision avoidance using motion extraction and looming detection, optical flow based on CNNs and eye detection using the DVS. Most of these algorithms are simplified proof-of-concept implementations showing the general applicability of TrueNorth but are not competitive with traditional machine learning algorithms yet, that is, achieving only 92.34 % in Esser et al. (2013) compared to 99.77 % in Ciresan et al. (2012b) correct classifications on MNIST. Schmitt et al. (2017) demonstrate training and deployment of DNNs on the MNIST-data set using the BrainScaleS hardware. A more sophisticated application is the multi object detection and classification task described in Akopyan et al. (2015), which detects moving and stationary people, bicyclists, cars, buses and trucks from a HD video stream in real-time recorded by a stationary camera using Haar-like features, saccades, K-means- and Grid-classifiers.
The use of the neuromorphic hardware Neurogrid (Benjamin et al., 2014) as computational back-end for Nengo and proof-of-concept solution for medical motor-prostheses, which shall be connected to biological neurons and thereby be controlled by the patient's brain, is described in Choudhary et al. (2012) and Dethier et al. (2011) respectively. The SpiNNaker system (Furber et al., 2014) also supports the simulation of neural models created using PyNN or Nengo (Mundy et al., 2015). Furthermore, several neural network architectures like CNNs (Serrano-Gotarredona et al., 2015) and pre-trained DBNs (Stromatias et al., 2015a; Stromatias et al., 2015b) have also been implemented on the SpiNNaker system. To enable biologically inspired learning like STDP (Bi and Poo, 2001), according rules adapting the synaptic weights depending on the timing of the spikes have efficiently been implemented on SpiNNaker in Diehl and Cook (2014) as well.

Purely neuromorphic systems

Some examples of closed-loop systems deployed in small robots are presented in S. Davies et al. (2010), Denk et al. (2013), and Galluppi et al. (2014). S. Davies et al. (2010) and Denk et al. (2013) interface two embedded DVS cameras directly with a SpiNNaker platform mounted on a small robot (see Fig. 2.4). The implemented neural networks enable simple autonomous behaviors like following a line (S. Davies et al., 2010) or approaching a light stimulus (Denk et al., 2013). In Galluppi et al. (2014), a small robot with a similar setup is able to perform trajectory stabilization using optical flow from the DVS cameras as input. Another simple task described in Galluppi et al. (2014) is the recognition and tracking of a light stimulus and keeping the robot at a certain distance and angular orientation with regard to this stimulus. The whole processing chain from visual perception over internal processing to motor control in these robot experiments are realized using spiking neuron models. The neural implementation of the latter two applications are done in PyNN and Nengo respectively.
While the aforementioned sensorimotor behaviors are manually engineered, Conradt et al. (2015) and T. C. Stewart et al. (2016) present attempts to learn more sophisticated, complex behaviors from simpler basic movements. These basic maneuvers are still manually engineered relating sensor cues to simple movements like driving forward with no obstacle in the sensors field of view, turning with an obstacle in front of the robot or driving backwards when being close to an obstacle. Conradt et al. (2015) describe a method on learning more sophisticated behaviors from recorded sensorimotor data obtained from driving the robot by remote control as training examples, which can be considered as a supervised learning approach. In T. C. Stewart et al. (2016), the training examples are taken from recording data of the robot driving around without human interference and just labeling those situations as positive examples when the robot performed the desired action by accident, which is considered as reinforcement learning. Both approaches are implemented on a small robot with the DVS as sensory input using Nengo and the NEF as well as its interface (Mundy et al., 2015) for running neural networks models on the SpiNNaker hardware (Furber et al., 2014).

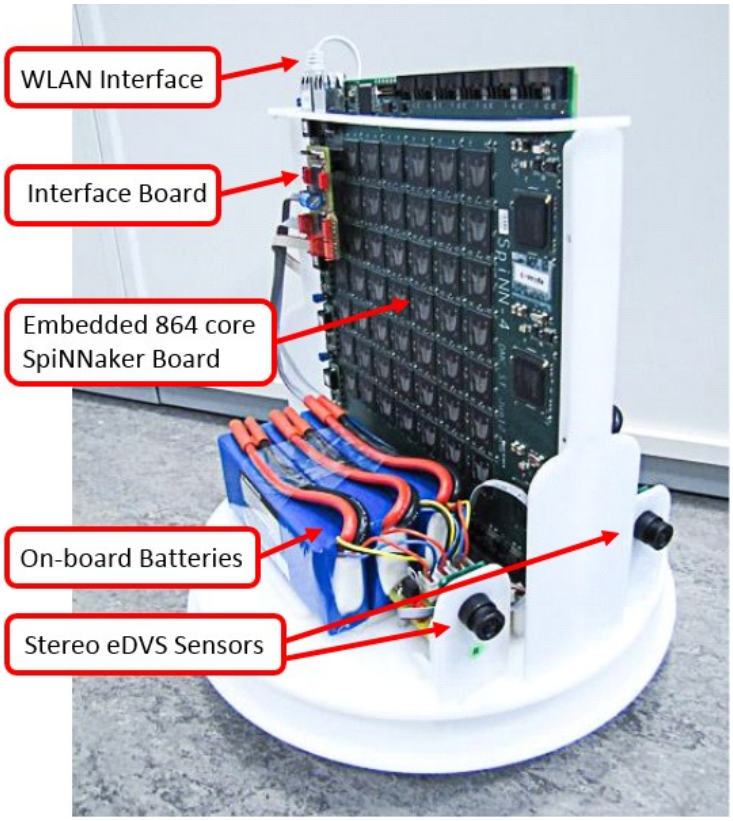


Figure 2.4: Example of a closed-loop, neuromorphic robotic system with two event-based embedded DVSs and a 48-node SpiNNaker board. Image source: Galluppi et al. (2014)

2.2 Cognitive Modeling

Understanding and building cognitive systems has seen extensive research over the last decades leading to the development of several cognitive architectures. A cognitive architecture is a "general proposal about the representation and processes that produce intelligent thought" (Thagard, 2012). On the one hand, these architectures are used to explain and better understand important aspects of human behavior and intelligence. On the other hand, they are also used to design computers and robots mimicking certain cognitive abilities of humans. In this respect, cognitive architectures are typically clustered in three main categories, namely *symbolism*, *connectionism* and *dynamicism* (Eliasmith, 2013). We will give a brief overview over symbolic and connectionist approaches in subsequent sections, whereas dynamicism (Schöner, 2008) is of lower relevance to the work at hand.

One important aspect of cognitive modeling and, more generally, AI, is knowledge representation. Any intelligent agent, artificial or biological, that wants to perform reasoning about the world it encounters, needs to be able to build an internal representation of its perceived information. This aspect is quite important for subsequent chapters, while a formal definition of what knowledge representation actually is appears to be difficult and thus is often avoided in the literature (Davis et al., 1993). Davis et al. (1993) describe five defining roles that such a representation can play. A representation is a surrogate, i.e., an internal substitute for a real-world entity and as such an imperfect approximation. Therefore, the choice for each representation implies a set of ontological commitments, which effect the focus of attention of the representation. When the focus of such a representation is to enable some kind of reasoning in intelligent machines or robotic systems, these systems need to be able to manipulate the representation and perform computations with it. Finally, as long as the machine needs to interact or communicate with humans in the sense that humans inform the machine about the world by, e.g., creating a representation, this representation itself also plays the role of a medium of human expression. Therefore, we put empha-

sis on the different aspects of knowledge representation in the different cognitive modeling architectures presented in this section.

2.2.1 Symbolic approaches

Symbolic approaches are often referred to as the *classical approach* to cognitive modeling or GOFAI (Good Old-Fashioned Artificial Intelligence). Most of the approaches rely on the metaphor of the *mind as computer*, supposing that cognitive systems have a symbolic "language of thought" (Fodor, 1975), that expresses the rationale and rules the systems follow. The corresponding analogue for computers are programming languages. The dominant paradigm of such approaches is "the manipulation of discrete atomic symbols by explicit rules" (S. D. Levy and Gayler, 2008). The most prominent approaches in this category are production systems, which typically rely on if-then-rules (or productions) and a control structure. One of the first and most influential achievements in this field is a program called the GPS (General Problem Solver), which was able to solve elementary problems in symbolic logic on its own. The steps GPS performed to solve a given problem often matched the steps reported by people solving the same problem. This success enabled the development of several other cognitive architectures such as Soar (Laird et al., 1987), EPIC (Executive-Process/Interactive Control) (Kieras and Meyer, 1997) and ACT (Adaptive Control of Thought) (J. R. Anderson, 1983) and its successor ACT-R (Adaptive Control of Thought-Rational) (J. R. Anderson, 1996), which all employed production systems at their core but adding their own extensions. ACT-R is the most modern of these architectures and arguably the most successful and thus it is the most widely used cognitive architecture. Although ACT-R incorporates some connectionist-like mechanism in its memory system, it is widely considered a symbolic architecture as it relies on symbolic representations and a production system as central procedural core. In general, symbolic approaches to cognitive modeling had the most success when addressing higher-level cognitive tasks, with a rigid set of rules and potential for pre-specified solutions. However, these approaches are rarely used when it comes to real-time critic systems such as robots or if the system needs to generalize beyond pre-specified situations.

2.2.2 Connectionist approaches

One of the first theories challenging the paradigm of GOFAI (Good Old-Fashioned Artificial Intelligence) was the "Society of Mind" view of specialized individual agents cooperating to accomplish a certain goal proposed by Minsky (1986). The strongest challenge however, was the emergence of connectionism (Rumelhart and McClelland, 1986), popularly referred to as neural networks, which offered novel learning algorithms such as backpropagation (Rumelhart et al., 1986) to solve a wide variety of problems. Connectionism explains cognitive phenomena by constructing models consisting of large networks of interconnected nodes performing rather simple input/output mappings. If these nodes, however, are connected to sufficiently large networks, the nodes' activity is able to implement cognitive behavior such as rules or analyzing patterns. We already gave a brief historical overview over the research conducted related to ANNs in section 2.1.1. The metaphor behind connectionist approaches to cognitive modeling is that of the *mind as brain*, as the processing employed in connectionism is often referred to as *brain-like* or *brain-inspired*. Connectionism has shown remarkable results in diverse applications such as computer vision, pattern recognition, sequential data analysis and language processing just to name a few. However, the most serious criticism of connectionist approaches are that they could not exploit systematic, compositional representations or logical reasoning of the form used in GOFAI. Furthermore, the nodes in connectionist networks typically simplify the computational and representational properties of biological neurons, which, in addition to the biological implausibility of the backpropagation algorithm, concerns cognitive modelers interested in biological realism. Finally, the ability to learn and derive internal representations of features from data, despite being one of the greatest strengths of neural networks approaches, is sometimes criticized for lack of comprehensibility.

2.2.3 Vector-based approaches

To address some of the concerns regarding both, symbolic and connectionist approaches to cognitive modeling, researchers developed a hybrid approach often referred to as VSA (Vector Symbolic Architecture), a class of connectionist distributed representations. In chapter 3, we give an in-depth introduction to the theory and mathematical properties of VSAs, which will be important ingredients for the remainder of this thesis. Here, we give a brief overview of the different variants of such architectures, their similarities and differences, related work and some applications. All of the modeling approaches presented in this section employ (high-dimensional) vectors as representational substrate. Similar to the representations derived from connectionist approaches, these are distributed representations, which offer nice properties such as robustness to noise and the support of distance metrics. Additionally, all of the approaches allow to treat such vectors as *symbol-like* entities, which can be manipulated through the architecture's algebraic operations (see chapter 3 for details). The first attempt on structured vector representations proposed by Smolensky (1990) used the tensor product as multiplication operation to bind two different vectors together. The tensor product approach already allows for a sufficiently complex embedded structure to do linguistic processing. However, scaling becomes a problematic issue as each uncompressed tensor product operation of two high-dimensional vectors increases the result's dimension, which quickly becomes impracticable. Hence, there are several architectures such as MAP (Multiply-Add-Permutate) (Gayler, 1998; Gayler, 2003), BSCs (Binary Spatter Codes) (Kanerva, 1988) and HRRs (Holographic Reduced Representations) (T. Plate, 1991; T. Plate, 1994), which propose different compressed multiplication operations replacing the tensor product and resulting in vectors with the same dimension as the input vectors. Furthermore, these different VSA variants differ in the choice of the numerical space to pick the vectors' elements from, that is, using binary, real- or complex-valued vectors. The SPA (Eliasmith, 2013) is built upon HRRs and extends these architectures, by proposing an efficient mechanism of implementing structured vector representations in populations of spiking neurons using the principles of the NEF (Eliasmith and C. H. Anderson, 2003) (again, we refer to chapter 3 and especially sections 3.3 and 3.4.3 for further details). This architecture has been used in Eliasmith et al. (2012) to built the currently largest functional model of a brain using a combination of structured vector representations for symbol-like processing and a spiking neuron substrate.

The most prominent application of vector representations despite cognitive modeling (Blouw et al., 2016; Crawford et al., 2016; Eliasmith et al., 2012) is language processing (Gayler, 2003). In this context, word embedding refers to the problem of finding (or automatically learning) desirably meaningful representations for words. Modern word embedding algorithms such as word2vec (Mikolov et al., 2013b; Mikolov et al., 2013c) or GloVe (Global Vectors) (Pennington et al., 2014) employ high-dimensional vectors as representational structure to encode words and language by learning in unsupervised fashion from large corpora of text. There are also attempts of using such representations in other domains to, e.g., better explain and quantify how DNNs learn and derive concepts (Fong and Vedaldi, 2018) or for embedding low-level vehicle sensor data in an abstract representation (Hallac et al., 2018).

Another approach employing vector representations is the work of companies such as Numenta (2019) and Cortical.io (2019). They employ binary vector representations similar to BSCs, which Ahmad and Hawkins (2015) refer to as SDRs (Sparse Distributed Representations), as the basis and main representation for their downstream cortical models such as HTM (Hierarchical Temporal Memory) proposed in Cui et al. (2017). To create high-dimensional vectors representing words or phrases, a method called semantic folding is applied (Webber, 2016). Similar to classical word embedding algorithms, the word vectors are created from large corpora of text. After laying out a two-dimensional semantic map of available contexts, the context of each particular word is marked as active (resulting in a 1 bit in the map) and a sparse, high-dimensional binary vector is created from the map through serialization. Despite language processing, such representations in cooperation with HTM models have shown to be useful for applications such as anomaly detection (Ahmad et al., 2017) or classification with noisy data (Ahmad and Scheinkman, 2019). One key difference to other cognitive architectures like the SPA is that the entries of the SDR vectors are directly interpreted as neural activity whereas the SPA distinguishes between

representational and neuronal space.

2.3 Automated Driving

"Robotics is the science of building computer-controlled mechanical devices, which are able to perceive and manipulate the physical world" (Thrun et al., 2005). Automated driving in automotive context is a special case of robotics, since an autonomous vehicle can be considered a wheeled mobile robot, which is able to fulfill the transportation capabilities of a traditional car without human input. In order to navigate safely to a desired goal, a mobile robot needs to solve several problems like localization ("where am I?"), path planning ("how do I get from A to B?"), environment perception ("where is everyone else?"), knowledge representation and reasoning ("which decisions to infer from available information?") as well as motion control ("how to move my actuators?"). In automotive context, an automated vehicle furthermore needs to detect the current state of the driver ("what is the driver up to?") to ensure that he can take over control in safety-critical situations or in case of malfunctions. The human driver as a fallback option in such situations is of crucial importance, since the level of driving automation is likely to gradually increase instead of a hard transition from manual driving to fully automated driving systems. In their J3016 standard[8], the SAE (Society of Automotive Engineers) delivers a classification system identifying six different levels of driving automation from "no automation" to "full automation". Table 2.1 gives an overview of the particular automation levels according to the SAE (2016) in more detail.

Level	Name	Narrative Definition
0	No Automation	the full-time performance by the human driver of all aspects of the dynamic driving task, even when enhanced by warning or intervention systems
1	Driver Assistance	the driving mode-specific execution by a driver assistance system of either steering or acceleration/deceleration using information about the driving environment and with the expectation that the human driver perform all remaining aspects of the dynamic driving task
2	Partial Automation	the driving mode-specific execution by one or more driver assistance systems of both steering and acceleration/deceleration using information about the driving environment and with the expectation that the human driver perform all remaining aspects of the dynamic driving task
3	Conditional Automation	the driving mode-specific performance by an automated driving system of all aspects of the dynamic driving task with the expectation that the human driver will respond appropriately to a request to intervene
4	High Automation	the driving mode-specific performance by an automated driving system of all aspects of the dynamic driving task, even if a human driver does not respond appropriately to a request to intervene
5	Full Automation	the full-time performance by an automated driving system of all aspects of the dynamic driving task under all roadway and environmental conditions that can be managed by a human driver

Table 2.1: Table depicting different levels of vehicle automation identified in SAE (2016)

[8]SAE (2016). *J3016, Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*. Tech. rep. SAE International. DOI: 10.4271/J3016_201609. URL: https://www.sae.org/standards/content/j3016_201609/.

In this section, we will briefly present the historical developments in automated driving research and present the current state-of-the-art for some selected tasks in its sub-domains. After reviewing different aspects of knowledge representation with particular focus on automated driving (more general aspects have been discussed in section 2.2), we present related work in the fields of driving context classification (section 2.3.3), object detection and classification (section 2.3.4) and trajectory prediction (section 2.3.5). These tasks are selected based on the applications investigated in subsequent chapters in the remainder of this thesis.

2.3.1 A brief history

On the road to fully automated driving, several ADAS (Advanced Driver Assistance Systems) have been developed during the last decades and thus made a huge jump by incrementally increasing complexity and therefore the level of autonomy. The history of automated driving research goes back to the 1980's, when governmental institutions funded several explorative projects all over the world to research functionalities such as automatic vehicle driving and intelligent route planning resulting in early prototypes. In 1986, several European research groups and vehicle manufacturers started the PROMETHEUS (PROgraMme for a European Traffic of Highest Efficiency and Unprecedented Safety) project (Dickmanns et al., 1990) and demonstrated a variety of different approaches to automated driving. Another research initiative established during that period is Carnegie Mellon University's Navlab (Thorpe et al., 1988), which achieved the first completely autonomous drive from Pittsburgh to San Diego. After that first explorative phase, the US government established the NAHSC (National Automated Highway System Consortium) in 1995 and shortly followed by the foundation of the AHSRA (Advanced Cruise-Assist Highway System Research Association) in Japan in 1996. The main contribution of this first phase was the identification and deep analysis of problems, that would need to be tackled by researchers, to understand requirements and possible effects of future automated vehicles. Bertozzi et al. (2000) give an overview of the achievements and perspectives obtained in the projects during that period.

A major milestone in the research field of automated driving was the first DARPA Grand Challenge in 2004, where unmanned vehicles had to complete a 240 km, unrehearsed off-road course autonomously through the Mojave Desert in Nevada to win the price money of $1 million. Although no participating vehicle successfully finished the race (Bacha et al., 2004) in the first challenge, valuable insights have been gained. Using those insights to make significant progress, five teams (out of 23) were able to successfully complete the second DARPA Grand Challenge in 2005 with Stanford's Stanley robot winning first place (Thrun et al., 2006). After the success of the second Grand Challenge, the DARPA organized the Urban Challenge in 2007, switching the focus to automated driving in urban environments (Buehler et al., 2009). In this competition, vehicles had to complete a 97 km urban area course autonomously in less than 6 h, while obeying California state driving laws and avoiding other participating vehicles as well as other objects using only on-board sensors and GPS (Global Positioning Systems). Six vehicles out of the 11 final participants successfully finished the competition, with Carnegie Mellon's Boss robot (Urmson et al., 2008) being named the winner finishing the course in little over 4 h with an average speed of approximately 22.5 km/h.

The technology developed for the DARPA challenges formed the basis for commercial ADAS, which have seen rapid progress since then and gradually made their way into series-production vehicles. There exists a large variety of commercial systems such as ACC (Adaptive Cruise Control) or intelligent parking assistance systems modern vehicles are already equipped with. These systems have the potential to increase comfort and safety in road traffic and, in the long run, enable fully autonomous driving (cf. Level 5 in Table 2.1). On the other hand, many research teams and initiatives were spawned or inspired from these competitions and continued their research work after the DARPA Challenges. Many researchers involved in the winning teams continued their research within Google's self-driving car project, which started in 2009 and evolved into the Spin-Off company Waymo[9] in 2016. Another research team contin-

[9]Waymo LLC (2018). *Waymo webpage*. URL: https://waymo.com/ (visited on 2018-02-08).

(a) **(b)**

Figure 2.5: The winning robots from the 2005 DARPA Grand Challenge and 2007 Urban Challenge. (a) shows Stanford's Stanley at the 2005 DARPA Grand Challenge. Image source: Thrun et al. (2006). (b) shows Carnegie Mellon's BOSS at the 2007 DARPA Urban Challenge. Image source: Urmson et al. (2008).

uing their efforts after the DARPA challenges is the Annieway team[10]. One of their major contributions is the release and maintenance of the KITTI vision benchmark suite (Geiger et al., 2013), a publicly available data set containing data from various test drives in the city of Karlsruhe, rural areas as well as highways focusing on providing real world data for vision tasks like stereo, optical flow and 3D object detection and tracking.

The main research goal after the DARPA Challenges was to develop automated driving with off-the-shelf sensors. Furgale et al. (2013) present a valet-parking approach for electrified vehicles using close-to-market sensors only. Lundgren et al. (2014) show an approach to vehicle self-localization using off-the-shelf sensors in combination with a detailed map. In Aeberhard et al. (2015), results from extensive testing of automated vehicles using mainly off-the-shelf sensors in highway scenarios are presented. The sensors used are RADAR (Radio Detection and Ranging) sensors for long-range detection, US (Ultrasonic Sensors) sensors for redundant, close-range detection, a GPS for vehicle-self localization as well as a front-facing mono camera, which are all available and integrated in series-production vehicles. The only exception are 2D LIDAR (Light Detection and Ranging) sensors, which are needed for high-resolution surround view.

2.3.2 Knowledge Representation

As a consequence of increasing complexity of ADAS applications, the number of sensors mounted in modern vehicles has grown in recent years and is likely to grow even further in the near future to cover the vehicle's surrounding as completely as possible. Another factor that leads to an increasing number of sensors in automotive context are safety considerations, which demand for redundancy in the overall setup to ensure functionality even in case of failure of one sensor. Therefore, automated vehicles need to have a way to combine information from multiple sensor sources. In the literature, the distinction between different approaches is typically made based on the level at which sensory data is combined (Elfring et al., 2016). The two main classes of approaches are mostly referred to as *low-level* and *high-level* sensor fusion. By *low-level* sensor fusion, we mean that raw, previously unprocessed sensor measurements are combined in a coherent representation fur further processing. In contrast, the goal of *high-level* sensor fusion is to combine preprocessed tracks or features, that have been extracted from raw sensor measurements by each individual (smart) sensor unit.

[10]Karlsruhe Institute of Technology (2018). *Annieway Project*. URL: http://www.mrt.kit.edu/annieway/ (visited on 2018-02-22).

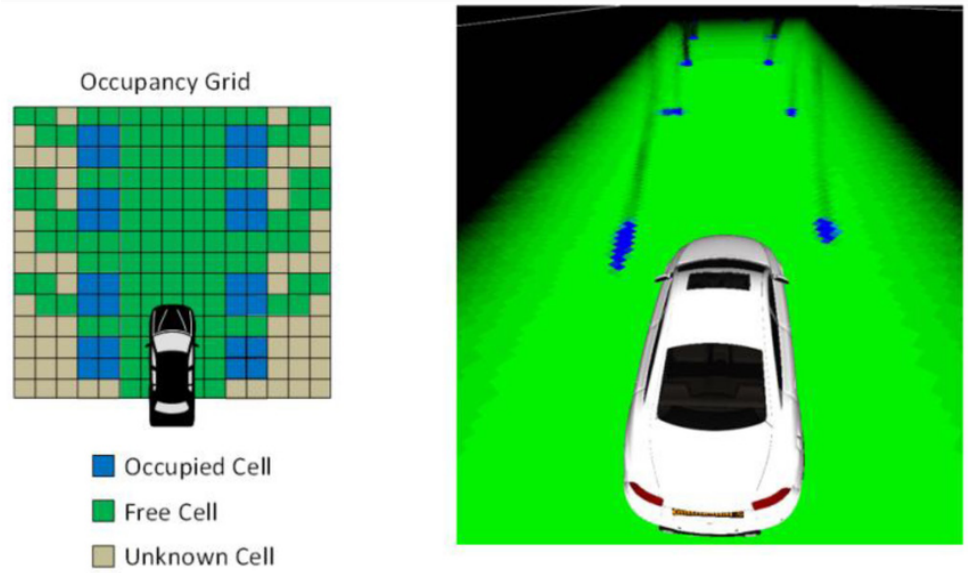


Figure 2.6: Occupancy-grid visualization for low-level sensor fusion. The left part depicts the principle of occupancy-grids, whereas the right part shows a real world example. Image source: Hohm et al. (2014)

Representations for low-level sensor fusion

One of the most widely used representations to combine low-level sensory data in robotics is an occupancy-grid map. This representation is widely used for mobile robot localization (Thrun et al., 2005), but there is also a substantial amount of research regarding occupancy-grids regarding automotive environment modeling (Tanzmeister et al., 2014; Steyer et al., 2018). The left part of Fig. 2.6 visualizes the principles of occupancy-grids, whereas the right part shows a real world example from automotive context. An occupancy-grid divides the represented space in discrete cells, which can take on different occupancy values (e.g., free, occupied or unknown). The basic assumptions for this representation are that the world is static at each time step and that each cell can have exactly one occupancy value, i.e., one cell is for example completely occupied or completely free. Therefore, an occupancy-grid is a well-suited representation for range sensors such as RADAR or LIDAR sensors. While static occupancy-grid maps are mainly used for localization and SLAM applications, there are also approaches to estimate dynamic parameters such as orientation and velocity within the grid to model dynamic environments (Tanzmeister et al., 2014). The incoming raw sensory data is typically combined through some sort of probabilistic (Bayesian) filter algorithm such as Kalman (1960) or particle-filters (Gordon et al., 1993). One of the main advantages of such a representation is that it is able to incorporate raw, unprocessed sensory data and that the discretization steps as well as the size of the cells can be chosen as fine-grained as the application demands. Furthermore, it is a natural and precise way to model and represent the free, traversable space around the robot. On the other hand, occupancy-grids have rather high requirements regarding memory and computational resources. Additionally, the representation contains no information about type or properties of the obstacles that lead to cells being occupied. Finally, in contrast to higher-level, more abstract representations, it can be more complex to incorporate sensor modalities other than range finders such as vision sensors.

Representations for high-level sensor fusion

In contrast to the low-level sensor fusion representation approaches described previously, high-level sensor fusion typically refers to sensory data being fused, that has already been preprocessed though some sort of temporal filtering. This usually happens at *tracked objects* level, i.e., object lists provided by indi-

vidual sensor units each applying their own temporal filtering or tracking based on the data they receive. To actually combine this sort of information, Bayesian filter approaches similar to the ones mentioned previously could be applied. However, such filters assume uncorrelated data and thus may deliver overconfident or diverging estimates since objects provided by different filters coming from several sensor sources are in fact correlated due to phenomena such as shared modeling assumptions, common noise acting on the objects being tracked or measurements arriving out of sequence. Therefore, more advanced track-to-track fusion algorithms need to be used for high-level sensor fusion (Tian and Bar-Shalom, 2010; Aeberhard et al., 2012). These methods typically differ regarding the amount of knowledge they assume to be known about the correlations between the tracks.
The main advantages of high-level sensor fusion are that the amount of transferred data is lower compared to low-level fusion due to the hierarchical structure of the estimation framework. Furthermore, as many sensors already employ on-board processing and provide data only at tracked object level, implementation of high-level fusion often does not require in-depth knowledge of the sensor characteristics. This allows a more abstract interface between sensors and fusion system and therefore easier replacement of individual sensor units in case the overall setup changes. On the other hand, high-level fusion ignores potentially useful information, as it might be discarded at sensor level before the fusion is performed. In summary, high-level sensor fusion is arguably the most popular and frequently used approach for combining information from different sensor sources in automated driving. The objects are typically represented as *point objects* using a unique identifier and a state vector for dynamic information such as position, velocity and acceleration. For certain application types, additional information such as the objects' size, shape and type need to be known (e.g., pedestrians have different dimensions and motion characteristics compared to trucks or vehicles). For other objects of interest (e.g., traffic signs or traffic lights), color and shape information might be important. While this type of representation is compact and abstract enough and thus well-suited for high-level fusion, the content and thus the represented values might vary depending on the sensor providing the information. Therefore, achieving a unified representation across different sensor modalities can become a challenging problem.

Other representations

Representation approaches other than the aforementioned ones are rather rarely used in an automotive context. Although semantic and structured information will grow more important with the advent of numerous and increasingly complex machine learning driven assistant systems, representations that are able to capture and manipulate such information are still hardly used in vehicle context. One example of an abstract vector representation in an automotive context called Drive2Vec is presented in Hallac et al. (2018): an unsupervised embedding of low-level sensory data into a vector, which is subsequently used to make predictions about the vehicle's state. Similar to word embedding approaches, Drive2Vec learns an embedding from a high-dimensional vector space (i.e., language or in this case several sensor streams) to a vector representation of comparably low dimension (representational space). Another example of an alternative knowledge representation approach is the work presented in Yamazaki et al. (2016), where the authors use symbolization, a language modeling technique, to obtain semantic descriptions of driving scenes.

2.3.3 Driving Context Classification

Classification of the current driving context, i.e., if the vehicle is currently driving on a highway or in an urban area, has been investigated mainly in the earlier days of ADAS research before digital maps were available at a large scale, which could be used in combination with the vehicle's current position measured using GPS to detect the driving context. However, inferring contextual information from onboard sensory data is appealing as either a fallback option in situations when GPS is not available or if keeping an updated map with driving context information is not feasible. Another simple approach is to classify the current driving context based on a set of conditional, logical rules using if-else statements

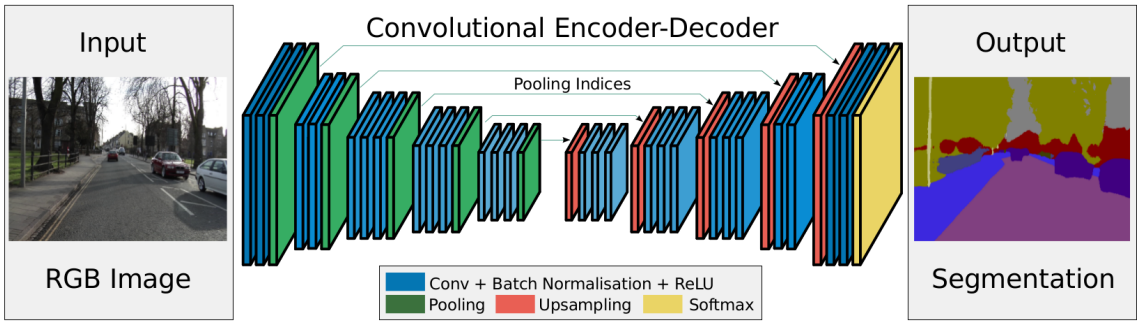


Figure 2.7: Illustration of the SegNet FCN (Fully Convolutional Neural Network) architecture for semantic segmentation of an image visualizing the input data as well as the output of the network with pixel-wise labels indicating class membership. Image source: Badrinarayanan et al. (2015)

such as "if the current velocity is greater than 100 km/h, then the current driving context is *highway*". However, this simple approach suffers from poor scaling and the difficulty to formulate exact definitions of all possible target categories in advance. Improving on a logical rule set, the approach proposed in Hauptmann et al. (1996) employs a combination of a fuzzy-logic system based on the ego-vehicle's dynamics such as velocity, active gear and acceleration and a data-driven feed-forward neural network, which additionally uses a front-facing camera. Engstrom and Victor (2001) apply statistical pattern recognition and a simple feed-forward neural network using solely the ego-vehicle's dynamics to classify four different driving context categories. Modern approaches tend to focus more on the classification and analysis of particular traffic situations as in Hermann and Desel (2008) or driving events as in D'Agostino et al. (2013), which can then be put in relation to the driver's behavior.

2.3.4 Object Detection and Classification

Detecting objects in the vehicle's surroundings and classifying their type is a central research problem in automated driving and more generally, in AI. Knowledge about type, position and behavior of other traffic participants as well as road lane topology is a crucial component for safe automated driving. The algorithmic choice for detecting dynamic objects depends heavily on the sensor modality used. If LIDAR sensors are used, typically model-based approaches involving the object's geometry and dynamics are employed (Petrovskaya and Thrun, 2009b; Petrovskaya and Thrun, 2009a; Darms et al., 2008). Classic computer vision approaches employ probabilistic methods combined with information about context, scale or shapes detected using engineered features such as part-based models or HOG (Histogram of Oriented Gradients) features to identify cars (Held et al., 2012), traffic signs (H. Li et al., 2015) or the road topology (Alvarez and Lopez, 2011; Beyeler et al., 2014). Such methods however have been drastically outperformed in recent years through sophisticated DNN architectures such as CNNs with remarkable results on tasks like traffic sign detection (Ciresan et al., 2012a; Sermanet and LeCun, 2011) achieving partly super-human performance (Stallkamp et al., 2012). Subsequently, DNNs have been used to segment the road, or more generally, the traversable space (Mohan, 2014; Bittel et al., 2015) or for semantic segmentation of the complete image, i.e., labeling each pixel in the image with the object class it belongs to (cf. Fig. 2.7), using FCNs (Fully Convolutional Neural Networks) (Badrinarayanan et al., 2015; Long et al., 2015; G. Chen et al., 2018). The approach proposed in X. Li et al. (2017) deals with the special case of detecting vulnerable road users, i.e., pedestrians and bicyclists, by incorporating a CNN variant for classification and localization. Huval et al. (2015) evaluate general applicability and usability of CNNs for lane and vehicle detection when deployed on a real-time critic system concluding that powerful GPUs enable real-time capabilities of such approaches. An overview over recent approaches to tackle object detection and semantic segmentation using DNNs in an automotive context can be found in Feng et al. (2019), while Janai et al. (2017) give a more general overview of computer vision for

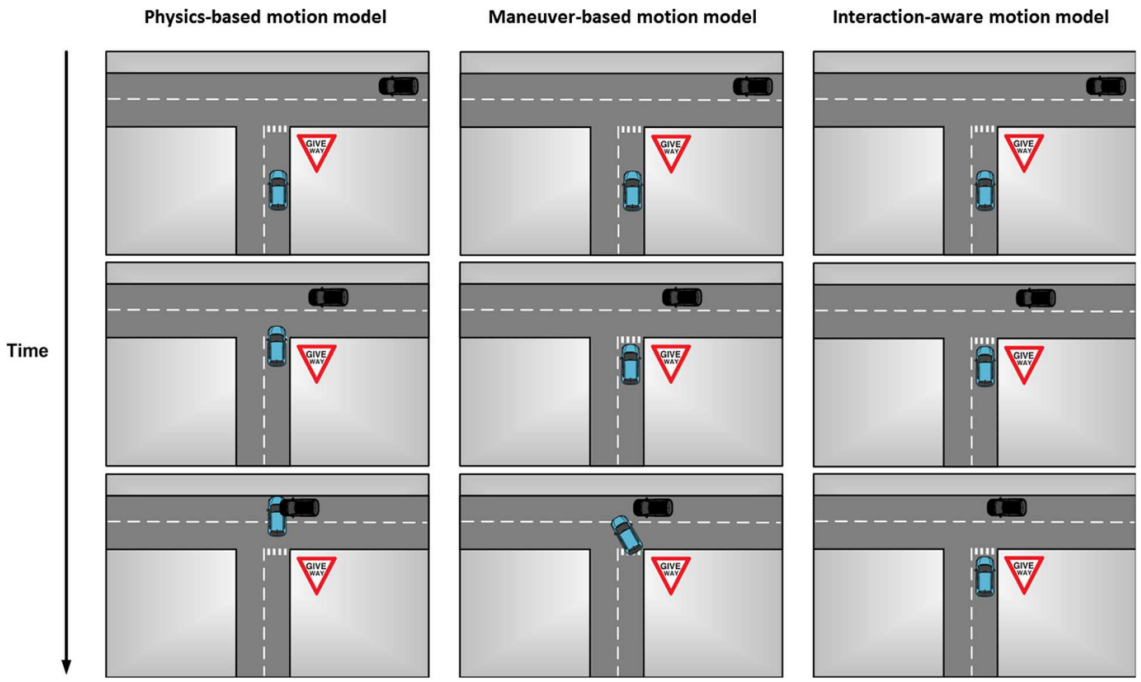


Figure 2.8: Examples visualizing different modeling approaches for motion prediction in automotive context. Image source: Lefèvre et al. (2014)

automated vehicles.

2.3.5 Trajectory Prediction

Predicting future behavior and positions of other traffic participants from observations is essential for collision avoidance and thus safe motion planning, and needs to be solved by human drivers and automated vehicles alike to reach their desired goal. However, future positions of vehicles not only depend on each vehicle's own past positions and dynamics like velocity and acceleration, but also on the behavior of the other traffic participants in the vehicle's surroundings. Motion prediction for intelligent vehicles in general has seen extensive research in recent years, as it is a cornerstone for collision-free automated driving. Lefèvre et al. (2014) classify such prediction approaches into three categories, namely *physics-based*, *maneuver-based* and *interaction-aware*, depending on their level of abstraction. Figure 2.8 illustrates the differences between the three families of approaches: the *physics-based* approach assumes a constant velocity of both vehicles, whereas the *maneuver-based* approach assumes that the blue car turns left while the black car goes straight. Finally the *interaction-aware* approach assumes similar motions as the *maneuver-based* approach whereas the motion of each vehicle is constrained by the traffic rules as well as the relative behavior of the other car.

Physics-based approaches represent vehicles as dynamic entities, which obey the laws of physics. To predict future motion, such approaches employ dynamic and kinematic models linking control inputs, car properties and external conditions to the evolution of the state of the vehicle. Hence, such approaches are well-suited for short term trajectory prediction but suffer from instabilities when predicting longer time-windows into the future and are unable to account for any change in vehicle motion caused by a particular maneuver (e.g., slowing down or performing a turn). *Maneuver-based* approaches are more advanced assuming the vehicle motion is a series of maneuvers, which are performed independently from the other traffic participants. Such approaches rely on early detection of the maneuver the driver intends to perform, which is then assumed to match the future behavior of the vehicle. If the identification of the maneuver was correct, these approaches allow for more accurate long term predictions compared to purely physics-based approaches. However, they ignore potential interconnections in the motion between

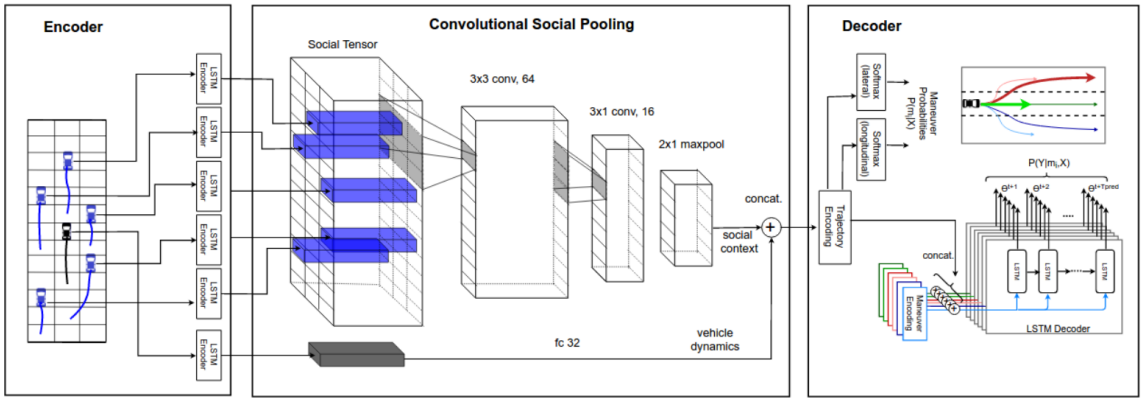


Figure 2.9: Visualization of one state-of-the-art LSTM-based architecture for vehicle trajectory prediction combining social pooling to account for the influence of other vehicles on the target vehicle with a maneuver-based prediction module. Image source: Deo and Trivedi (2018a).

several traffic participants, which, in practice, share the road and the motion of one vehicle will influence the motion of other vehicle in its surroundings.
There exist a growing number of different *interaction-aware* approaches to account for those dependencies and mutual influences between traffic participants or, more generally, agents in the scene. Probabilistic models like cost maps (Bahram et al., 2016) account for physical constraints on the movements of the other vehicles. Classification approaches categorize and represent scenes in a hierarchy (Bonnin et al., 2012) based on the most generic ones to predict behavior for a variety of different situations. Data-driven approaches to behavior prediction mainly rely on LSTM neural network architectures (Hochreiter and Schmidhuber, 1997), which have proven to be a powerful tool for sequential data analysis. Alahi et al. (2016) model interactions in the learning network architecture by introducing so-called social-pooling layers to connect several LSTMs each predicting one agent at a time. Altche and La Fortelle (2017) use a LSTM network and account for interactions by including distances between the target vehicle and other agents directly in the training data. A similar approach is proposed in Deo and Trivedi (2018b), but the authors combine LSTM networks with an additional maneuver classification network to predict future vehicle motion. Deo and Trivedi (2018a) adapted the combination of several LSTM networks to encode vehicle trajectories, (convolutional) social-pooling layers to account for interactions between the vehicles, and a maneuver-based LSTM decoder to predict vehicle trajectories in highway situations (see Fig. 2.9). LSTM-models are currently considered the most successful approaches and therefore the state-of-the-art regarding trajectory prediction. One issue with data-driven approaches to behavior prediction accounting for relations between agents is that the number of other vehicles is variable. If information about vehicles other than the target are encapsulated in the input of the neural network, typically a specific number of other vehicles of interest are manually chosen a priori to avoid this issue (Altche and La Fortelle, 2017; Deo and Trivedi, 2018b). If the information about other vehicles is encapsulated in the network architecture, it might be necessary to train several networks depending on the situation at hand.

2.3.6 Online Learning

Common to all approaches mentioned in section 2.3.5 is the fact, that the models are trained offline on batched data and remain unchanged during deployment. In contrast, incremental or online learning approaches, which attempt to tackle learning tasks by processing sequential data one at a time, gained growing interest as an attractive alternative to update learning models during deployment. This approach is particularly interesting in the context of big data and in situations, where the system needs to learn from continuously incoming data streams and a complete data set during offline training is not available.

There exists an increasing number of online learning approaches in several problem domains (see Losing et al. (2018), Gomes et al. (2017), and Hoi et al. (2018) for a comprehensive overview of the field). However, such models adapting their (neural) weights at run time through online learning are rather rarely investigated in automotive context due to safety considerations, issues with convergences time as well as the lack of proofs/guarantees that the models converge at all. The model presented in Maye et al. (2011) uses self-supervised online learning to recognize, classify and add new maneuvers of the ego-vehicle at run time. The approach employed in Graf et al. (2014) uses case-based reasoning to learn to predict driving behavior for specific driving situations, namely intersection scenarios. Losing et al. (2017) employ incremental learning to personalize maneuver prediction at intersections. An alternative approach to combining several weak learning models such as stumps or smoothing splines through an averaging scheme is boosting (Taieb and Hyndman, 2014), which offers superior performance over the individual learners. An online learning approach based on SNNs has been shown in DeWolf et al. (2016) to be successful in similar domains such as adaptive robot arm control, but has not been applied yet in an automotive application such as trajectory prediction.

2.3.7 Data sets

With the advent and success of sophisticated DNN architectures and their success in several domains related to automated driving such as object detection (see section 2.3.4) and behavior prediction (see section 2.3.5), the demand for large-scale data sets for training such networks grew significantly in recent years. Furthermore, there is an interest in publicly available data sets, which not only enable competition between researchers but also allow for transparent benchmarking of approaches against each other. Data sets for image classification such as the MNIST data set for hand-written digits (LeCun et al., 1998), the GTSRB for traffic sign recognition (Stallkamp et al., 2012) and the ImageNet data set (Deng et al., 2009) provided researchers with large amounts of labeled image data facilitating significant progress in this research field.

One of the first data sets being made publicly available on a larger scale for research in an automotive context and encouraging competition is the KITTI vision benchmark suite (Geiger et al., 2013), which was recorded using the Annieway vehicle prototype[11] in urban environments in the region of the German city Karlsruhe. The KITTI data set offers raw sensory data for different sensor modalities such as LIDAR and vision alongside ground truth labels and benchmarks for several tasks such as road detection, object detection, optical flow and tracking. Another large-scale data set targeted at computer vision in an automotive context is the Cityscapes data set (Cordts et al., 2016). It provides high-definition images recorded from a driving vehicle with accurate pixel-wise labels for semantic segmentation. Furthermore, the Cityscapes data set also contains labels for depth information obtained from stereo vision, which for instance allows to train neural networks to provide depth information based only on monocular input. A comparable data set for semantic image segmentation in a broader context (i.e., not restricted to an automotive context) is the MS COCO (Microsoft Common Objects in COntext) data set (Lin et al., 2014). Furthermore, the MS COCO data set also provides ground truth captions for each images allowing to train neural networks translating visual input to language. Another more recent data set similar to the KITTI data set is the Apollo Scape data set (Huang et al., 2018), which is provided by the Chinese company Baidu and contains data and ground truth labels for several automotive applications such as scene parsing, lane segmentation, localization, tracking and trajectory prediction. The NGSIM US-101 data set (Colyar and Halkias, 2018) is a publicly available data set originally intended for driver behavior and traffic flow models (He, 2017), that has been used to train trajectory predictions models in Altche and La Fortelle (2017) and Deo and Trivedi (2018b) as well. It was recorded on a segment of approximately 640 m length with 6 lanes on the US-101 freeway in Los Angeles, California using cameras observing freeway traffic from rooftops with trajectory-data being extracted later from the obtained video footage.

[11] Karlsruhe Institute of Technology (2018). *Annieway Project*. URL: http://www.mrt.kit.edu/annieway/ (visited on 2018-02-22).

2.4 Summary

In this chapter, we have taken a round trip through several research areas related to the thesis at hand. We visited the historical and current developments in the field of AI, machine learning and neuromorphic engineering. Especially the young and emerging research field of neuromorphic engineering shows promise to be an energy-efficient possibility for future applications with tight restrictions regarding their energy-budget and computational resources. As a mobile application with a growing number of sensors and increasingly complex modules (often employing modern machine learning techniques) involved not only in autonomous navigation but also for interacting with the passengers inside and other traffic participants outside the car, intelligent vehicles in general offer promising application possibilities for the technology developed in this research area. However, the computing hardware prototypes (Furber et al., 2014; Akopyan et al., 2015; M. Davies et al., 2018) as well as the novel senor technologies (Lichtsteiner et al., 2008) are not mature enough yet to either become commercial products or to be deployed in series-production vehicles. Additionally, the principles of SNNs in combination with dedicated computing hardware show promise to be an energy-efficient algorithmic substrate to be applied in future automotive applications. However, this algorithmic approach has only been investigated in rather simple robotic applications (Conradt et al., 2015; T. C. Stewart et al., 2016; Galluppi et al., 2014) and, similar to the neuromorphic hardware, is not mature enough yet, especially for such safety-critical applications like automated driving.

Furthermore, we presented an overview of different architectures for cognitive modeling putting emphasis on vector-based representations such as the SPA or, more generally, VSAs. After showing alternative approaches such as symbolic and connectionist cognitive architectures, we also reviewed the most prominent applications of such architectures and the representations they employ. We have seen, that there is a gap between architectures that are well-suited for modeling higher-level cognitive tasks and lower-level tasks closer to the dynamics of perception and action. While the former is most successfully tackled with symbolic modeling architectures, the latter often demands for precise mathematical modeling for dealing with the complexity and dynamics of real-world physics. Therefore, little attempts have been made to use cognitive modeling in real robotic systems and if, the investigated tasks are often simplified (Neubert et al., 2016) and still far away from the demands imposed by complex systems like automated vehicles. Although we do not attempt to close this gap in our proposed work, we show that structured vector representations are an interesting option to encode automotive scenes in an abstract, unified and sensor independent substrate. Today's sensing and processing hardware is currently not intended to support such a representation. However, the full strength of this approach can only unfold once it is possible to employ cognitive modeling and spiking neuron principles in the complete chain from sensing to behavior with dedicated hardware support.

Finally, we summarized the current state-of-the-art in several sub-domains of automated driving related to our work. After a brief historical overview, we focused on reviewing approaches currently used for representing knowledge in an automotive context. As stated earlier, the techniques employed here use mainly Bayesian statistics and mathematical modeling to deal with the complexity of the real world. Subsequently, we reviewed the state-of-the-art in several selected tasks from sub-domains related to automated driving. These tasks mirror the application examples we have chosen as use-cases to investigate our approaches and will be revisited in subsequent chapters (see chapters 5, 6 and 7).

Admittedly, our work touches and incorporates prior work from very different research fields by developing an approach for representing automotive scenes in an abstract vector representation inspired by modern cognitive modeling approaches. To the best of our knowledge, our work is the first to employ such a unified vector representation in an automotive context. The only other work encapsulating vehicle data in a compact vector representation is presented in Hallac et al. (2018), whose approach uses sensory data at a far lower level (i.e., closer to the actual sensor data) than we do and targets different application scenarios.

In the next chapter, we give a detailed introduction to the mathematical theory behind VSAs in general

and more specifically to the SPA as well as to the principles of the NEF. Together with the review presented here, this will form the theoretical basis for the remainder of the thesis.

3 Theoretical background

In this chapter, we present the theoretical background and mathematical formalism needed as basis for later chapters. We introduce VSAs (Vector Symbolic Architectures), describe their most important properties and present proofs where relevant for this thesis. We proceed with one particular instantiation of VSA, the SPA (Semantic Pointer Architecture), and present more specific properties, which do not necessary hold for all VSAs. Additionally, we give a brief introduction to the theory of the NEF (Neural Engineering Framework), a set of mathematical tools enabling the implementation of the SPA in SNNs. Then, we shift our focus to cognitive modeling based on such vector representations keeping it as generic as possible without particular attention on the SPA. We show several approaches to generate vector vocabularies, which form the basis of more complex structured representations built from them using the VSA's algebraic operations. Bringing all of the presented tools together, we show how the NEF can be used to implement vector representations, which have been generated using the SPA, in a spiking neuron substrate.

3.1 Mathematical properties of Vector Symbolic Architectures

The term VSAs - first coined by Gayler (2003) - refers to a class of similar approaches for cognitive modeling making use of distributed representations. The basic idea behind all of those approaches is to represent structure, i.e., cognitive concepts, symbols or language, in a high-dimensional vector space by mapping each entity to be represented to a (possibly random) vector. One of the most important properties of high-dimensional vector spaces enabling this kind of representation is the fact that two high-dimensional random vectors are likely to be dissimilar. In the following, we will show what we mean by fuzzy terms like *dissimilar* or *likely* and provide more precise statements.
One main requirement in the context of cognitive modeling is the ability of the modeling framework to address the binding problem (Treisman, 1999). Jackendoff (2002) phrases this as the problem of "combining independent bits into a single coherent percept". One strength of VSAs is that they offer the possibility to manipulate their entities (i.e., vectors) through algebraic operations, usually at least one *addition-like* and *multiplication-like* operation each. Typically, the multiplication operation is used for binding different representations into a new vector. This operation, depending on the vector representation, is constructed with some desirable properties in mind (see Definition 3.10). A first attempt on using a multiplication operation for binding vectors was done by Smolensky (1990) using the tensor product. The major drawback of this approach is exploding dimensionality of the tensor product. For finite dimensional vector spaces V and W of dimensions n and m, the tensor space $V \otimes W$ is a vector space of dimension $n \cdot m$. As a consequence, each binding operation $v \otimes w$ for vectors $\mathbf{v} \in V, \mathbf{w} \in W$ would increase the dimension of the representational space, which is computationally unfeasible and leads to poor scaling. This lead researchers to define several slightly different multiplication or binding operations, depending on the underlying numerical structure. The most prominent examples are element-wise multiplication in the MAP-architecture proposed by Gayler (1998), the XOR-operation in BSCs proposed in Kanerva (2000) and Kanerva (2009) as well as circular convolution HRRs proposed in T. Plate (1991) and T. Plate (1994).

Definition 3.1. Let $N \subseteq K$ be a subset of some number field K (i.e., a set of numbers) and $D \in \mathbb{N}$ a natural number. Furthermore, let

$$V_D(N) = \{\mathbf{x} = (x_0, \cdots, x_{D-1}) \,|\, x_i \in N\} \subseteq K^D$$

be the set of all D-tuples with entries in N. Let

$$\oplus : V_D(N) \times V_D(N) \longrightarrow K^D, (\mathbf{v}, \mathbf{w}) \longmapsto \oplus(\mathbf{v}, \mathbf{w}) =: \mathbf{v} \oplus \mathbf{w},$$
$$\circledast : V_D(N) \times V_D(N) \longrightarrow K^D, (\mathbf{v}, \mathbf{w}) \longmapsto \circledast(\mathbf{v}, \mathbf{w}) =: \mathbf{v} \circledast \mathbf{w}$$

be functions with $\oplus$ following the rules of ordinary addition - namely commutativity, associativity, existence of a neutral element and existence of inverse elements - and for any elements $\mathbf{u}, \mathbf{v}, \mathbf{w} \in V_D(N)$

$$\mathbf{u} \circledast (\mathbf{v} \oplus \mathbf{w}) = \mathbf{u} \circledast \mathbf{v} \oplus \mathbf{u} \circledast \mathbf{w}.$$

If there is furthermore a distinct element $\mathbf{1} \in V_D(N)$ with

$$\mathbf{v} \circledast \mathbf{1} = \mathbf{1} \circledast \mathbf{v} = \mathbf{v}$$

for any $\mathbf{v} \in V_D(N)$ and a function $\phi : V_D(N) \times V_D(N) \longrightarrow [-1, 1]$, we call $(V_D(N), \circledast, \oplus, \phi)$ a *VSA (Vector Symbolic Architecture)* of dimension D. The function ϕ is called a *measure of similarity*. If N is a subset of the real or complex numbers, i.e., $N \subset \mathbb{R}$ or $N \subseteq \mathbb{C}$, we call any VSA $(V_D(N), \circledast, \oplus, \phi)$ *continuous*.

Although the set $V_D(N)$ might not be a vector space in the strict mathematical sense (in most cases it is at least a subset of a vector space), we will refer to its elements as *vectors*. Before we proceed in deriving some important properties of VSAs, we present some of the most prominent examples.

Example 3.2. *Vector Symbolic Architectures*

1. *The first example of a VSA is Binary Spatter Codes (BSCs). Kanerva (2009) restricts the elements of his vectors to binary values, i.e., $N = \{0, 1\} = \mathbb{F}_2$. The operations $\circledast$ and $\oplus$ in this case are the XOR-function and a thresholded sum respectively. With $\mathbf{v}_i = (v_{i0}, \cdots, v_{iD-1}) \in V_D(N)$ and $i \in \{1, \cdots, n\}$, the operation $\oplus$ is usually defined in the following way*

$$\mathbf{v}_1 \oplus \cdots \oplus \mathbf{v}_n =: \mathbf{x} = (x_0, \cdots, x_{D-1}) \text{ with}$$
$$x_j := \begin{cases} 1 & \sum_{i=1}^{n} v_{ij} \geq \frac{n}{2} \\ 0 & \sum_{i=1}^{n} v_{ij} < \frac{n}{2} \end{cases}.$$

   This definition ensures, that the results of the addition operation $\oplus$ remain binary. Usually, a normalized Hamming distance

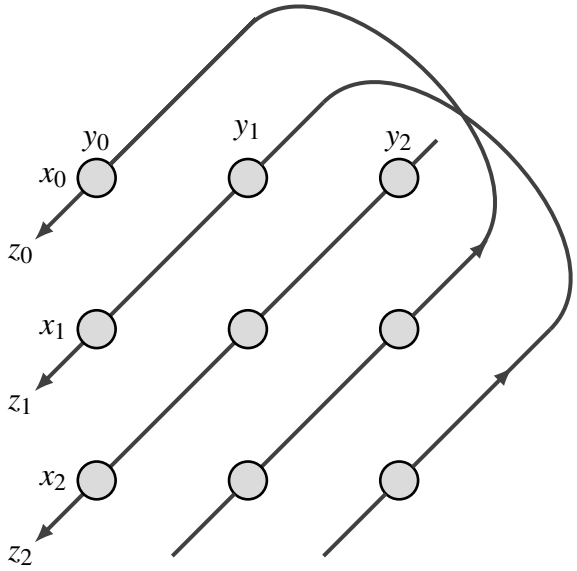
$$\phi(\mathbf{v}, \mathbf{w}) := 1 - \frac{2}{D} |\{v_i \neq w_i | i \in \{0, \cdots, D-1\}\}|$$

   is used as a measure of similarity in this architecture. BSCs have some interesting properties compared to other VSAs: The neutral element for both operations $\circledast$ and $\oplus$ is the vector $\mathbf{0} := (0, \cdots, 0)$, while all vectors are self-inverse regarding the multiplication operation $\circledast$, i.e., $\mathbf{v} \circledast \mathbf{v} = \mathbf{0}$ for any $\mathbf{v} \in V_D(N)$.

2. *The first example of a VSA in continuous space is the MAP (Multiply-Add-Permutate) architecture proposed by Gayler (1998) with $N \subseteq \mathbb{R}$ and the cosine similarity as measure of similarity*

$$\phi(\mathbf{v}, \mathbf{w}) = \frac{\mathbf{v} \cdot \mathbf{w}}{\|\mathbf{v}\| \, \|\mathbf{w}\|} = \cos(\theta),$$

   with θ being the angle between the vectors $\mathbf{v}, \mathbf{w} \in V_D(N)$. The operations $\circledast$ and $\oplus$ are simply element-wise multiplication and addition with neutral elements $\mathbf{1} = (1, \cdots, 1)$ and $\mathbf{0} := (0, \cdots, 0)$ respectively.



$$\mathbf{z} = \mathbf{x} \circledast \mathbf{y}$$
$$z_0 = x_0y_0 + x_2y_1 + x_1y_2$$
$$z_1 = x_1y_0 + x_0y_1 + x_2y_2$$
$$z_2 = x_2y_0 + x_1y_1 + x_0y_2$$

Figure 3.1: Visualization of circular convolution as compressed outer product for 3-dimensional vectors. Image adapted from T. A. Plate (1994).

3. *Another example of a VSA in continuous space is Holographic Reduced Representations (HRRs) proposed by T. Plate (1991). The main difference compared to the MAP architecture is, that T. Plate (1994) in general allows complex vector values, i.e., $N \subseteq \mathbb{C}$ and uses a different multiplication operation $\circledast$: circular convolution. For any two vectors $\mathbf{x}, \mathbf{y} \in V_D(N)$, circular convolution $\circledast$ is defined as*

$$\mathbf{z} = \mathbf{x} \circledast \mathbf{y} \qquad \text{with } z_j := \sum_{k=0}^{D-1} x_k y_{(j-k) \bmod D}.$$

The neutral element regarding circular convolution is $\mathbf{1} = (1, 0, \cdots, 0)$. One important property of this operation is the fact that circular convolution can efficiently be computed using the DFT (Discrete Fourier Transform). The DFT is defined as the function

$$DFT : \mathbb{C}^D \longrightarrow \mathbb{C}^D, \mathbf{x} \longmapsto \left(\sum_{j=0}^{D-1} x_j \zeta_D^{-jk} \right)_{k=0}^{D-1} \qquad \text{with } \zeta_D = \exp\left(\frac{i2\pi}{D}\right). \tag{3.3}$$

Similarly, the IDFT (Inverse Discrete Fourier Transform) is defined as the function

$$IDFT : \mathbb{C}^D \longrightarrow \mathbb{C}^D, \mathbf{x} \longmapsto \left(\frac{1}{D} \sum_{j=0}^{D-1} x_j \zeta_D^{jk} \right)_{k=0}^{D-1}. \tag{3.4}$$

From the convolution theorem (see Bracewell, 2000, Chap. 6) we know, that we can calculate the circular convolution of any two vectors $\mathbf{v}, \mathbf{w} \in V_D(N)$ by

$$\mathbf{v} \circledast \mathbf{w} = IDFT\left(DFT(\mathbf{v}) \odot DFT(\mathbf{w})\right), \tag{3.5}$$

with $\odot$ denoting element-wise multiplication in this case. This induces that circular convolution obeys the same rules (commutativity and associativity) as element-wise multiplication, as both operations are the same except for a change of basis.

As mentioned earlier, one of the most important features of (high-dimensional) VSAs is the fact that two random vectors are likely to be dissimilar. We will derive this result in the following Theorem.

Theorem 3.6. *Let $(V_D(N), \circledast, \oplus, \phi)$ a Vector Symbolic Architecture. For two randomly chosen vectors $\mathbf{v}, \mathbf{w} \in V_D(N)$, the distribution of the similarity $\phi(\mathbf{v}, \mathbf{w})$ is a version of the beta-distribution $\mathscr{B}_{\frac{D-1}{2}, \frac{D-1}{2}}$ scaled and shifted to the interval $[-1,1]$ with mean $\mu = 0$ and variance $\sigma^2 = \frac{c^2}{D}$ up to a constant c. The standardized distribution trends towards a normal distribution with growing D.*

Proof. We will only give the proof of this Theorem for real valued VSAs, i.e., $N \subseteq \mathbb{R}$ and ϕ as the cosine similarity. Without loss of generality, we assume the vectors $\mathbf{v}, \mathbf{w}$ picked randomly from the unit sphere $\mathbb{S}^{D-1} = \{\mathbf{v} \in \mathbb{R}^D | \, \|\mathbf{v}\| = 1\}$, as we can simply normalize the vectors by $\frac{\mathbf{v}}{\|\mathbf{v}\|}$. Since binary VSAs can be associated with a euclidean sphere as well, the same result can be proven for those architectures with similar arguments (see Kanerva, 1988, for details). Due to symmetry of the unit sphere $\mathbb{S}^{D-1}$, we can furthermore - again without loss of generality - choose one vector as a unit vector, i.e., $\mathbf{w} = (1, 0, \cdots, 0)$. Thereby, the cosine similarity for $\mathbf{v} = (v_0, \cdots, v_{D-1})$ is given by $\phi(\mathbf{v}, \mathbf{w}) = v_0$ By fixing one coordinate, we get the constraint $\sum_{i=1}^{D-1} v_i^2 = 1 - v_0^2$, which is equivalent to a lower dimensional sphere $\mathbb{S}^{D-2}$ with radius $\sqrt{1 - v_0^2}$. Hence, the cosine similarity $\phi(\mathbf{v}, \mathbf{w}) =: x$ is proportional to the surface of a conical frustum constructed from $\mathbb{S}^{D-2}$ with radius $\sqrt{1-x^2}$, slope $\frac{1}{\sqrt{1-x^2}}$ and some height h, i.e., the density function is proportional to

$$f_{\phi(\mathbf{v},\mathbf{w})}(x) \propto \frac{\sqrt{1-x^2}^{(D-2)}}{\sqrt{1-x^2}} h \propto \left(1-x^2\right)^{\frac{D-3}{2}}.$$

Substituting $x = 2u - 1$, we get

$$\left(1-(2u-1)^2\right)^{\frac{D-3}{2}} \propto \left(u-u^2\right)^{\frac{D-3}{2}} = \left(u(1-u)\right)^{\frac{D-3}{2}} = u^{\left(\frac{D-1}{2}-1\right)} (1-u)^{\left(\frac{D-1}{2}-1\right)},$$

which is the density function of the beta distribution $\mathscr{B}_{\frac{D-1}{2}, \frac{D-1}{2}}$. Thus, the cosine similarity is also beta distributed, but scaled and shifted to the interval $[-1,1]$ by $x = 2u - 1$.
For $\alpha = \beta = \frac{D-1}{2}$, the mean of the beta distribution is $\tilde{\mu} = \frac{1}{2}$. Applying the substitution, we get the mean of the shifted distribution $\mu = 2\tilde{\mu} - 1 = 0$.
Making use of the simplification that the distribution of similarity is the same as the distribution in the first coordinate, the variance is given by the expected value of the square value of the first coordinate, i.e., $\mathbb{E}(v_0^2)$. Since all coordinates are identically distributed, we get

$$\mathbb{E}(v_0) = \frac{1}{D} \sum_{i=0}^{D-1} \mathbb{E}(v_i^2) = \frac{1}{D} \underbrace{\mathbb{E}\left(\sum_{i=0}^{D-1} v_i^2\right)}_{=:c^2} = \frac{c^2}{D}.$$

Hence, the variance of the distribution of the cosine similarity is $\sigma^2 = \frac{c^2}{D}$. In the particular case of the unit sphere $\mathbb{S}^{D-1}$, we get $c^2 = 1$ and a variance of $\sigma^2 = \frac{1}{D}$.
To see the convergence behavior of the standardized distribution, we look at the logarithm of its density function

$$\log\left(f_{\phi(v,w)}\left(\frac{x}{\sqrt{D}}\right)\right) = \frac{D-3}{2} \log\left(1 - \frac{x^2}{D}\right) + C. \tag{3.7}$$

Using the Taylor series approximation of the logarithm, equation (3.7) transforms to

$$\log\left(f_{\phi(v,w)}\left(\frac{x}{\sqrt{D}}\right)\right) = \frac{D-3}{2}\left(-\frac{x^2}{D} + \frac{x^4}{4D} \pm \dots\right) + C = -\frac{1}{2}x^2 + \frac{3}{2D}x^2 + \mathscr{O}\left(\frac{x^4}{D}\right) + C$$
$$\longrightarrow -\frac{1}{2}x^2 + C = \log\left(f_{\mathscr{N}}(x)\right) \text{ for } D \longrightarrow \infty.$$

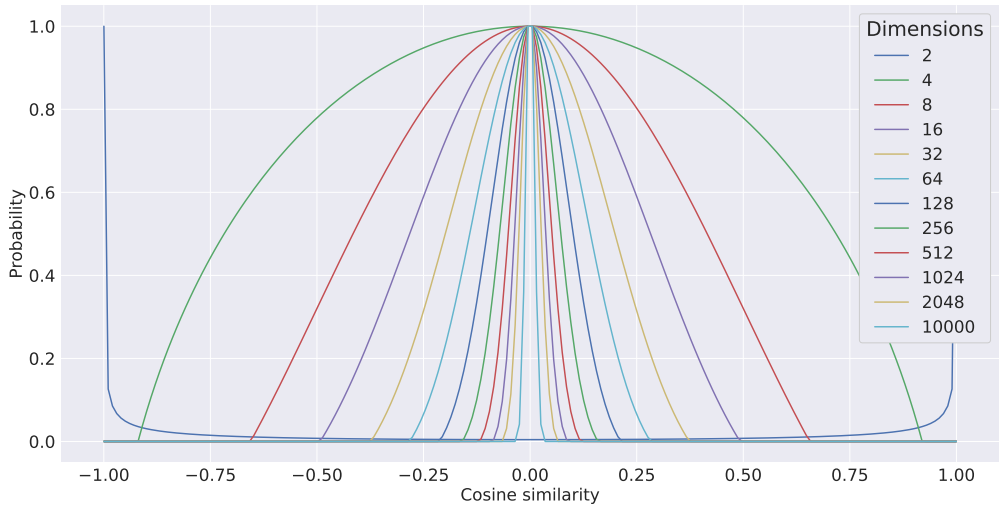


Figure 3.2: Visualization of the distribution of the cosine similarity between two randomly chosen vectors depending on their dimension.

Hence, with growing D the standardized distribution of the cosine similarity trends to a normal distribution. □

Theorem 3.6 states that the probability of finding two random, non-orthogonal vectors in a VSA decreases with growing vector dimension. Figure 3.2, which shows the probability distributions of cosine similarity for different vector dimensions, illustrates this result. Furthermore, Theorem 3.6 allows us to give a more formal definition of the term *dissimilar*.

Definition 3.8. Let $(V_D(N), \circledast, \oplus, \phi)$ be a VSA (Vector Symbolic Architecture) of dimension D and $c \in \mathbb{N}$ a constant. We call any two vectors $\mathbf{v}, \mathbf{w} \in v_d(n)$ *dissimilar*, if

$$|\phi(\mathbf{v}, \mathbf{w})| \leq \varepsilon, \text{ with } \varepsilon := \frac{c}{\sqrt{D}}. \tag{3.9}$$

Analogously, we call any two vectors $\mathbf{v}, \mathbf{w} \in V_D(N)$ *similar*, if $|\phi(\mathbf{v}, \mathbf{w})| > \varepsilon$. Similar vectors are denoted by $\mathbf{v} \approx \mathbf{w}$.

Definition 3.8 can also be stated as follows: we consider any two vectors similar, if their similarity is higher than what we would expect from two randomly chosen vectors. Therefore, we make use of the fact that for growing dimension D the cosine similarity follows approximately a normal distribution $\mathcal{N}_{\mu,\sigma}$, with $\mu = 0$ and $\sigma = \frac{1}{\sqrt{D}}$ and the so-called three-sigma-rule. This rule, which follows from Chebyshev's inequality, states that the probability $\mathbb{P}(\mu - 3\sigma \leq X \leq \mu + 3\sigma) \geq 0.95$ for any unimodal distributed random variable X. For normally distributed X, we even have

$$\mathbb{P}(\mu - 2\sigma \leq X \leq \mu + 2\sigma) \approx 0.954$$
$$\mathbb{P}(\mu - 3\sigma \leq X \leq \mu + 3\sigma) \approx 0.997.$$

Given Definition 3.8, the probability of two randomly chosen vectors being similar is below 5 % for $c = 2$ and even below 0.3 % for $c = 3$, while the actual numerical interval $[-c\sigma, c\sigma]$ only depends on the vector dimension D. Hence, we denote the weaker version of Definition 3.8 by $\varepsilon_{weak} = \frac{2}{\sqrt{D}}$ as *weak similarity threshold* and the stronger version by $\varepsilon_{strong} = \frac{3}{\sqrt{D}}$ as *strong similarity threshold*. For lower VSA dimensions such as $D \leq 50$, it can be beneficial to work with the weak similarity threshold, whereas for higher dimensions the stronger version can be used.
Based on the definition of similarity, we derive criteria for "good" multiplication functions in VSAs.

Definition 3.10. Let $(V_D(N), \circledast, \oplus, \phi)$ be a VSA (Vector Symbolic Architecture) of dimension D. We call its multiplication function

$$\circledast : V_D(N) \times V_D(N) \longrightarrow K^D, (\mathbf{v}, \mathbf{w}) \longmapsto \circledast(\mathbf{v}, \mathbf{w}) =: \mathbf{v} \circledast \mathbf{w}$$

a *binding function* if

1. for any two vectors $\mathbf{v}, \mathbf{w} \in V_D(N)$, the vector $\mathbf{v} \circledast \mathbf{w}$ is dissimilar to both $\mathbf{v}$ and $\mathbf{w}$, i.e.,

$$|\phi(\mathbf{v}, \mathbf{v} \circledast \mathbf{w})| \leq \varepsilon \text{ and } |\phi(\mathbf{w}, \mathbf{v} \circledast \mathbf{w})| \leq \varepsilon.$$

2. for any vector $\mathbf{v} \in V_D(N)$, there exists a vector $\bar{\mathbf{v}} \in V_D(N)$ with $\mathbf{v} \circledast \bar{\mathbf{v}} \approx \mathbf{1}$. We call $\bar{\mathbf{v}}$ a *pseudo-inverse* element. If furthermore $\mathbf{v} \circledast \bar{\mathbf{v}} = \mathbf{1}$, we call the vector $\bar{\mathbf{v}}$ *exact inverse*.

It is worth noting, that all multiplication operations mentioned in Example 3.2 fulfill the criteria for binding functions as stated in Definition 3.10. The first criteria is intended to allow structured representations in VSAs. Representations built solely upon an addition function lack a mechanism to impose structure, as the sum of vectors is similar to all summands. For continuous VSAs, this result is straightforward due to the linearity of the dot product, but it holds true for BSCs as well. Therefore, summing vectors only allows to encode unordered sets of entities. The property of binding functions to map two vectors to a vector dissimilar to both inputs enables structured representations.
The second criteria of Definition 3.10 for binding functions is the basis to decode or recover the individual vector ingredients from structured representations. The existence of a (pseudo-) inverse element allows the retrieval of $\mathbf{v}, \mathbf{w} \in V_D(N)$ from $\mathbf{v} \circledast \mathbf{w}$ by

$$\bar{\mathbf{v}} \circledast (\mathbf{v} \circledast \mathbf{w}) = \underbrace{\bar{\mathbf{v}} \circledast \mathbf{v}}_{\approx \mathbf{1}} \circledast \mathbf{w} = \tilde{\mathbf{w}} \approx \mathbf{w}. \tag{3.11}$$

In case of exact inverse elements, the right hand side of equation (3.11) becomes an exact equality $\tilde{\mathbf{w}} = \mathbf{w}$. In most cases, however, the result $\tilde{\mathbf{w}}$ is not exactly equal to $\mathbf{w}$, but similar. It is this inherent inexactness of most VSAs that makes them suitable candidates for cognitive modeling (Eliasmith, 2013). On the other hand, it imposes a functional demand for a clean-up memory. A clean-up memory is a mechanism, which maps noisy versions of vectors like $\tilde{\mathbf{w}}$ to their exact counterparts, here $\mathbf{w}$. Therefore, we need to have a set of vectors, which represent established concepts or symbols the system has knowledge of.

Definition 3.12. Let $(V_D(N), \circledast, \oplus, \phi)$ be a VSA (Vector Symbolic Architecture) of dimension D with binding function $\circledast$. We call a finite subset $\vartheta \subsetneq V_D(N)$ a *vocabulary*. A function $\gamma : K^D \longrightarrow \vartheta$ is called a *clean-up memory*, if

1. for any vector $v \in K^D$ we have

$$\phi(\mathbf{v}, \gamma(\mathbf{v})) > \phi(\mathbf{v}, \mathbf{w}), \text{ for any vector } \mathbf{w} \in \vartheta, \gamma(\mathbf{v}) \neq \mathbf{w}.$$

2. for any two similar vectors $\mathbf{v} \neq \tilde{\mathbf{v}} \in K^D, \mathbf{v} \in \vartheta$, i.e., $\tilde{\mathbf{v}} \approx \mathbf{v}$, we have $\gamma(\tilde{\mathbf{v}}) = \mathbf{v}$.

Definition 3.12 states, that the cleaned-up version of a vector is more similar to the original (noisy) version than any other vector in the vocabulary.

3.2 The Semantic Pointer Architecture

In this section, we focus on one particular VSA, namely the SPA, as we will be using it throughout this work. The SPA is an adoption of Plate's previously mentioned HRRs (cf. Example 3.2). Revisiting Definition 3.1, the underlying number field of the SPA is the field of real numbers, i.e., $N \subseteq \mathbb{R}$, the addition $\oplus$ and multiplication $\circledast$ operations are element-wise addition and circular convolution respectively, and the cosine similarity serves as measure of similarity ϕ. We will use $\mathscr{S}(D)$ as a short notation for the D-dimensional SPA $(V_D(\mathbb{R}), \circledast, \oplus, \phi)$. Eliasmith (2013) gives an in-depth description of the SPA. However, we recapitulate some important properties, which will be used later in this work.

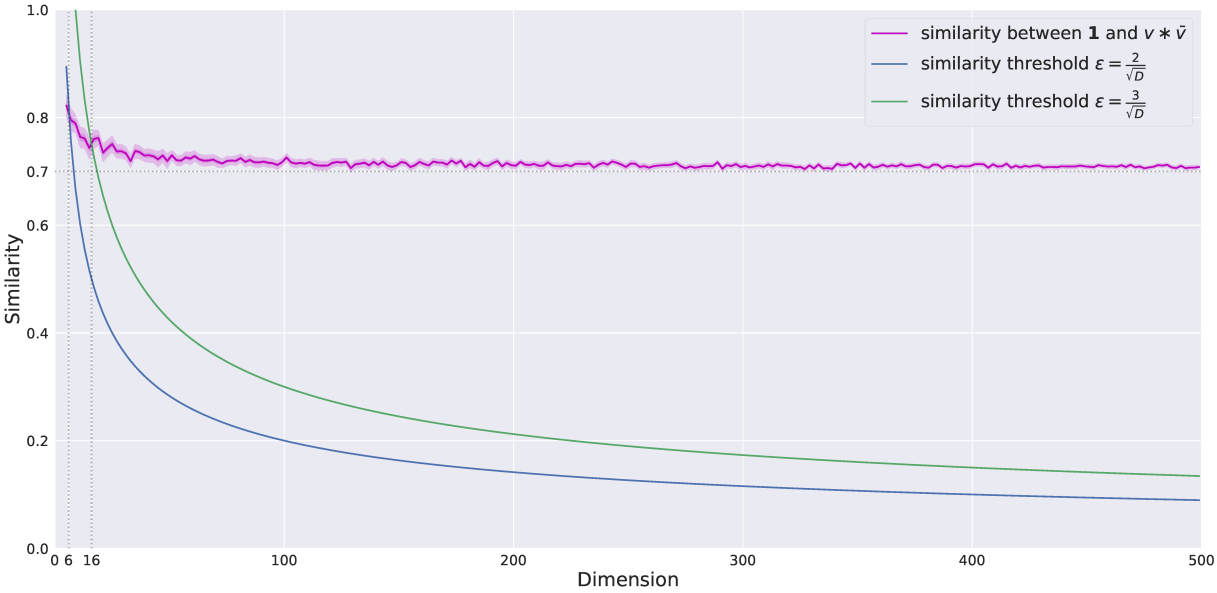


Figure 3.3: Visualization of the similarity $\phi(\mathbf{1}, v \circledast \bar{v})$ between the neutral element $\mathbf{1}$ and the result of applying the pseudo-inverse to different vectors for varying vector dimensions. This plot shows the result of 100 samples compared to the similarity threshold ε.

Lemma 3.13. *Let* $\mathbf{v} = (v_0, \ldots, v_{D-1})$ *be an element of a D-dimensional SPA* $\mathscr{S}(D)$*, i.e.,* $v_0, \ldots, v_{D-1} \in \mathbb{R}$*. The vector* $\bar{\mathbf{v}} = (v_0, v_{D-1}, \ldots, v_1)$ *is a pseudo-inverse element of* $\mathbf{v}$ *with respect to circular convolution, i.e.,* $\mathbf{v} \circledast \bar{\mathbf{v}} \approx \mathbf{1}$.

Here, we skip an explicit proof for this lemma but rather point to T. Plate (1994, Section 3.1.2 and 3.1.3) for a in-depth derivation. However, we visualize the similarity $\phi(\mathbf{1}, \mathbf{v} \circledast \bar{\mathbf{v}})$ between the neutral element $\mathbf{1}$ and the result of applying the pseudo-inverse $\mathbf{v} \circledast \bar{\mathbf{v}}$ compared to the similarity threshold ε (weak and strong version) in Fig. 3.3. Therefore, we randomly chose 100 sample vectors for various dimensions, convolved them with their pseudo-inverse and compared the result to the neutral element $\mathbf{1}$. We observe, that this similarity remains almost constant slightly above 0.7 and already for low vector dimensions ($D > 20$) way above the strong similarity threshold of $\varepsilon = \frac{3}{\sqrt{D}}$.
Lemma 3.13 states that we can find a pseudo-inverse element $\bar{\mathbf{v}}$ for any vector $\mathbf{v}$ in a D-dimensional SPA (given the dimension D is sufficiently large). Although we can also find an exact inverse element $\mathbf{v}^{-1}$ for most vectors $\mathbf{v}$, it is often more useful to work with pseudo-inverses instead of exact inverse elements. We have already seen, that we can use the DFT (Discrete Fourier Transform) and IDFT (Inverse Discrete Fourier Transform) to calculate circular convolution efficiently by element-wise multiplication (this follows from the convolution theorem, see Bracewell, 2000, Chap. 6) in the frequency domain, i.e.,

$$\mathbf{v} \circledast \mathbf{w} = IDFT\left(DFT(\mathbf{v}) \odot DFT(\mathbf{w})\right). \tag{3.5 revisited}$$

Furthermore, the DFT of the convolutive neutral element $\mathbf{1} = (1, 0, \ldots, 0)$ is

$$DFT(\mathbf{1}) = (\exp(i0), \ldots, \exp(i0)) = (1, 1, \ldots, 1). \tag{3.14}$$

This gives us a way of finding an exact inverse element $\mathbf{v}^{-1}$ by

$$DFT(\mathbf{1}) = DFT(\mathbf{v}) \odot DFT(\mathbf{v}^{-1}). \tag{3.15}$$

By denoting the j-th element of the Fourier Vector $DFT(\mathbf{v})$ with $DFT_j(\mathbf{v})$, we get

$$DFT_j(\mathbf{v}) \odot DFT_j(\mathbf{v}^{-1}) = 1 \quad \text{for } j \in \{0, \ldots, D-1\}. \tag{3.16}$$

If we express $DFT_j(\mathbf{v}) \in \mathbb{C}$ in polar coordinates, i.e., $DFT_j(\mathbf{v}) = r_j \exp(i\varphi_j)$, we directly get from equation (3.16) that $DFT_j(\mathbf{v}^{-1}) = \frac{1}{r_j}\exp(-i\varphi_j)$. By using the symmetry property of the real-valued DFT, we can see that the transform $DFT_j(\bar{\mathbf{v}})$ of the pseudo-inverse element $\bar{\mathbf{v}}$ from Lemma 3.13 is the complex conjugate of $DFT_j(\mathbf{v})$, i.e., we can write $DFT_j(\bar{\mathbf{v}}) = r_j \exp(-i\varphi)$ in polar coordinates. From those equations, we can see that the pseudo-inverse $\bar{\mathbf{v}}$ has the same norm as the original vector $\mathbf{v}$, i.e., $\|\mathbf{v}\| = \|\bar{\mathbf{v}}\|$, whereas the norm of the exact inverse $\mathbf{v}^{-1}$ can become significantly larger than the norm of $\mathbf{v}$ in some cases. This is due to the fact that elements of the transform of the pseudo-inverse have the same lengths r_j as the original vector, whereas the transformed elements of the exact inverse have lengths $\frac{1}{r_j}$, which can become significantly large when r_j is close to 0. The relation between the vector norm and the magnitudes in the frequency domain is given by Parseval's theorem (also known as Rayleigh's theorem, see Bracewell, 2000, Chap. 6), which states

$$\|\mathbf{v}\|^2 = \sum_{k=0}^{D-1} |v_k|^2 = \frac{1}{D}\sum_{k=0}^{D-1} |DFT_k(\mathbf{v})|^2 = \sum_{k=0}^{D-1} r_k^2. \tag{3.17}$$

This can lead to additional noise when retrieving vectors from structured representations (cf. equation (3.11)). However, there is a certain class of vectors, for which the pseudo- and exact inverse element coincide.

Definition 3.18. Let $\mathbf{v}$ be a vector in a D-dimensional SPA $\mathscr{S}(D)$ with exact and pseudo-inverse elements $\mathbf{v}^{-1}$ and $\bar{\mathbf{v}}$ respectively. We call $\mathbf{v}$ a *unitary vector*, if and only if $\mathbf{v}^{-1} = \bar{\mathbf{v}}$. We denote the set of unitary vectors by $\mathscr{U} \subset \mathscr{S}(D)$.

Definition 3.19. Let $\mathbf{v}$ be a vector in a D-dimensional SPA $\mathscr{S}(D)$. We define the *convolutive power* by an exponent $p \in \mathbb{R}$ by

$$\mathbf{v}^p := \Re\left(IDFT\left((DFT_j(\mathbf{v})^p)_{j=0}^{D-1}\right)\right),$$

where $\Re$ denotes the real part a of a complex number $a+ib \in \mathbb{C}$.

Unitary vectors take a special role in the SPA as they have some interesting and useful properties.

Lemma 3.20. *Let $\mathscr{U}$ be the set of unitary vectors in the D-dimensional SPA $\mathscr{S}(D)$. The following statements hold*

i *All elements of $\mathscr{U}$ have unit length, i.e., we have $\|\mathbf{u}\| = 1$ for any vector $\mathbf{u} \in \mathscr{U}$.*

ii *$\mathscr{U}$ is closed under convolutive exponentiation, i.e., $\mathbf{u}^p \in \mathscr{U}$ for any $\mathbf{u} \in \mathscr{U}$ and $p \in \mathbb{R}$.*

iii *The product under circular convolution of two unitary vectors is unitary, i.e. $\mathbf{u}_1 \circledast \mathbf{u}_2 \in \mathscr{U}$ for $\mathbf{u}_1, \mathbf{u}_2 \in \mathscr{U}$.*

iv *Convolution with unitary vectors preserves the norm, i.e., $\|\mathbf{v}\| = \|\mathbf{v} \circledast \mathbf{u}\|$ for any $\mathbf{v} \in \mathscr{S}(D), \mathbf{u} \in \mathscr{U}$.*

Proof. To show that unitary vectors have unit length, we directly calculate the first component of convolution $\mathbf{z} := \mathbf{u} \circledast \bar{\mathbf{u}}$ between a unitary vector $\mathbf{u}$ and its pseudo-inverse $\bar{\mathbf{u}}$:

$$z_0 = \sum_{k=0}^{D-1} u_k \bar{u}_{-k \bmod D} = \sum_{k=0}^{D-1} u_k^2 = \|\mathbf{u}\|^2.$$

Since $\mathbf{u}$ is a unitary vector, we have $\mathbf{u}^{-1} = \bar{\mathbf{u}}$ and therefore $\mathbf{u} \circledast \bar{\mathbf{u}} = \mathbf{1}$. Thus, for the first component of $\mathbf{z} := \mathbf{u} \circledast \bar{\mathbf{u}}$, we have

$$1 = z_0 = \|\mathbf{u}\|^2,$$

which gives the first result of the lemma.

To show that the set of unitary vectors $\mathscr{U}$ is closed under convolutive exponentiation, we write each component of $DFT(\mathbf{u})$ for $\mathbf{u} \in \mathscr{U}$ in polar coordinates, i.e., $DFT_j(\mathbf{u}) = r_j \exp(\varphi_j)$. From previous considerations, we know that $DFT_j(\mathbf{u}^{-1}) = \frac{1}{r_j}\exp(-\varphi_j)$ and $DFT_j(\bar{\mathbf{u}}) = r_j \exp(-\varphi_j)$, which gives $r_j = 1$ for $j = 0, \ldots, D-1$ because $\mathbf{u}$ is a unitary vector. Thus, for any $p \in \mathbb{R}$, we have

$$DFT_j(\mathbf{u}^p) = DFT_j(\mathbf{u})^p = r_j^p \exp(ip\varphi) = 1^p \cdot \exp(ip\varphi),$$

which makes $\mathbf{u}^p$ itself a unitary vector, as all magnitudes are 1.
Similarly, we proof that the convolution of two unitary vectors is again unitary. Let $\mathbf{u}_1, \mathbf{u}_2 \in \mathscr{U}$ be unitary vectors, then we have

$$DFT(\mathbf{u}_1 \circledast \mathbf{u}_2) = DFT(IDFT(DFT(\mathbf{u}_1) \odot DFT(\mathbf{u}_2))) = DFT(\mathbf{u}_1) \odot DFT(\mathbf{u}_2).$$

Writing both $DFT_j(\mathbf{u}_1) = r_{1j}\exp(\varphi_{1j})$ and $DFT_j(\mathbf{u}_2) = r_{2j}\exp(\varphi_{2j})$ in polar coordinates, we get

$$DFT_j(\mathbf{u}_1 \circledast \mathbf{u}_2) = r_{1j} \cdot r_{2j} \cdot \exp(\varphi_{1j} + \varphi_{2j}) = 1 \cdot \exp(\varphi_{1j} + \varphi_{2j}),$$

as we have $r_{1j} = 1$ and $r_{2j} = 1$ for all magnitudes since both $\mathbf{u}_1$ and $\mathbf{u}_2$ are unitary vectors. This in turn makes their product under convolution unitary as well.
To proof that convolution with unitary vectors preserves the norm, we use Parseval's theorem (cf. equation (3.17)) again. For any vector $\mathbf{v} \in \mathscr{S}(d)$ and a unitary vector $\mathbf{u} \in \mathscr{U}$, we denote $\mathbf{z} := \mathbf{v} \circledast \mathbf{u}$ and get

$$\|\mathbf{v} \circledast \mathbf{u}\|^2 = \sum_{k=0}^{D-1} |z_k|^2 = \frac{1}{D}\sum_{k=0}^{D-1} |DFT_k(\mathbf{z})|^2 = \frac{1}{D}\sum_{k=0}^{D-1} |DFT_k(\mathbf{v}) \cdot DFT_k(\mathbf{u})|^2. \tag{3.21}$$

Writing $DFT_k(\mathbf{v}) = r_{vk}\exp(i\varphi_{vk})$ and $DFT_k(\mathbf{u}) = r_{uk}\exp(i\varphi_{uk})$ in polar coordinates, with $r_{uk} = 1$ for $k = 0, \ldots, D-1$ as $\mathbf{u}$ is unitary, equation (3.21) transforms to

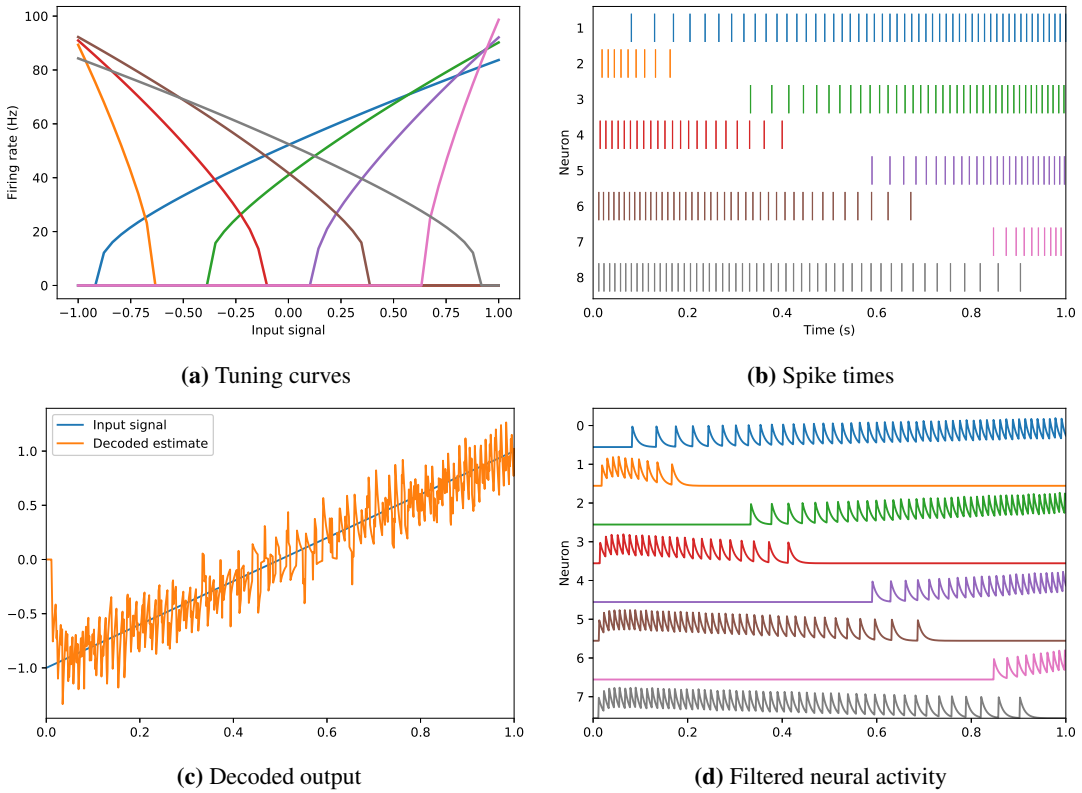
$$\|\mathbf{v} \circledast \mathbf{u}\|^2 = \frac{1}{D}\sum_{k=0}^{D-1} |r_{vk}\exp(i(\varphi_{vk} + \varphi_{uk}))|^2 = \frac{1}{D}\sum_{k=0}^{D-1} |r_{vk}|^2 = \frac{1}{D}\sum_{k=0}^{D-1} |DFT_k(\mathbf{v})|^2 = \|\mathbf{v}\|^2.$$

□

3.3 The Neural Engineering Framework

In this section, we make a short excursion and give a brief overview of the NEF (Neural Engineering Framework), as we will be making use of it in forthcoming chapters. The NEF (Eliasmith and C. H. Anderson, 2003) is a mathematical theory, which provides a set of methods to construct biologically plausible, large-scale neural models. These methods can be divided into the three main principles of the NEF: *representation*, *transformation* and *dynamics*. The Nengo[1] software suite is a python library, which implements the NEF's principles (Bekolay et al., 2014). Nengo has been used to build a variety of neural models, e.g., models of the Basal Ganglia system (T. C. Stewart et al., 2010; T. Stewart et al., 2012) and Spaun (Semantic Pointer Architecture Unified Network), a large-scale, functional model of the brain, which is able to perform eight cognitive tasks (Eliasmith et al., 2012). Furthermore, Nengo has been used to interface neural models with physical, neuromorphic hardware systems and robots (Conradt et al., 2015; T. C. Stewart et al., 2016). Here, we give a brief introduction to the NEF's principles and refer to Eliasmith and C. H. Anderson (2003), Eliasmith (2013), and Bekolay et al. (2014) for more details.

[1] Applied Brain Research Inc. (2018). *The Nengo neural simulator*. URL: https://www.nengo.ai/ (visited on 2018-04-05).



(a) Tuning curves

(b) Spike times

(c) Decoded output

(d) Filtered neural activity

Figure 3.4: The representation principle of the NEF. Images adapted from Bekolay et al. (2014).

3.3.1 Representation

The first principle of the NEF, representation, provides mathematical tools to encode information, namely time-varying real-valued vectors, in the activity of neural populations. It is based on the assumption, that neurons have a "preferred direction vector" in the represented space, each neuron responds most strongly to. This assumption is grounded by the findings of Georgopoulos et al. (1989) that each neuron in motor cortex of rhesus monkeys has a different preferred arm direction. The NEF expands this idea to neural representations in general. Let A be a population of $N \in \mathbb{N}$ neurons encoding a subset V of a real-valued vector space, i.e., $V \subseteq \mathbb{R}^n$. Given a function $\mathbf{x} : \mathbb{R} \longrightarrow V$, we can write the activity a_i of the i-th neuron in a neural population encoding a time-varying vector $\mathbf{x}(t)$ as a spike train, i.e., a sum of delta functions

$$a_i(\mathbf{x}(t)) = \sum_{j=1}^{m_i} \delta(t - t_j) = G_i(\underbrace{\alpha_i \langle \mathbf{e}_i, \mathbf{x}(t) \rangle + J_i}_{=:c}) \quad \text{for } 1 \leq i \leq N, \tag{3.22}$$

where G_i is the spiking neural non-linearity, α_i is the gain of the neuron, $\mathbf{e}_i$ is the neuron's preferred direction or encoding vector and J_i is a bias current to account for neural background activity and t_j are the m_i spike-times of the i-th neuron. Notably, the current flowing into the cell is completely determined by c, whereas the spiking behavior of the neuron model is represented by the non-linear function G_i. The input current c and therefore the NEF's encoding process is independent of particular spiking neuron models.

To decode the input values $\mathbf{x}(t)$ back out of the neural population A, the spike train is convolved with an exponentially decaying filter $h : \mathbb{R} \longrightarrow \mathbb{R}$ to simulate the process of neurons generating post-synaptic current after spiking (cf. Fig. 3.4d) resulting in

$$\tilde{a}_i(\mathbf{x}(t)) = \sum_{j=1}^{m_i} h(t) * \delta(t - t_j) = \sum_{j=1}^{m_i} h(t - t_j). \tag{3.23}$$

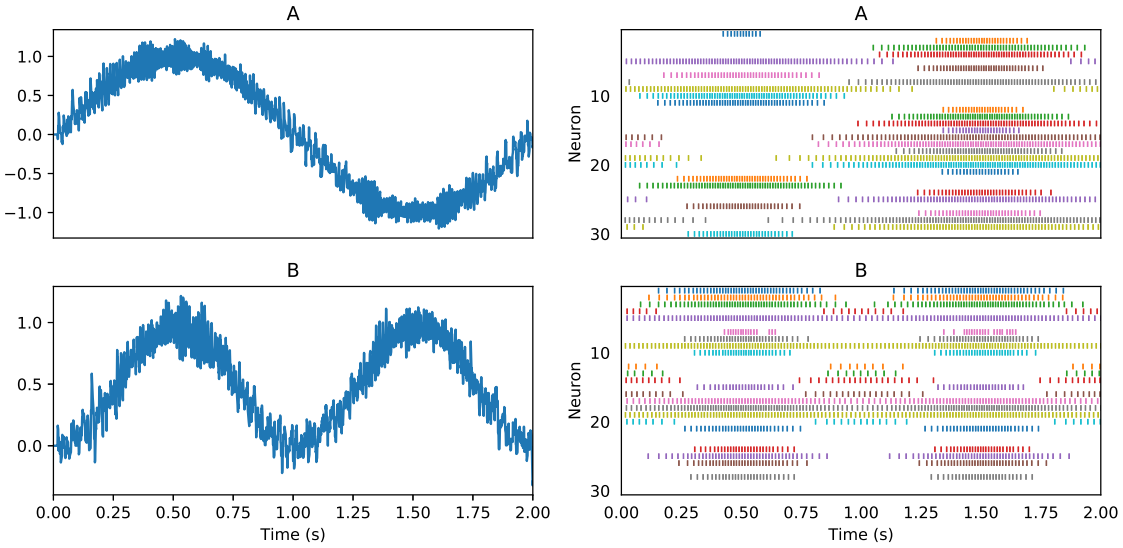


Figure 3.5: The transformation principle of the NEF. Images adapted from Applied Brain Research Inc. (2018).

A simple model of an exponential decaying filter is the function $h : \mathbb{R} \longrightarrow \mathbb{R}, t \longmapsto e^{-t/\tau_p}$, where τ_P denotes the post-synaptic time constant. We obtain an estimation $\hat{\mathbf{x}}(t)$ of the original input $\mathbf{x}(t)$ as a weighted sum with some decoder values $\mathbf{d}_i$

$$\hat{\mathbf{x}}(t) = \sum_{i=1}^{N} \tilde{a}_i(\mathbf{x}(t))\,\mathbf{d}_i. \tag{3.24}$$

To calculate the optimal decoders $\mathbf{d}_i$, we need to minimize the error between input $\mathbf{x}(t)$ and decoded output $\hat{\mathbf{x}}(t)$

$$E = \int \left(\mathbf{x}(t) - \sum_{i=1}^{N} \tilde{a}_i(\mathbf{x}(t))\,\mathbf{d}_i \right)^2 \mathrm{d}\mathbf{x}(t). \tag{3.25}$$

Nengo solves for the decoders $\mathbf{d}_i$ by default using least squares optimization (Eliasmith, 2013)[Appendix B1]. Figure 3.4 visualizes this encoding process for $V = [-1,1] \subset \mathbb{R}$ and a population of 8 neurons. Figure 3.4a shows the tuning curves of individual neurons, which define how these neurons respond to specific input values. Equation (3.22) is depicted in Fig. 3.4b, which shows a raster plot of the neurons' spike times based on the input signal shown in Fig. 3.4c. Figure 3.4d, which shows the filtered neural activity for each neuron, visualizes Equation (3.23). Finally, Fig. 3.4c depicts the original input value as well as the estimated output of the neural populations' activity (cf. Equation (3.24)). Note, that the neural population's decoded output is only a noisy approximation of the original input value, whose accuracy can be improved by increasing the number of neurons in the population.

3.3.2 Transformation

The second main principle of the NEF, *transformation*, provides the mathematical tools to compute functions across connections between populations of neurons. Let A resp. B be populations of N resp. M neurons encoding a time-varying vector $\mathbf{x}(t) \in V \subset \mathbb{R}^n$ resp. $\mathbf{y}(t) \in W \subset \mathbb{R}^m$ according to the representation principle and a function $f : V \longrightarrow W \subset \mathbb{R}^m$. In order to approximate the function f across a connection from population A to population B, we use the tools of the representation principle, but we calculate a different set of decoder values $\mathbf{d}_i^f$ for population A by minimizing the error

$$E = \int \left(f(\mathbf{x}(t)) - \sum_{i=1}^{N} \tilde{a}_i(\mathbf{x}(t))\,\mathbf{d}_i^f \right)^2 \mathrm{d}\mathbf{x}(t). \tag{3.26}$$

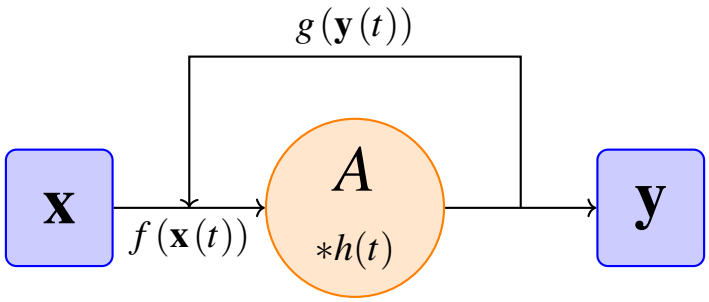


Figure 3.6: The dynamics principle of the NEF for recurrent connections.

Given encoders $\mathbf{e}_j^B$ and gain α_j^B for $1 \leq j \leq M$ of population B, we can derive a weight matrix for the connection from A to B approximating the function f by

$$w_{ij} = \alpha_j^B \mathbf{d}_i^f L \mathbf{e}_j^B \quad \text{for } 1 \leq i \leq N \text{ and } 1 \leq j \leq M, \tag{3.27}$$

where L is a $M \times N$ linear operator. Here, the NEF makes the assumption, that connection weights can be factored into encoders, decoders and a linear transform. Figure 3.5 visualizes the NEF's transformation principle for $V = W = [-1,1] \subset \mathbb{R}$, two neural populations A, B containing 30 neurons each. The left panel of plots shows the populations' decoded outputs whereas the right panel depicts the neurons' spike times. Population A uses the representation principle to encode a sine function, whereas the transformation principle was used to calculate the function $f : V \longrightarrow W, x \longmapsto f(x) = x^2$ across the connection from A to B.

3.3.3 Dynamics

The third main principle of the NEF, *dynamics*, provides a set of mathematical tools to implement dynamical systems in neural populations through recurrent connections. Let A be a population of neurons with an incoming connection approximating the function $f : V \longrightarrow W \subset \mathbb{R}^m$ and a recurrent connection approximating the function $g : W \longrightarrow W$ (cf. Fig. 3.6). Thus, the overall function the population is approximating is

$$\mathbf{y}(t) = h(t) * (f(\mathbf{x}(t)) + g(\mathbf{y}(t))) \tag{3.28}$$

with exponential decaying filter function $h : \mathbb{R} \longrightarrow \mathbb{R}, t \longmapsto e^{-t/\tau}$. By applying the Laplace transform to Equation (3.28), we get

$$\mathbf{Y}(s) = \frac{1}{1+s\tau} \left(F(\mathbf{X}(s)) + G(\mathbf{Y}(s))\right). \tag{3.29}$$

We can rearrange Equation (3.29) to

$$s\mathbf{Y}(s) = \frac{G(\mathbf{Y}(s)) - \mathbf{Y}(s)}{\tau} + \frac{F(\mathbf{X}(s))}{\tau}. \tag{3.30}$$

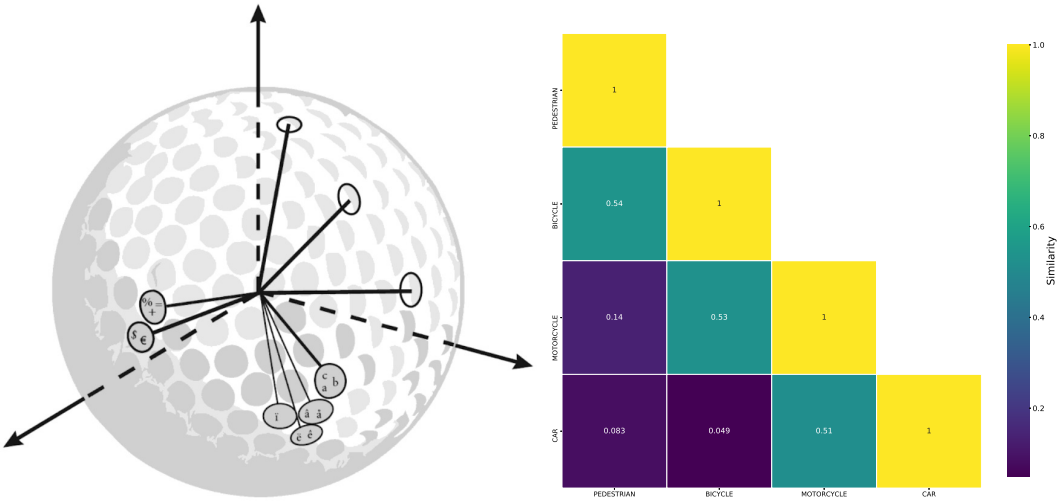
Transforming back leads to the differential equation

$$\frac{\partial \mathbf{y}(t)}{\partial t} = \frac{g(\mathbf{y}(t) - y)}{\tau} + \frac{f(\mathbf{x}(t))}{\tau}. \tag{3.31}$$

Thus, to construct a neural model approximating a differential equation of the form

$$\frac{\partial \mathbf{y}(t)}{\partial t} = a(\mathbf{y}(t)) + b(\mathbf{x}(t)) \tag{3.32}$$

with functions $a : W \longrightarrow W$ and $b : V \longrightarrow W$, the first two principles of the NEF can be used to create a neural population of the form as shown in Fig. 3.6. By setting the functions $g(\mathbf{y}(t)) = \tau a(\mathbf{y}(t)) + \mathbf{y}(t)$ and $f(\mathbf{x}(t)) = \tau b(\mathbf{x}(t))$, we obtain a neural model approximating the desired dynamical system described by the differential Equation (3.32).



(a) **(b)**

Figure 3.7: Aspects of vector vocabularies: (a) "Conceptual golf ball" depicting the idea of semantic vectors. Image source: Eliasmith (2013). (b) Cosine similarities in a small, manually engineered vector vocabulary of dimension 256.

3.4 Cognitive Modeling with Vector Symbolic Architectures

In this section, we give a brief introduction of how we can use the theory shown in sections 3.1 and 3.2 to represent (structured) information using VSAs. We give a general overview of different ways to establish vocabularies containing atomic vectors and how we can build more complex representations from those elementary ingredients. Gallant and Okaywe (2013) refer to these two steps as the first two of three stages for generating structured vector representations: the *pre-processing stage* (generating a vocabulary; see also section 4.1) and the *representation generation stage* (building structured representations from the vocabulary; see also 4.2). Furthermore, we will see how these representations can be implemented in SNNs using the principles of the NEF described in section 3.3.

3.4.1 Vocabularies

Let $\vartheta \subset \mathscr{S}(D)$ be a vocabulary in the D-dimensional SPA, where each vector $v \in \vartheta$ represents one item, i.e., a symbol, word or concept, in the representational space. The content and size, i.e., the items of interest to be represented in such a vocabulary and their number as well as the way the representing vectors are established is highly task-dependent. In its simplest form, all vectors in the vocabulary are chosen at random. This approach is feasible due to the properties of high-dimensional vector spaces (cf. Theorem 3.6 in section 3.1) that the probability of randomly chosen vectors being dissimilar grows with vector dimension. Therefore, we have low probability of unintentionally confusing two different concept vectors. However, this approach does not capture any similarities between items being represented. Ideally, the goal is that the similarity between vectors in the vocabulary ϑ somewhat reflects the similarity between represented items. Figure 3.7a depicts this idea: one subset of the space is assigned to vectors representing letters whereas another subset contains vectors representing special characters. For a small number of items, the simplest way to create a vocabulary respecting some kind of similarity is to manually engineer the desired properties from randomly chosen vectors. Let us assume we want to derive representative vectors for four different classes of traffic participants, namely *pedestrian*, *bicycle*, *motorcycle*, and *car*. A simple structured vocabulary could be constructed in the following way

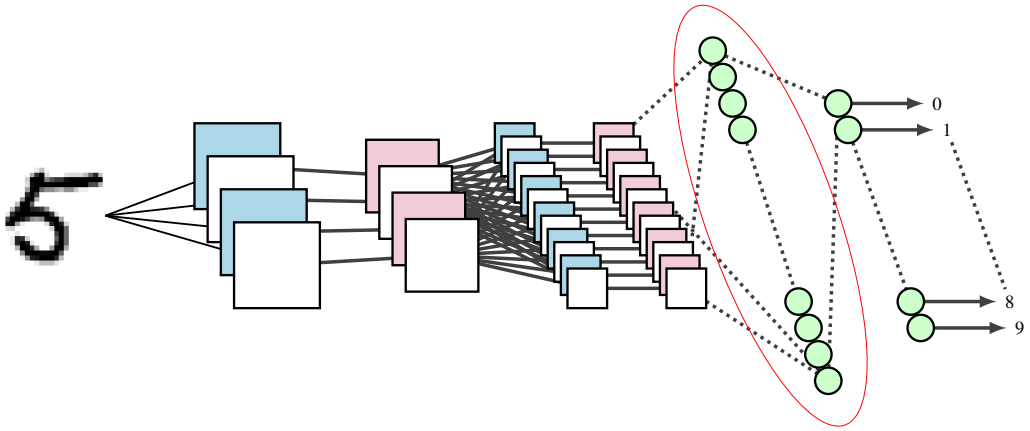


Figure 3.8: A schematic visualization of a CNN network architecture with the second to last layer, whose activity can be considered a compressed, lower-dimensional representation of the high-dimensional visual input, highlighted by a red ellipsis.

$$\begin{aligned}
\textbf{PEDESTRIAN} &:= \textbf{DRIVE} \circledast \textbf{MUSCLE} + \textbf{ACTUATOR} \circledast \textbf{LEG} \circledast \textbf{TWO} \\
\textbf{BICYCLE} &:= \textbf{DRIVE} \circledast \textbf{MUSCLE} + \textbf{ACTUATOR} \circledast \textbf{WHEEL} \circledast \textbf{TWO} \\
\textbf{MOTORCYCLE} &:= \textbf{DRIVE} \circledast \textbf{MOTOR} + \textbf{ACTUATOR} \circledast \textbf{WHEEL} \circledast \textbf{TWO} \\
\textbf{CAR} &:= \textbf{DRIVE} \circledast \textbf{MOTOR} + \textbf{ACTUATOR} \circledast \textbf{WHEEL} \circledast \textbf{FOUR},
\end{aligned}$$

with **DRIVE**, **MUSCLE**, **MOTOR**, **ACTUATOR**, **LEG**, **WHEEL**, **TWO** and **FOUR** $\in \mathscr{S}(D)$ all being atomic vectors chosen at random. Figure 3.7b shows the similarities in an example vocabulary in $\mathscr{S}(256)$ constructed in the aforementioned way. This vocabulary is designed to map simple motion properties into a vector representation. Thus, this manually engineered vocabulary yields the desired property that vulnerable road users (in this case pedestrians and bicyclists) are more similar to each other than motor vehicles, whereas there is also similarity between different types of cyclists. This approach allows the engineer to encapsulate almost any desired kind of similarity without losing transparency during the encoding process, meaning that the reason for certain similarity artifacts is traceable. However, not only does this approach become impracticable with increasing number of items in the vocabulary, it is also prone to biases imposed by the preferences of the human engineer.

A more sophisticated way to create a vector vocabulary is to learn it automatically from data. In contrast to purely random vocabularies, the idea here is that the learning system is able to capture the intrinsic similarity between objects in the vector representation. Again, the choice of learning paradigm (supervised vs. unsupervised) and architecture, e.g., CNNs or SOMs, depends not only on the given task, but also on the kind of similarity the vectors should encapsulate. This can be visual similarity (e.g., items or objects that "look" similar), auditory similarity (e.g., objects producing similar sound), semantic similarity (e.g., words with similar meaning) or similarity in characteristics (e.g., similar motion characteristics).

A similar approach to derive vectors representing digits from 0 to 9 was used in Eliasmith et al. (2012) for the network performing the visual digit recognition task as part of the larger Spaun model, which was derived by training a DBN consisting of four Resctricted Boltzmann Machine layers. This approach of using the activity of the second to last layer to generate atomic representational vectors can be generalized to any neural network for classification with dense layers at the end. A typical way of learning a vocabulary whose vectors conserve visual similarity in supervised fashion are CNNs. These network architectures are inspired by the human visual cortex and compress high-dimensional visual input to lower dimensional representations. They usually consist of a series of convolutional and pooling layers followed by one or more fully connected layers, where the last layer provides the classification result (cf.

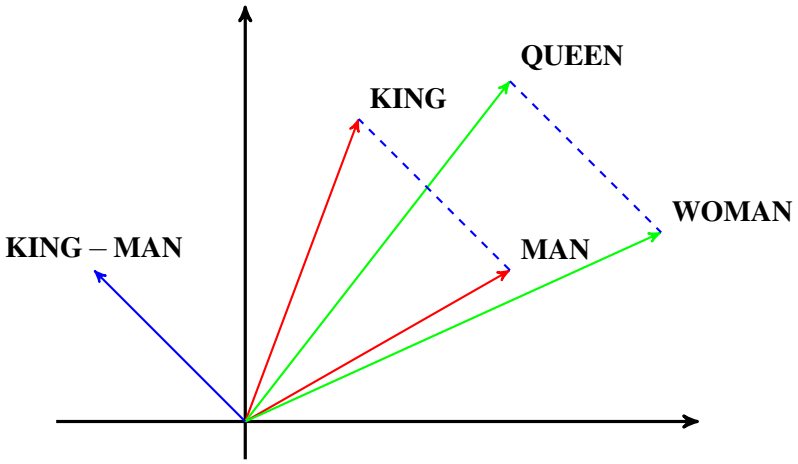


Figure 3.9: A typical example illustrating the semantic similarity structure between vectors representing the words *king*, *queen*, *man* and *woman* learned by Word2Vec allowing algebraic manipulation of the encoded entities. Image inspired from Mikolov et al. (2013c).

Fig. 3.8). The activity of the second to last layer (the last fully-connected one before the actual classification) for each known class in the data set can be considered a representational vector for the current visual input. A simple way of getting a representative vector for each class is simply calculating the element-wise, normalized mean over all examples in the test set. We employ this approach in section 4.1.3 to generate vocabularies encapsulating the visual similarity of traffic signs and traffic participants.
The aforementioned approach to create vector vocabularies encapsulating similarity by using neural networks, which are trained in supervised fashion, is suitable for capturing visual or auditory similarity structure in a vectors. The generation of representational vectors for related or similar words or, more generally, semantic similarity in language is a problem often referred to as word embedding, which is a research question related to computational NLP (Natural Languange Processing). The most successful approaches such as *word2vec* (Mikolov et al., 2013a; Mikolov et al., 2013b) or *GloVe* (Pennington et al., 2014) typically use large corpora of text as input data and are trained in unsupervised fashion based on co-occurrence statistics of words. The objective of those procedures aims at maximizing the dot product of word vectors that appear often in similar contexts while minimizing it for negative samples, which do not occur in the training data and thus are sampled at random. For a given sentence $S = \{w_1, \ldots, w_n\}$, a context $C(w)$ of a word $w = w_i$ of size k is a dynamic window of the form $C(w) = \{w_{i-k}, \ldots, w_{i-1}, w_{i+1}, \ldots, w_{i+k}\}$. The underlying assumption is that words sharing many contexts are similar to each other such that automated training with the aforementioned objective will produce a good word embedding. Surprisingly, these learning procedures lead to linguistic regularities and algebraic relations between word vectors (Mikolov et al., 2013c). A common example is visualized in Fig. 3.9: subtracting the vector representing **MAN** from the one representing **KING** and adding the vector representing **WOMAN** results in a vector most similar to the one representing **QUEEN**, i.e., $\textbf{KING} - \textbf{MAN} + \textbf{WOMAN} \approx \textbf{QUEEN}$. However, Goldberg and O. Levy (2014) and O. Levy et al. (2015) point out that the formal reasons for successful learning of word embeddings of those approaches are not well understood.

3.4.2 Encoding structure

In section 3.4.1, we have already seen that there are many ways to create vector vocabularies, that encode different types of similarity. However, the strength of VSAs lies in the possibility to build structured representations from atomic vectors. Unordered sets can be represented by simply adding up atomic vectors. Through the properties of the similarity measure, the sum is similar to each vector contained in the sum. The most common approach to incorporate structure is to use so-called role-filler pairs (Gayler,

2003). Such a pair is combined via the VSA's binding operation, where one vector of the pair takes the role of a variable while the other one can be considered the value of the variable. We have already seen a simple example of this approach in section 3.4.1, where a small vocabulary representing four different classes of traffic participants was manually created. In NLP (Natural Languange Processing) applications, this approach is useful when building up vectors representing phrases from a word vector vocabulary. The phrase *The dog chases the boy* could be encoded in a vector as

$$\mathbf{agent} \circledast \mathbf{dog} + \mathbf{verb} \circledast \mathbf{chase} + \mathbf{theme} \circledast \mathbf{boy},$$

assuming we already have a vocabulary containing atomic vectors representing those items. Another typical example of this approach is the encoding of relations between items, such as "**A** is the mother of **B**". This relation could be represented in vectors through

$$m(\mathbf{A}, \mathbf{B}) = \mathbf{motherof} + \mathbf{mother} \circledast \mathbf{A} + \mathbf{child} \circledast \mathbf{B}.$$

The strength of VSAs is that the vectors representing such relations can subsequently be manipulated and combined using the VSA's algebraic operations. Kanerva (2000) shows that this approach can be used to create a system, that is able to infer one relation, which is induced by another relation, through learning from example. Their example features the aforementioned *mother of* relation, which induces the *parent of* relation

$$p(\mathbf{A}, \mathbf{B}) = \mathbf{parentof} + \mathbf{parent} \circledast \mathbf{A} + \mathbf{offspring} \circledast \mathbf{B},$$

i.e., we have $m(\mathbf{A}, \mathbf{B}) \Longrightarrow p(\mathbf{A}, \mathbf{B})$. Combining several examples

$$M = \sum_{i=0}^{n} m(\mathbf{A}_i, \mathbf{B}_i) \circledast p(\mathbf{A}_i, \mathbf{B}_i)$$

yields a transition vector, which gives $M \circledast m(\mathbf{X}, \mathbf{Y}) \approx p(\mathbf{X}, \mathbf{Y})$ for unseen examples **X** and **Y**. However, this approach is based on the self-cancellation property (i.e., $\mathbf{X} \circledast \mathbf{X} = \mathbf{1}$) in certain VSAs (in particular BSCs), as it leads to M containing the vector $\mathbf{motherof} \circledast \mathbf{parentof} + \mathbf{mother} \circledast \mathbf{parent} + \mathbf{child} \circledast \mathbf{offspring}$ (see Kanerva, 2000, for details). A similar approach was used in Kleyko et al. (2015) for a prototype of concept learning. Rasmussen and Eliasmith (2011) employ a similar approach to encode a rule for inductive learning in a transition vector using the VSA's algebraic operations to solve RPMs (Raven's Progressive Matrices). The authors represent consecutive cells in an RPM and encode the transition from one cell to another in a particular transition vector and, from a sequence of those, create a general transition vector encapsulating the general rule for going to the next cell in the matrix. They implement these vectors as well as their inductive learning system in a SNN model, a process we will also briefly discuss in the next section,

3.4.3 Implementation in Spiking Neural Networks

In this section, we show how VSAs and in particular the SPA can be implemented in SNNs. In section 3.3.1, we have already discussed the representation principle of the NEF, that allows to encode time-varying vectors in the activity of populations of spiking neurons using Equation (3.22). Given that all representations of entities in the SPA are vectors, we can directly apply Equation (3.22) to encode any vector representation generated using the principles and algebraic operations of the SPA. Similarly, we can employ Equation (3.24) to decode back out the original vector from the neurons' activities. However, to manipulate the encoded vectors into structured representations using the SPA's algebraic operations, we also need to be able to implement these operations in networks of spiking neurons. Again, the tools of the NEF allow this implementation. The implementation of element-wise addition of vectors is straightforward: assuming we have created populations A and B encoding the vectors **v** and **w** respectively using

Equation (3.22), we can now use the transformation principle and Equation (3.26) to connect *both* populations A and B to a third population C with both connections approximating the identity function. The population C receiving input from both populations A and B will end up representing an approximation of the sum $\mathbf{v}+\mathbf{w}$.
Similarly, we can use the NEF's principles to implement the SPA's binding operation, circular convolution, in a network of spiking neurons. Revisiting Equation (3.5), we can write the circular convolution $\mathbf{z}=\mathbf{v}\circledast\mathbf{w}$ of two vectors $\mathbf{v},\mathbf{w}$ as

$$\mathbf{z}=\mathbf{v}\circledast\mathbf{w}=IDFT\left(DFT(\mathbf{v})\odot DFT(\mathbf{w})\right). \tag{3.5 revisited}$$

We can consider the DFT and IDFT as linear functions given by the matrices

$$W=\frac{1}{\sqrt{D-1}}\left(\zeta_D^{i\cdot j}\right)_{i,j=0}^{D-1},\qquad W^{-1}=\frac{1}{\sqrt{D-1}}\left(\zeta_D^{-i\cdot j}\right)_{i,j=0}^{D-1} \tag{3.33}$$

and thus write circular convolution as matrix multiplication

$$\mathbf{z}=\mathbf{v}\circledast\mathbf{w}=W^{-1}\left((W\mathbf{v})\cdot(W\mathbf{w})\right). \tag{3.34}$$

Applying the NEF's transformation principle, we can implement circular convolution in a spiking neuron substrate. To finally be able to unbind vectors, i.e., calculate $\mathbf{v}=\mathbf{z}\circledast\bar{\mathbf{w}}$ using the pseudo-inverse element $\bar{\mathbf{w}}=(w_0,w_{D-1},w_{D-2},\ldots,w_1)$ (cf. Lemma 3.13), we can adapt Equation (3.34). The function transforming a vector into its pseudo-inverse element is a simple permutation of elements given by the matrix

$$C_{pinv}=\begin{pmatrix}1&0&\cdots&0&0\\0&0&\cdots&0&1\\0&0&\cdots&1&0\\\vdots&\vdots&\iddots&\vdots&\vdots\\0&1&0&\cdots&0\end{pmatrix}, \tag{3.35}$$

which allows us to transform Equation (3.34) into

$$\mathbf{v}\approx\mathbf{z}\circledast\bar{\mathbf{w}}=W^{-1}\left((W\mathbf{z})\cdot(W\cdot C_{pinv}\mathbf{w})\right). \tag{3.36}$$

The implementation of a SNN performing a biologically plausible version of a cleanup memory employing the NEF is shown in T. C. Stewart et al. (2011). Thus, applying the principles of the NEF, we can represent the representational vectors of the SPA in the activity of populations of spiking neurons and implement its algebraic operations in networks of spiking neuron populations. A subtle detail worth noting with regard to implementing VSAs in SNNs is that the vectors used for representing concepts or entities of interest are *not* vectors of neural activity. That is, there are two distinct representational vector spaces: the space of neural activities (i.e., measurable properties like spike patterns) and the vector space used to generate the vocabulary and structured representations vectors, which in turn are represented by the activities of populations in the neural space (see Eliasmith, 2013, for further details).
There are several reasons for implementing cognitive architectures like the SPA in a spiking neuron substrate. Regarding the application of vector-based representations in cognitive modeling, it makes sense to consider an implementation in neural activity, since the biological systems able to perform cognitive tasks are also systems of neural networks. Hence, when generating models of some cognitive function or behavior such as the model for inductive rule generation applied to RPMs in Rasmussen and Eliasmith (2011), implementation in a spiking neuron substrate allows to compare the model's neural activity with neural activity measured in the brain of human subjects performing a similar task. Such a comparison enables more detailed model analysis and improvements by allowing researchers to draw inspirations from actual biological systems. Considering applications of the SPA in domains such as automated driving as proposed in later chapters 5, 6 and 7, the possibility of implementation in a spiking neuron

substrate is particularly interesting in terms of energy-efficiency. As discussed in section 2.1.3, there exists a growing number of neuromorphic hardware devices dedicated to efficiently processing networks of spiking neurons. Such dedicated computing hardware allows to run SNNs more efficiently than traditional computing hardware, which often requires several orders of magnitude less energy (Hunsberger and Eliasmith, 2016). This is particularly interesting in mobile applications such as robotic systems or, more particularly, automated driving, which poses increasingly strong restrictions on the energy-budget of individual components due to growing setups of both, on-board sensors and computing hardware. Since these novel computing devices are not yet available as commercial products but rather prototypes developed in academic research, the deployment of the models developed in this thesis on neuromorphic computing hardware integrated into a vehicle's architecture is unfortunately out of scope of this thesis.

3.5 Summary

In this chapter, we have introduced the theoretical background and mathematical properties of Vector Symbolic Architectures, which is the basis for subsequent chapters. Furthermore, we established a formal, mathematical notion for VSAs in general (section 3.1), which, to the best or our knowledge, is not available in the literature except for treatments of particular instantiations of VSAs such as in T. Plate (1994), Gayler (1998), and Kanerva (2009). Then, we shifted our focus to one particular VSA, the SPA, presenting its most important aspects and properties with respect to subsequent chapters of this thesis. Furthermore, we gave a brief description of the Neural Engineering Framework, which is typically used to implement large-scale neural models of spiking neurons such as the one presented in Eliasmith et al. (2012). Finally, we have introduced the basic principles of cognitive modeling using VSAs in general and the SPA in particular. We showed several general approaches to generate vocabularies of atomic vectors, which form the basis of more complex structured representations, which are built from the vocabulary using the VSA's algebraic operations. Finally, we connected the mathematical description and theory of both, the VSAs and the NEF by depicting how the NEF can be used to implement the SPA in a spiking neuron substrate, which offers interesting possibilities regarding energy-efficiency in mobile applications such as automated driving. We will use the theory and findings shown here in subsequent chapters to derive vector-based representations of automotive data measured and collected in actual driving situations to tackle tasks like classifying the current driving context or prediction the future development of the scene based on past observations.

4 Distributed representations of automotive scenes

In chapter 2 and especially in section 2.3.2, we have seen that, already today, there is a plethora of ADAS (Advanced Driver Assistance Systems) in intelligent vehicles. In future vehicles, the number of modules tackling different sub-tasks necessary to enable (semi-) autonomous driving and to interact with humans inside and outside the car will increase even more. Given the complexity of the physical world and the recent success of DNNs in diverse applications, a substantial amount of such modules could be data-driven with increasingly large neural networks under the surface. In a worst case scenario, each of these systems will encapsulate its own representation of knowledge about the data it processes in complete separation from other, potentially related, systems. Typically, the representations used rely completely on numerical values and lack possibilities to be enriched or combined with symbol-like representations. On the other hand, increasingly deep neural network architectures are not only hungry for data to generalize sufficiently beyond the examples they have been trained on, but also tend to require a substantial amount of computational resources. Although this aspect is more severe for the training process, it becomes more important for mobile applications such as automated vehicles during the deployment phase.

In this thesis, we propose a novel representation for automotive scenes based on modern cognitive modeling techniques, namely the SPA. The SPA is one particular example from a family of cognitive architectures commonly referred to as VSAs (see section 2.2.3 and chapter 3 for further details). One of the key components of these cognitive architectures is to use high-dimensional vectors for representation. This representational approach offers several desirable features. High-dimensional vectors are one variant of distributed representations in the sense that information is captured over all dimensions of the vector instead of one single number. This aspect makes distributed representations more robust to noise in the sense that a few noisy entries influence the overall information carried by the vector less compared to low-dimensional representations. Furthermore, vector representations allow to encode both, symbol-like and numerical structures in a similar and unified way. Additionally, the algebraic operations enable manipulation and combination of represented entities into structured representations. One potential advantage of this approach is that the number of dimensions remains fixed independent of the number of entities combined through the architecture's algebraic operations. Finally, vectors are a suitable representational substrate to be used in combination with neural networks. On the one hand, vectors are a natural input to classic ANNs, but they also offer the possibility to be efficiently implemented in SNNs using the principles of the NEF (see Eliasmith, 2013, but also section 3.4.3). Given a widespread implementation of the representations proposed here in combination with SNNs as algorithmic substrate within intelligent vehicles, the latter offers the potential to deploy such neural representations on dedicated neuromorphic hardware (cf. section 2.1.3). Although neuromorphic computing hardware as well as the corresponding neural algorithms are mainly used in academic research and often lack the technical maturity required by industrial applications, they show promise to be an energy-efficient option for future automated vehicles once reaching the required level of maturity.

In this chapter, we introduce our proposed approach to encapsulate high-level information about automotive scenes in high-dimensional, semantic vectors using the SPA as representational substrate. For this encoding phase, we follow the first two stages, namely *preprocessing* and *representation generation*, of the three-stages process established in Gallant and Okaywe (2013). The third and final stage, *output computation*, will be the subject of subsequent chapters, where we investigate concrete applications and use cases. The preprocessing stage is the step of creating a suitable vector vocabulary, whereas the representation generation stage is the process of building up structured representations from the atomic

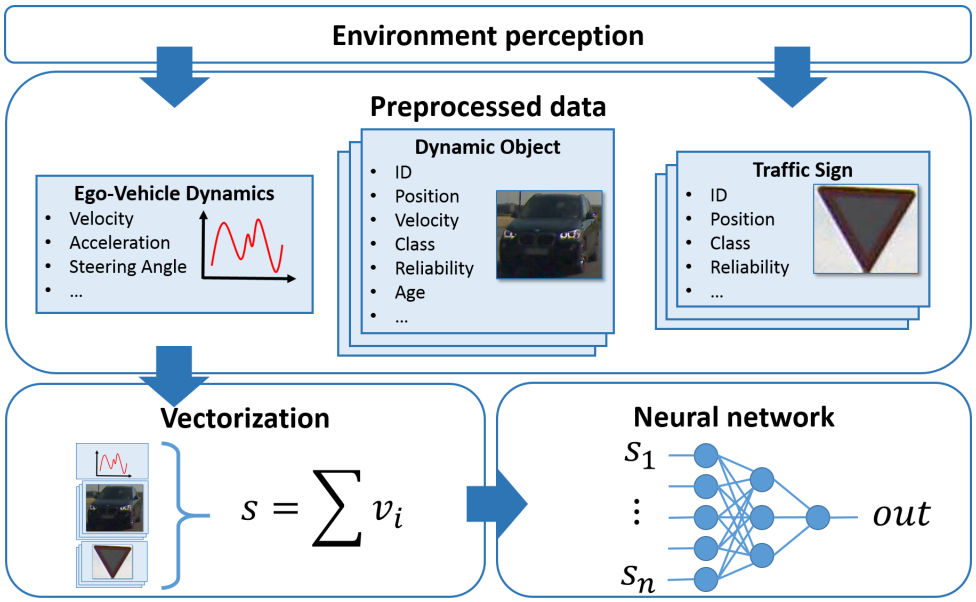


Figure 4.1: Visualization of the general flow of information of our proposed approach.

vectors within the vocabulary. Furthermore, we analyze how different types of data could be encoded in such a representation, we show possible variations of how to encapsulate data in vectors and how they influence the final representation. We also investigate potential limitations imposed by such representations to provide insights into how many concepts can be efficiently encoded in our representations without loss of information.

4.1 Preprocessing stage - generating a vocabulary

Figure 4.1 visualizes the general flow of information of our proposed system. To represent high-level information about a scene in an abstract vector representation, we work with already processed data, which comes either from individual sensors performing their own low-level processing, or from a higher-level, central module already fusing information from several sensors. We simply refer to this step as *environment perception* in Fig. 4.1, whereas its output is referred to as *preprocessed data*. This data is typically available as lists of objects present in the current scene and is translated into a semantic vector representation by first assigning atomic vectors to entities of interest and then building up more complex, structured representations by using the SPA's algebraic operations. In this section, we will investigate the first step of assigning atomic vectors to entities of interest, i.e., creating a suitable vector vocabulary. We have already seen in section 3.4.1 that such vocabularies can be created in several different ways, which we will here investigate with the specific focus of encoding automotive scenes.

4.1.1 What types of data to encode?

The data to be encoded in a semantic vector substrate depends not only on the information available from the current sensor-setup, but also on the task at hand. For instance, if we want to classify the current driving context (like in chapter 5), the relevant information might be different to the task of predicting a vehicle's trajectory (like in chapter 6). Here, we give an overview of what information in general is available in an intelligent vehicle and how to encode it in a semantic vector substrate. We distinguish between symbol-like information such as the type of a dynamic object or numerical information such as the current acceleration of the ego-vehicle. In this section, we focus mainly on the symbol-like information, which is suitable to be represented using a single atomic vector or an algebraic combination of several atomic vectors. In section 4.2.1, we will focus on numerical information and different options of

encoding them in vectors. Here, we will closely follow the structure of section 3.4 and present different possibilities to generate a vocabulary of atomic vectors to built structured representations upon.
If the vectors in the vocabulary are not chosen at random, the general goal when creating the vocabulary is to generate atomic vectors that carry inherent structure or meaning. This meaning is typically reflected by similar concepts being mapped to similar vectors. However, there are several possible notions of similarity that can be encoded in the vocabulary, which we will specify in this section.

Visual similarity

A first simple and comprehensible notion of similarity is visual similarity between two entities: "do they look similar". Encoding this notion in vectors, we would expect the vector representations to encapsulate this type of similarity within the relation between the vectors, meaning that vectors representing visually similar entities will have large cosine similarity. In conjunction with other information, this type of similarity could be useful to detect a wrong classification, in case it has high similarity to another one that makes more sense in the situation or context at hand. An example would be the German traffic signs indicating a speed limit of 30 km/h and 80 km/h, which are both circular with a red frame and a similar looking black number on white background in the middle. However, encountering a speed limit sign of 30 km/h in an urban situation is more plausible than a sign indicating a speed limit of 80 km/h.

Similarity of motion

Another notion of similarity, that is a candidate to be encoded in a vector vocabulary, is the similarity of motion properties. For instance, bicycles and motorcycles have more similar motion properties (dynamics of vehicles with two wheels) than for example a pedestrian and a truck. Apart from motion properties such as dynamics of the movement, the number of wheels or the mean expected velocity, the direction of movement of traffic participants can be also a notion of similarity to be encoded in the vocabulary. For instance, traffic participants such as bicycles or cars moving towards us might be encoded more similarly to each other than to parked cars or those moving away. Furthermore, some entities are more likely to change their motion or direction: while traffic lights frequently alternate and parked cars might start moving, trees, buildings and traffic signs are expected to remain static. The notion of similarity of motion can be useful in various ways. In a first step, knowing the motion of other traffic participants could help in classifying the situation: on a multi-lane highway we would expect cars around us to move in the same direction, whereas in an urban driving situation, the motion of other traffic participants is more diverse. Additionally, this notion of similarity could potentially help in focusing the system's attention or, more precisely, use computing power more efficiently on entities that are more relevant to decision making. For example, a change of the "motion status" (e.g., when a bus stops) might need particular attention.

Semantic similarity

While visual similarity already captures a significant part of perceivable information about entities in automotive context, there is further information, that could be encoded in the vector vocabulary and that is different, or maybe even contrary to visual similarity. Considering automotive situations, we as humans do not only assess them based on visual appearance but also by incorporating underlying and, most likely, previously acquired knowledge about the objects in the scene. This underlying information can be considered the semantic aspect. Revisiting the aforementioned example, the speed limit sign for 30 km/h is *visually* more similar to 80 km/h than to 20 km/h. However, in the context (or semantics) of an automotive situation such as driving in an urban environment, speed limit signs for 20 and 30 km/h should be contextually or semantically more similar to each other than signs for 30 and 80 km/h, since they are more likely to appear in similar contexts and both describe the traffic rule restricting driving to slow velocities.

Semantic similarity is not quite as intuitive as visual similarity. In general, we want to encode objects and concepts sharing similarity in *meaning* in vectors with a high cosine similarity. However, it is not intuitively clear what similar meaning actually refers to and how to properly define semantic similarity in an automotive context. In the field of generating word embeddings for natural language processing, the typical assumption is that words sharing similarity in meaning appear in close proximity with high probability within text corpora. This assumption could be transferred to automotive context as well, for instance, thinking of traffic signs indicating speed limits appearing in similar contexts or driving situations. For example, traffic signs indicating moderate speed limits are more likely to appear in urban driving situations compared to higher speed limit signs being less likely to appear in such a context. However, on the one hand, it is not clear how to transfer this approach to other object classes such as traffic participants, whose appearance probability is less context dependent than for traffic signs. On the other hand, the process of automatically training a system to learn this form of embedding is not clear as it would probably demand for another learning model to extract contextual information from the rich features of driving contexts. Therefore, we will now focus on the potential meaning of objects appearing in an automotive environment and how their semantic meaning could be embedded into a vector representation while leaving aside intangible concepts representing vehicle dynamics such as *velocity* or *acceleration*.

Traffic signs Any traffic sign carries an explicit meaning defined in traffic law and anyone with a driver's license should know its meaning and be able to immediately explain it. The meaning of a traffic sign is an instruction for the driver's behavior to, for instance, not surpass a certain velocity or to give way to other traffic participants. There are sub-groups of signs with similar meanings such as signs indicating speed limits, prescribed direction or warnings for potentially dangerous road conditions or to pay increased attention. Encoding the semantic structure of traffic signs in a vector vocabulary, we expect not only all traffic signs will be similar but also that all signs within a certain sub-group end up being more similar to one another compared to signs from other subgroups. For instance, signs indicating speed limits should be similar to one another, ideally with signs indicating lower velocities such as $20\,\mathrm{km/h}$ and $30\,\mathrm{km/h}$ should be more similar than $30\,\mathrm{km/h}$ and $130\,\mathrm{km/h}$. Beside their explicit meaning, the number of traffic signs is finite and limited to a small number compared to the number of words in a typical human-level language vocabulary. Therefore, it is possible to manually engineer their semantic similarity, which makes it easier to impose our human understanding onto the structure, although the resulting vocabulary will most likely differ from the structure an unsupervised learning model would pick up from the data.

Traffic participants While the meaning of traffic signs is clear and explicit, it is far more difficult to derive a meaning of a traffic participant such as a car or a pedestrian or a measure of similarity between them. It is unclear if a truck is semantically more similar to a motorcycle or to a car without considering any contextual information while ignoring similarity of motion properties, which have already been discussed in section 4.1.1. However, if we do consider contextual information, we as humans decide intuitively if a truck and a motorcycle are more similar when compared to a pedestrian by, e.g., considering their velocity of motion or their vulnerability. We also know from experience that the meaning of a car approaching from the right potentially means that it has the right of way when we encounter a situation at a crossroads without traffic signs indicating other right of way rules. Hence, the situational context has a significant impact on comprehending the *meaning* of traffic participants, which in most cases directly results in appropriate driving actions to take such as decelerating or changing the lane. However, such a situational understanding is impossible to derive without additional information such as position, velocity or direction of each traffic participant. Consequently, it is impossible to encapsulate such semantic or contextual similarity in a vector vocabulary directly but rather encode another notion of similarity for atomic vectors of traffic participants and built situational similarity through structured representations using the VSA's algebraic operations.

Summary on similarity

All of the aforementioned notions cover a certain aspect of similarity. Ideally, it is desirable for a vector vocabulary to encapsulate more than one notion of similarity comparable to a human understanding all these different aspects of similarity. However, it is not clear if it is helpful or even possible, to encapsulate several notions of similarity into one coherent vector vocabulary or if it is more suitable to have separate vocabularies for each similarity of interest and combine them in structured representations using the VSA's algebraic operations as mentioned, e.g., in Crawford et al. (2016). For the remainder of this section, we give an overview over different options of how to generate vector vocabularies encoding their own notion of similarity.

4.1.2 Random and manually engineered vocabularies

The simplest possible option to generate a vocabulary of atomic vectors is to sample them randomly, in case of continuous VSAs such as the SPA, from the D-dimensional unit sphere (Voelker et al., 2017). Naturally, randomly chosen atomic vectors do not carry semantic meaning or any intended notion of similarity. However, given a sufficiently large dimension D of the chosen VSA and its theoretical properties (see chapter 3), we can assume that randomly chosen vectors will be dissimilar enough to avoid accidentally mistaking them for one another. Another advantage of this simple approach is that it is comparatively easy to create a vocabulary avoiding any complex learning system to embed the concepts of interest in semantic vectors. On the other hand, if specific applications demand for the vectors to actually carry semantic information meaning that similar concepts need to be mapped to similar vectors for the application to succeed, this simple approach can be extended by manually engineering a vocabulary reflecting the desired similarity structure. This is typically achieved by randomly choosing a set of auxiliary vectors and building atomic vectors with the desired similarity from them through the VSA's algebraic operations (cf. section 3.4.1). However, this approach is only feasible and appropriate for rather small sized vocabularies since manually designing semantic vectors with certain similarity properties becomes intractable quickly with an increasing number of concepts to be embedded. Furthermore, manually designing vocabularies involves design choices by human engineers, which is sensitive to potentially undesired biases in the similarity structure of the vectors. For instance, the example vocabulary created in section 3.4.1 solely focused on the motion characteristics and typical actuators of different traffic participants. However as mentioned before, there are many other possible similarity structures such as visual (or auditory), motion or semantic similarity, to be considered when designing the vocabulary.

4.1.3 Visual vocabularies

The next step to generate vocabularies with an inherent similarity structure is to encapsulate visual similarity in a vector embedding. Visual similarity is an intuitive concept and we can make an educated guess that this notion of similarity could be beneficial for several tasks when encoded directly in the vector vocabulary. However, it is preferable to automatically learn to embed visual similarity in vectors instead of manually engineering the similarity between entities as this approach does not scale well for an increasing number of items to be encoded. One option for such an automated learning system to generate a structured vocabulary encoding visual similarity is to adapt a DNN (Deep Neural Network) for image classification. Such a neural network can be thought of as an efficient image compression machine. While earlier and intermediate layers learn sensibility to visual features such as edges and shapes, the information in the image is compressed into a single dimension, namely the label, at the final classification layer. Considering the special case of a CNN (Convolutional Neural Network), one option would be to simply use the output of one of the later fully-connected layers and regard it as a vector since we expect the vectors of visually similar images to be similar regarding their cosine similarity. Here, we focus our efforts of generating a visual vector vocabulary on items of interest to automated driving, namely the aforementioned categories of traffic signs and traffic participants.

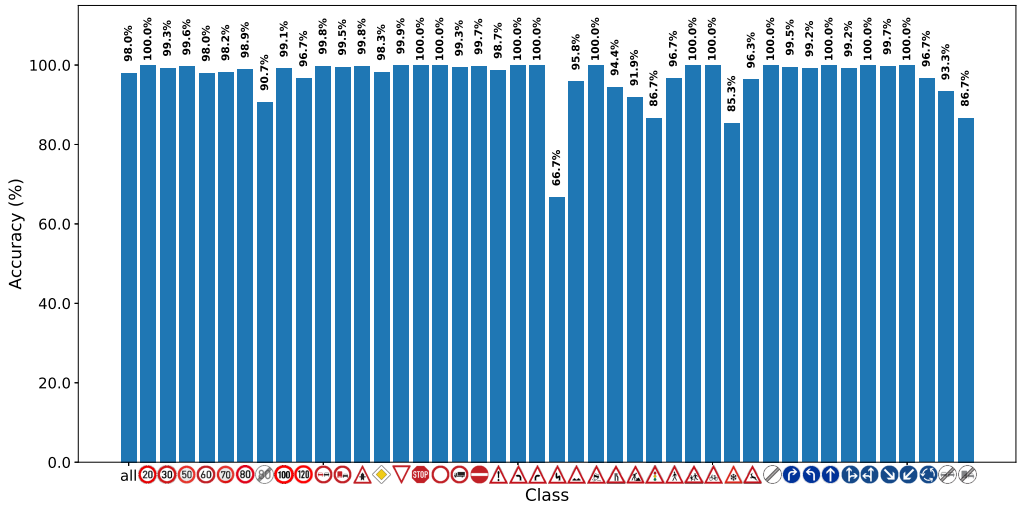


Figure 4.2: Accuracy performance of the CNN for traffic sign classification on the test part of the GTSRB data set for all traffic signs (most left bar) and all individual traffic signs.

Traffic signs

To achieve the task of encoding traffic signs encountered by an automated vehicle, we employ a variant of the state-of-the-art CNN for traffic sign classification proposed by Ciresan et al. (2012a). We train a simplified version of this network on the GTSRB (German Traffic Sign Recognition Benchmark) (Stallkamp et al., 2012), which is a data set including a total of 51 840 images of 43 different classes of traffic signs. Although the GTSRB data set does not contain all possible traffic signs, it is a suitable and sufficiently large data set for our purposes of learning a visual vector vocabulary. The original multi-column network proposed by Ciresan et al. (2012a) combines several variants of the same network architecture trained on the original input images as well as four different, high-contrast normalized versions of the input images by averaging the predictions of all individual networks. For simplicity, we only use a single-column version of this network to generate a visual vector vocabulary. Figure 4.2 shows the classification accuracy of our network on the test part of the GTSRB data set for all traffic signs (most left bar) and all individual signs. The network achieves competitive results with 98 % classification accuracy on all traffic signs while detecting all but four traffic signs with accuracy values way above 90 %, which is sufficient for our purposes.

To generate our vocabulary vectors, we cut off the classification layer with softmax activation and use the previous 300-dimensional, fully connected layer as output. With this simple adaptation, the CNN produces a 300-dimensional vector as output for each image fed into the network. To generate a representative vector for each class of traffic signs, several approaches are conceivable, including (weighted) mean and median by dimension or regarding whole vectors. In this work, we choose the simple mean to create this representative from several examples. To select the example vectors to calculate the representative mean vector from, we select only instances for which the network produces correct classification predictions alongside high confidence values from the test subset of the GTSRB data set. The great majority of examples even satisfies the restriction of 100 % confidence, hence we use that strict confidence value to avoid including examples the network is doubtful about, which might deteriorate the properties of the representative vector. This procedure leads to a representative vector that "points" to the center of mass of all (high confidence) vectors of each class of traffic signs. However, we still need to confirm that these representative vectors fulfill the properties they have been constructed for, namely sharing a high cosine similarity with all individual samples from the respective traffic sign class. Figure 4.3 shows

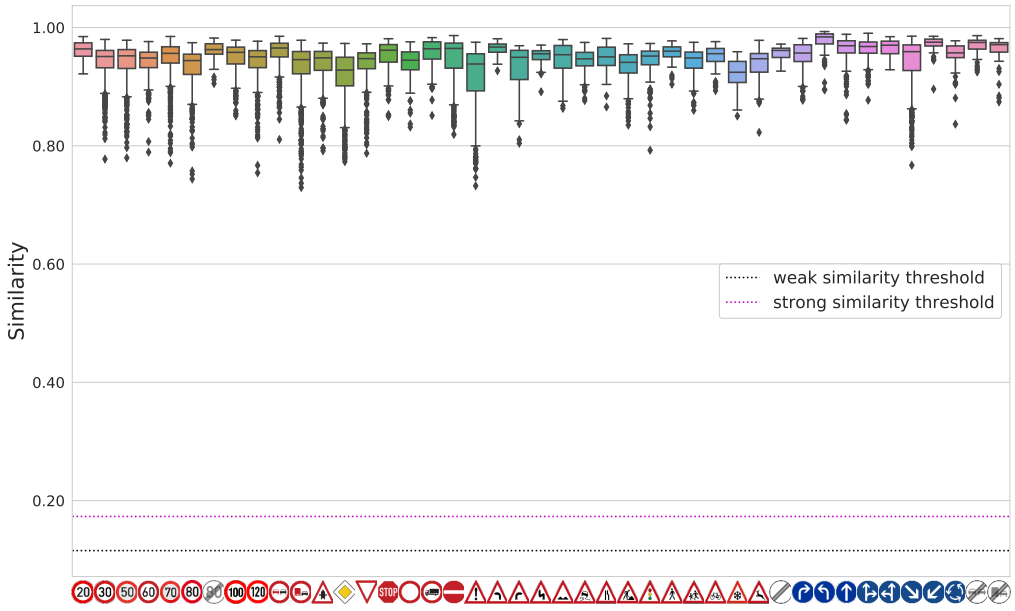


Figure 4.3: Boxplots depicting the cosine similarities between the representative (mean) vector for each traffic sign class and all individual vector samples it has been created from.

the cosine similarity between each vocabulary vector encoding one traffic sign class with all individual vector samples it has been created from. As expected, we observe high similarity values close to the maximum value of 1 and way above both, weak and strong, similarity thresholds for 300-dimensional vectors.

Furthermore, we expect these representative vectors, which are now our vocabulary vectors encoding the respective traffic signs, to resemble the visual similarity structure of the image classes. To confirm this inherent similarity structure, we calculate pairwise similarities between all vectors in the vocabulary, which are visualized in Fig. 4.4. We observe similarities in groups of signs indicated by green areas in the heat map visualization. Most prominent are the high similarities in three groups of traffic signs forming green triangles in the heat map. These groups are round signs with red borders (top left corner), triangular warning signs with red borders (middle right) and blue signs indicating driving directions (close to bottom right). Furthermore, several signs stand out particularly: The traffic sign indicating "Priority ahead", which is a red triangle just like any warning signs, indeed shows high similarity to all warning signs. Similarly, the traffic sign indicating no entry for trucks, a red circle with a black truck inside, looks a lot like speed limit signs and indeed shows a high similarity to all speed limit signs. Consequently, we conclude that it is possible to encode visual similarity with an automatic learning approach using CNNs to encapsulate the visual features of a given data set into a vocabulary of semantic vectors.

Traffic participants

To encode the object categories for traffic participants *Bicycle*, *Car*, *Motorcycle*, *Pedestrian*, *Truck* necessary to properly represent dynamic automotive scenes in visual vectors, we employ an approach similar to the one used for traffic signs. We adopt a general purpose image data set, the Imagenet data set (Deng et al., 2009), which includes suitable categories for all of these classes. As an additional advantage, Imagenet is a widely used data set, so there already exists a number of successful classification networks, which can be adapted for our purposes. Concretely, we employ the following categories as they appear

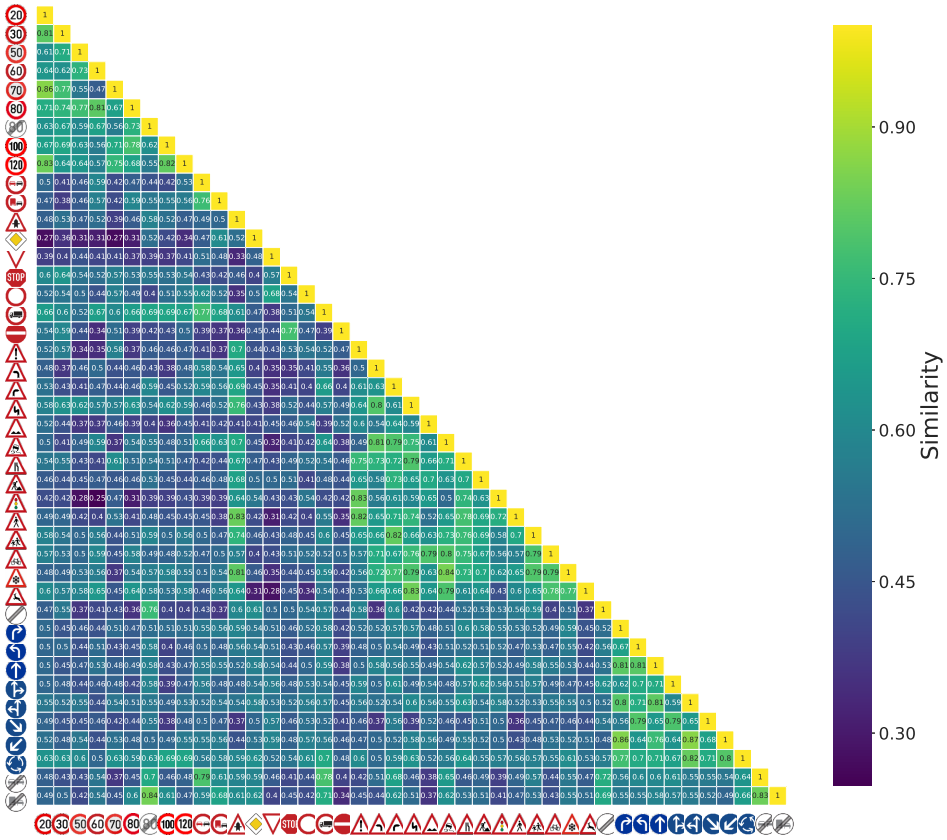
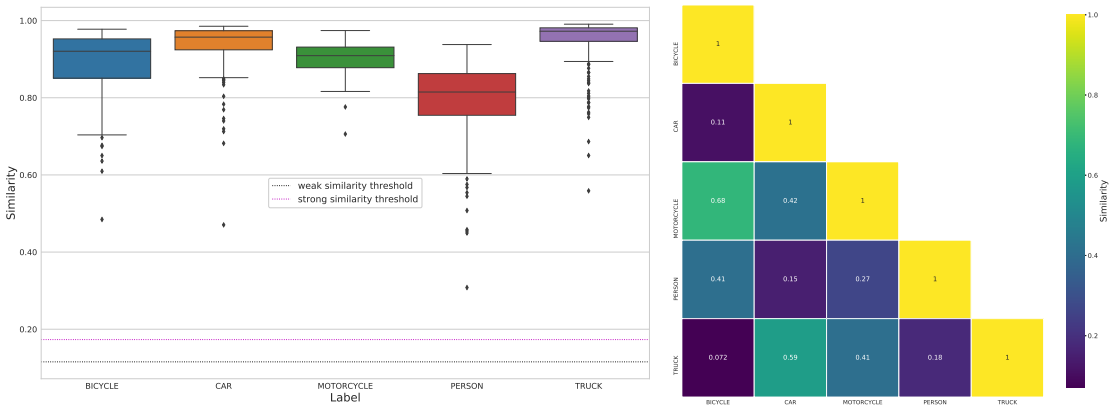


Figure 4.4: Pairwise similarities between representative vectors encoding traffic signs in a visual vector vocabulary.

visually most suitable for the objects categories we want to encode: We use the original labels 'SAFETY BICYCLE' to learn visual vectors for *Bicycle*, 'USED CAR' for *Car* (other car categories include ambulances and many sports / racing cars), 'MOTORCYCLE', 'PERSON' for *Pedestrian* (since there is no special category for people in the vicinity of roads) and 'TRUCK'. For each category, there are at least 1200 images available in the data set.

While there is a number of state-of-the-art classification networks available achieving good results on the Imagenet data set, we are looking for a network with only moderately complex structure for the sake of implementation simplicity and ease of adaptation, while performance is of secondary priority. VGG19 is a deep CNN (Convolutional Neural Network) proposed by Simonyan and Zisserman (2014) with a decent performance on the Imagenet data set (top 1 performance 73 % and top 5 performance 91 %) and a comparatively simple layer by layer architecture that allows easy extraction of results from intermediate layers. Our goal is to train a variant of the VGG19 network on these 5 classes and extract feature vectors of an intermediate layer to use them as vocabulary vectors. Therefore, we adapt the original network architecture in the following way: We cut off the last two fully connected layers as well as the classification layer and replace them with a fully connected, 300-dimensional layer and a 5 dimensional classification layer. We train the modified network using the categorical cross entropy loss and "standard" stochastic gradient descent optimization.

Table 4.1 depicts the classification performance of our adapted VGG19 network for the 5 selected traffic



(a) **(b)**

Figure 4.5: Similarity plots for the visual vocabulary vectors representing traffic participants. (a) Boxplots depicting the cosine similarities between the representative (mean) vector for each traffic participant class and all individual vector samples it has been created from. (b) Pairwise similarities between representative vectors encoding traffic participants in our visual vector vocabulary.

participant categories. Except for the "MOTORCYCLE" class, the network achieves decent classification results above 75 %, which is sufficient for our purposes. Similar to the creation of the visual vocabulary for traffic signs, we calculate the mean of vectors produced by the second to last of the network's layers when making correct predictions with 100 % confidence. Even for the "MOTORCYCLE" class, there are at least 139 of such vectors available, whereas for all other classes, we have at least 200 of such vectors. To confirm that the vocabulary vectors created in this fashion are visually representative enough for each class, we calculate the cosine similarity between the vocabulary vectors and all vector samples they have been created from. Figure 4.5a visualizes these similarities as box plots. Similar to the traffic sign vocabulary, we observe high similarity values close to the maximum value of 1 and way above both, weak and strong, similarity thresholds for 300-dimensional vectors. We also calculated pairwise similarities between all vocabulary vectors encoding traffic participants, which are visualized in Fig. 4.5b. As expected, visually similar classes such as "BICYCLE" and "MOTORCYCLE" as well as "CAR" and "TRUCK" share relatively high cosine similarities. Less similar, but still significantly higher than the similarity thresholds, are traffic participants sharing the visual features of persons such as "PERSON" and "BICYCLE", "PERSON" and "MOTORCYCLE" as well as "BICYCLE" and "MOTORCYCLE". All other pairs have similarity values in the order of magnitude or below the similarity thresholds and are therefore considered dissimilar while we do not attach the greatest importance to the actual (low) numbers. Consequently, we can conclude, that we are able to encode visual similarity of traffic signs as well as traffic participants in a visual vector vocabulary.

	BICYCLE	CAR	MOTORCYCLE	PERSON	TRUCK
Classification accuracy	94.3 %	76 %	55 %	82.6 %	99 %

Table 4.1: Classification accuracy of the adapted VGG19 network for our selected 5 classes of traffic participants.

4.1.4 Semantic vocabularies

In this section, we go one step further and try to encapsulate semantic similarity structures within a vector vocabulary. As discussed in section 4.1.1, semantic similarity as a concept is comparatively intuitive for traffic signs and less obvious for other objects, such as traffic participants. Furthermore, we also highlighted in that section that semantic similarity in other domains such as language modeling typically

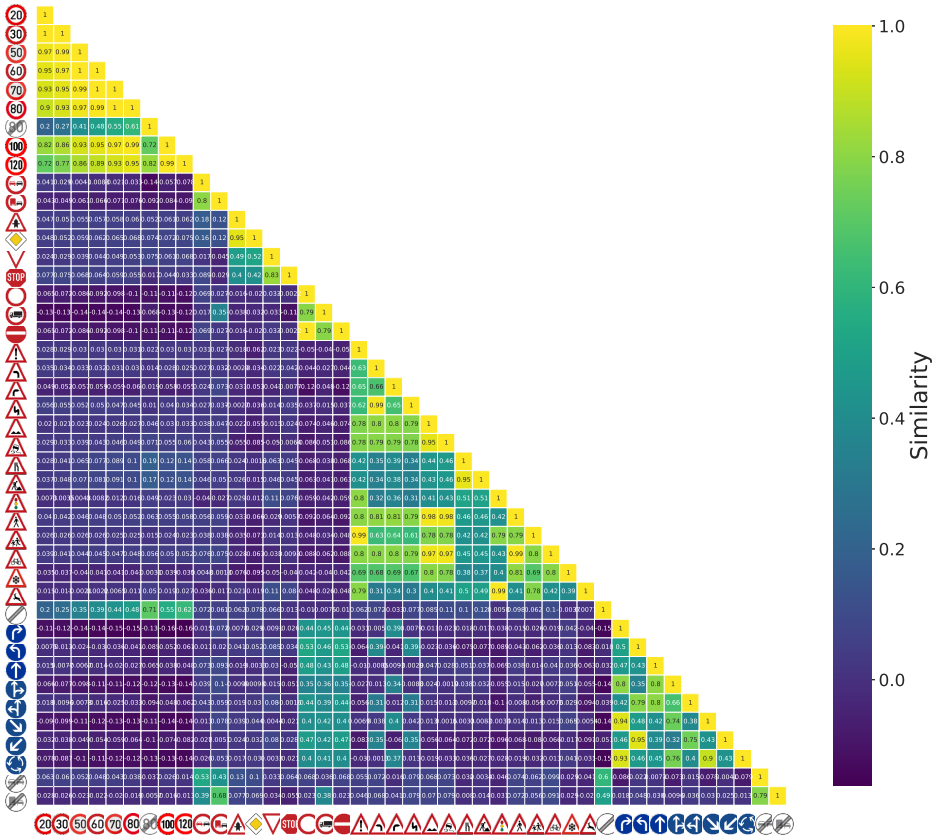


Figure 4.6: Pairwise similarities between representative vectors encoding traffic signs in a manually designed semantic vector vocabulary.

unfolds through proximity, i.e., that similar words appear in similar contexts or proximity within the text. While this could give a hint towards what kind of learning procedure could be used to automatically generate a semantic vocabulary in automotive context, availability of suitable data sets is rather limited. Analogously to the visual vocabulary, we consider the encoding of traffic signs and traffic participants separately in this section as well.

Traffic signs

Our goal is to encode the meaning of traffic signs, i.e., the driving instruction or traffic rule they indicate to the driver in a semantic vector vocabulary. As with visual similarity, this goal could be achieved by either manually engineering the similarity structure through the VSA's algebraic operations from randomly chosen atomic vectors or through some automated learning approach. If we were to learn this similarity structure automatically, there are two possible approaches. Similar to word embedding algorithms for language, we could either learn the meaning of traffic signs explicitly from a large corpus of text that describes traffic signs and their meanings in context. Alternatively, we could try to learn the semantic meaning of traffic signs implicitly from many dynamic driving situations. Unfortunately, there are no suitable data sets available for either of the aforementioned learning approaches. Additionally,

an implementation of the latter, implicit learning approach would be quite complex, as it would require additional steps to extract structural understanding from the driving scene, which would be necessary for the vocabulary generation system to create suitable vectors. To our knowledge, such a system does not exist. Consequently, the only remaining option is to manually design the desired similarity structure as described in section 4.1.2.
To encode the semantic meaning of traffic signs, we choose atomic vectors for the basic building blocks of the representation at random and create semantic structure by employing the algebraic operations of the SPA. We apply the role-filler pair approach described in section 3.4.2. We randomly choose vectors for the roles **TYPE** and **MEANING** encoding the type and the meaning of a particular traffic sign. For some traffic signs, we need an additional role **REASON** giving further information to the vocabulary vector in order to distinguish similar traffic signs from one another. As potential filler vectors for the **TYPE** role, we choose random vectors representing the following traffic sign classes included in the GTSRB: **LIMIT**, **PASSING**, **PRIORITY**, **DIRECTION** and **ATTENTION**. Similarly, we create filler vectors for the **MEANING** role like **RIGHTOFWAY**, **GIVEWAY**, **SLOW**, **PREPARETOSTOP**, **CONCENTRATE**, **LEFT**, **RIGHT**, **STRAIGHT**, **OVERTAKING**. Finally, we create filler vectors for the **REASON** role indicating the reason for increased attention or other additional information such as **PEDESTRIANS**, **CHILDREN** or **SLIPPERYROAD**, **ROADWORKS**. Given these role and filler vectors, we generate semantic vocabulary vectors encoding the meaning of traffic signs in the following way

$$\mathbf{SIGN} = \mathbf{TYPE} \circledast \mathbf{FT} + \mathbf{MEANING} \circledast \left(\sum_{i=0}^{n} \beta_i \cdot \mathbf{FM}_i \right) + \gamma \cdot \mathbf{REASON} \circledast \mathbf{FR}, \tag{4.1}$$

where $\mathbf{FT}$, $\mathbf{FM}_i$ for $i = 0, \ldots, n$ and $\mathbf{FR}$ are placeholders for the filler vectors and $\beta_i \in \mathbb{R}$ for $i = 0, \ldots, n$ and $\gamma \in \mathbb{R}$ are weighting factors. In case, the additional **REASON** role is not needed for a particular traffic sign, the corresponding weight factor γ is set to 0. For instance, the vocabulary vector encoding the traffic sign indicating danger is calculated as

$$\mathbf{DANGER} = \mathbf{TYPE} \circledast \mathbf{ATTENTION} + \mathbf{MEANING} \circledast (\mathbf{SLOW} + \mathbf{PREPARETOSTOP}) . \tag{4.2}$$

Traffic signs indicating speed limits form somewhat of a special case, as we want to encode them in such a way, that signs indicating lower speed limits are more similar to one another than to signs indicating higher speed limits. To achieve that, we need to encode numerical values that are closer to one another more similarly than numerical values with larger intervals. Here, we employ a very simple encoding scheme using the function

$$\varphi : \mathbb{R} \longrightarrow \mathbb{R}^D, x \longmapsto (\sin(x), \cos(x), 0, \ldots, 0) . \tag{4.3}$$

In other words, we create a vector representing the numerical value of the speed limit by setting the first two dimensions to $\sin(x)$ and $\cos(x)$ for the encoded numerical value $x \in [0, \frac{\pi}{2}]$ and all other entries to 0 (note that we will discuss other, more complex approaches to encode numerical values in vectors in section 4.2.1). We use $200\,\mathrm{km/h}$ as general speed limit, i.e., we map all speed limit values between 0 and 200 to the interval $\left[0, \frac{\pi}{2}\right]$ with $200\,\mathrm{km/h} \equiv \frac{\pi}{2}$.
Figure 4.6 shows the pairwise similarities between the manually designed semantic vectors encoding traffic signs in the GTSRB created in the aforementioned fashion. As expected, we see a highly structured area in the top left corner, the speed limit signs. The other groups of signs (overtaking, priority, warning and direction) are also visible as triangles on the right hand side. All non-related entities have low similarities of less than 0.1. Brighter spots in the large dark area to the left represent similarities across sign groups, especially, for signs related to trucks and curves or bends. Consequently, we were successful in manually designing a vocabulary encapsulating semantic similarity of traffic signs as an alternative to the visual vocabulary created in section 4.1.3.

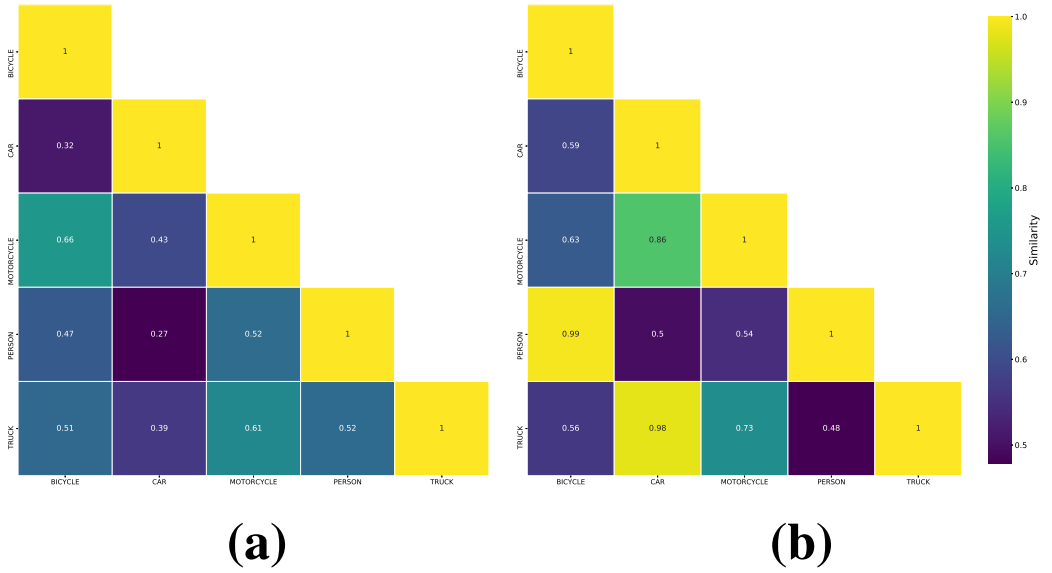


Figure 4.7: Pairwise similarities between representative vectors encoding traffic participants in a semantic vector vocabulary. (a) Learned with word2vec (b) manually designed.

Traffic participants

As discussed in section 4.1.1, semantic similarity for traffic participants is hard to capture intuitively. We concluded that the general meaning of traffic participants is highly context-dependent, which in turn can only be learned from dynamic driving data or from a text corpus describing traffic situations. However, such data sets are either not available or do not even exist. Hence, as a first step towards the goal of encoding semantic similarity, we will therefore encapsulate the similarity between the five classes of traffic participants already discussed in section 4.1.3, namely *Bicycle*, *Car*, *Motorcycle*, *Pedestrian*, *Truck*. Here, we employ two different approaches: an automated learning approach making use of the well-established word embedding of the word2vec algorithm (Mikolov et al., 2013b) trained on the Google News data set and, similar to the semantic vocabulary of traffic signs, manual design, which is only possible due to the small size of our vocabulary. Word2vec is an unsupervised learning approach generating a word embedding, i.e., vectors representing every word encountered in the training text, where words that appear in similar context, i.e., close proximity within the text, are mapped to similar vectors. For our purposes, we simply extract the (300-dimensional) vectors representing the objects of interest (traffic participants) from the learned vocabulary. The manually designed semantic vocabulary is generated similarly to the aforementioned vocabulary of traffic signs using the SPA's algebraic operation and the role-filler-pairs approach based on two key properties, which give good yet simple descriptions of the traffic participant's semantic properties: speed and vulnerability represented by randomly chosen atomic vectors **SPEED** and **VULNERABILITY**. Hence, we encode the five classes of traffic participants through

$$\mathbf{PARTICIPANT} = \mathbf{SPEED} \circledast \varphi(s) + \mathbf{VULNERABILITY} \circledast \varphi(v), \tag{4.4}$$

using the encoding function φ from Equation (4.3).
Figure 4.7 depicts the pairwise similarities of both, the semantic vocabulary using word2vec vectors (Fig. 4.7a) and the manually designed vocabulary vectors (Fig. 4.7b). We observe that the pre-trained word vectors from word2vec are not entirely successful to capture the kind of semantic similarity we are interested in. While vectors are similar to each other, the individual similarities do not always match our human understanding, especially when considering an automotive context. For instance, the vector

encoding *person* is significantly more similar to the one representing *truck* than to the one representing *car* Furthermore, *car* is the traffic participant least similar to *truck*.
For the manually designed vocabulary (Fig. 4.7b), results are more convincing. We are able to achieve a higher similarity between vectors encoding *car* and *truck* as well as *bicycle* and *person* with lower similarities between *person* and *truck* as well as *bicycle* and *truck*. This vocabulary is also not an ideal representation but gets much closer to the desired semantic structure between these entities in an automotive context.

4.1.5 Visual-semantic vocabularies

After having created vector vocabularies encoding visual (section 4.1.3) and semantic similarity (section 4.1.4) structures, we aim to generate a vector vocabulary, which combines both aforementioned similarities in one comprehensive vocabulary. The idea is that for high or low similarity in both, visual and semantic domain we want the resulting vectors to preserve or even increase that similarity structure. For a diverging similarity in the two domains, the fusion process should somewhat dilute the two extremes resulting in vectors with medium similarity. Contrary to other visual-semantic fusion methods, these properties can be achieved employing again the algebraic operations of the SPA and the role-filler-pair approach. Therefore, we randomly choose two additional role vectors, **VISUAL** and **SEMANTIC** and construct the visual-semantic vocabulary vectors as

$$\mathbf{VISUALSEMANTIC} = \mathbf{VISUAL} \circledast \mathbf{V} + \mathbf{SEMANTIC} \circledast \mathbf{S}, \tag{4.5}$$

where $\mathbf{V}$ and $\mathbf{S}$ are placeholders for the visual and semantic vocabulary vectors to be fused. Given the mathematical properties of the SPA's algebraic operations (cf. chapter 3), this fusion procedure guarantees that for two similar vectors in either of the input domains, the resulting visual-semantic vectors will preserve that similarity.

Traffic signs

Figure 4.8 shows the pairwise similarities of the visual-semantic vocabulary vectors representing traffic signs. We observe that opposite similarities from the input domains are smoothed out through the convolution-based fusion process. For instance, the strong visual similarity between the traffic sign indicating "Priority ahead" and all attention signs (triangular shape, red border, black symbol in the middle), which has been clearly visible in Fig. 4.4, does not correspond to a large semantic similarity and was canceled during the fusion procedure. The same holds true for the clearing signs as well as for the similarities between signs indicating "No entry trucks" and "roundabout" and the speed limits. On the other hand, for those traffic signs with high semantic similarity such as between signs indicating overtaking rules for trucks and the sign indicating no entry for trucks, this similarity is preserved in the joint visual-semantic vocabulary. Particularly for the speed limits, we observe that a high similarity in both domains translates into a high joint similarity, whereas in other cases, the manually designed semantic similarity is partly "overwritten" with the visual information. In general, many signs from different groups with very low semantic similarity (the dark blue block in the bottom left of Fig. 4.8) are increased by the visual part.

Traffic participants

In section 4.1.4, we created two semantic vocabularies, which have been learned with word2vec and manually designed. Figure 4.9 depicts the pairwise similarities of two visual-semantic vocabularies, where the two different vocabularies from section 4.1.4 are used for the semantic part. We observe fusion results similar to those for the visual-semantic traffic sign vocabulary, where opposite similarities from the input domains are smoothed out through the convolution-based fusion process while differences in semantic structure between the two methods remain visible in the complete vocabulary. Consequently,

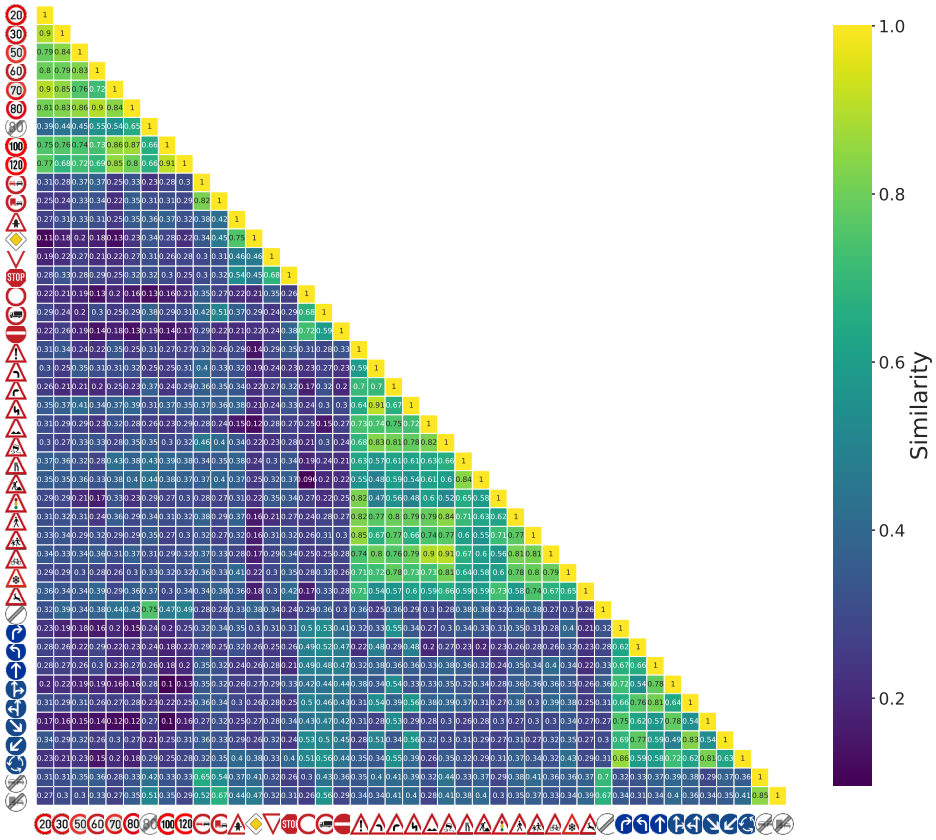


Figure 4.8: Pairwise similarities between vectors encoding traffic signs in a visual-semantic vocabulary.

the visual-semantic encoding based on manually designed semantics appear to be a convincing vector representation for the similarity structure expected for traffic participants in an automotive context.

4.1.6 Summary on vocabularies

In this section, we investigated several options of generating a suitable vector vocabulary for entities of interest in an automotive context. We have shown a way of learning a visual structure for two classes of categories appearing in an automotive context, namely traffic signs and traffic participants using CNNs. Furthermore, we were able to create a learned semantic vocabulary for traffic participants, but not traffic signs due to lack of suitable training data sets. Hence, we designed the semantic vocabulary for traffic signs manually using the mathematical properties and algebraic operations of the SPA. Furthermore, we also generate an alternative semantic vocabulary for traffic participants as a baseline to compare the learned semantic structure, which has not been adapted to the automotive context, to. Finally, we successfully generated a visual-semantic vocabulary encapsulating both, visual and semantic similarity by using the SPA's algebraic operations and the role-filler-pair approach. This visual-semantic vocabulary appears to represent the expected similarity structure between the entities of interest in automotive context.

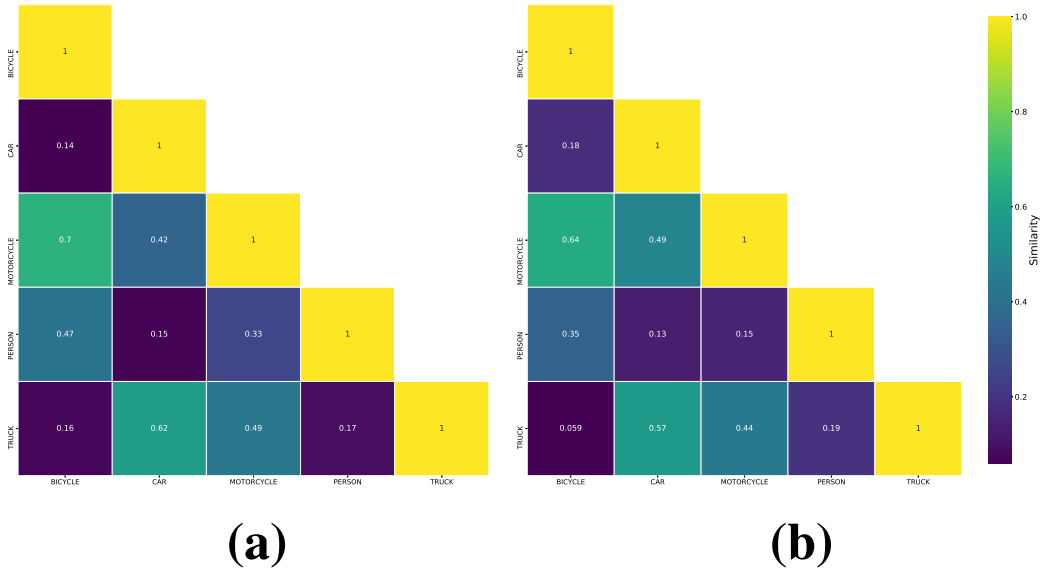


Figure 4.9: Pairwise similarities between vocabulary vectors encoding traffic participants in a visual-semantic vocabulary, where the semantic part is (a) learned with word2vec (b) manually designed.

It is worth noting however that the manual generation process of the semantic vocabularies, as mentioned in 4.1.2, imposes a significant amount of design choices biased by the human engineer. However, the choice of generating some of the semantic vectors in this context manually is due to the fact that on the one hand, the vocabulary is comparatively small while on the other hand, there are no suitable data sets available to learn the desired semantic structure from. Furthermore, the described procedure to fuse visual and semantic similarities into one coherent vocabulary is only one of several available options. It would have also been possible to try to automatically learn such a fusion process by projecting the visual vocabulary to the semantic one by combining visual properties and language descriptions like for instance in Karpathy and Fei-Fei (2017). However, such a learning approach again would require a suitable, sufficiently large data set, which in turn is not available for the entities investigated here.
Finally, the process of generating structured representations from an available vocabulary created with any of the techniques described in this section is independent from the particular vocabulary at hand. Hence, although it seems intuitive and useful to encode similarities of potential interest to the task to be solved using the vector representation, it remains to be seen, if using a vocabulary encapsulating any form of similarity actually improves task performance. Furthermore, the choice which similarity structure to use naturally appears to be heavily task-dependent. Thus, we will investigate the influence of the vocabulary structure on the task performance for the selected task of driving context classification in chapter 5 by simply evaluating the same models with changing underlying vocabularies.

4.2 Representation generation stage

After having created a vector vocabulary suitable for a specific application of interest as shown in section 4.1, we focus our attention in this section to the second stage of the encoding process proposed by Gallant and Okaywe (2013), the *representation generation stage*. This second step is the process of actually generating structured vector representations from a given vocabulary of atomic vectors. As mentioned by Gallant and Okaywe (2013): “Although the preprocessing and output computation stage can involve significant machine learning, there are reasons for the representation generation stage to *avoid*

machine learning". Using machine learning in the stage of representation generation means to adjust the representation to the specific set of applications, which restricts them to be used for different, but related sets of problems. Furthermore, involving learning in the representation generation might slow down the whole system prohibiting practical usage of the representation in application scenarios. However, in certain situations it could make sense to involve a learning step at the representation generation stage, for example if the system should detect novelties and assign them to new vectors or representations and/or needs to memorize them.
In this work however, we will focus our efforts mainly on representations which are manually designed without involving machine learning models. Thereby, we aim to generate representations that are general enough to be applied to a variety of different problems with only moderate modifications necessary when being applied to one particular task. One crucial aspect in the field of automotive scene representation is encoding dynamic data such as positions or velocities, which constitutes a significant part of the situational context. This context is essential for the representation to capture the semantics of the scene and gather a comprehensive understanding of the situation at hand. Depending on the application, different requirements on such a representation might be necessary. For instance, for some applications it might be necessary that the encoded numerical values need to be decoded back out exactly from the representing vector while for others an approximate recovery might be sufficient. Other tasks might demand for the representational vectors to have unit length independent from the encoded value. We therefore investigate several possible options of how to encode numerical values in semantic vectors focusing on different aspects of possible requirements.
Encoding numerical values, although an important one, is only one aspect of representing automotive scenes in high-dimensional vectors. The second crucial aspect of the representation generation stage is that of how to build up vectors representing structured information and complex interrelations encountered in automotive scenes. We need to combine several pieces of information into one or a set of scene vectors of fixed length representing the semantics of the situation. We will investigate how the SPA's algebraic operations can be used to combine numerical and symbol-like information to build such complex representations.
However, given the mathematical properties of VSAs in general and the SPA in particular, there are natural limitations to the amount of information that can be encoded in such a vector representation. Every algebraic operation, addition and circular convolution, will introduce a certain amount of noise into the representation, which imposes the functional demand for a clean-up memory (cf. chapter 3) for some applications. These limitations are actually not a flaw of such architectures, but rather a feature for being able to model limitations of cognitive functions of living beings, who are also not able to store unlimited amounts of information. The limitations of the modeling architecture are strongly connected to the chosen dimension of the underlying vector space. Therefore, we will finally analyze what kind of limitations these architectures impose on us and how much information can effectively be encoded in such a vector representation before noise reaches a critical level.

4.2.1 Different vector representations for numerical values

In this section, we investigate different approaches to map numerical information to semantic vectors. We have already seen one simple option for such an encoding in section 4.1.4, where we used the function φ from Equation (4.3). Here, we will show a more complex adoption of this encoding again making use of the sine and cosine functions alongside two more possibilities of how to represent numerical values.

Sine-Cosine-based representations

The idea behind using the sine and cosine functions to encode numerical values in high-dimensional vectors is their property of adding up to 1 when being squared, the Pythagorean identity

$$\sin^2(x) + \cos^2(x) = 1. \tag{4.6}$$

For the simple encoding given in Equation (4.3)

$$\varphi : \mathbb{R} \longrightarrow \mathbb{R}^D, x \longmapsto (\sin(x), \cos(x), 0, \ldots, 0), \qquad \text{((4.3) revisited)}$$

Equation (4.6) leads to the resulting vector $\varphi(x)$ having unit length, which is a desirable feature for many representations.
Since most of the dynamic data to be encoded in automotive context such as positions and velocities is two-dimensional, we also show a more complex adaptation of this simple trigonometrical encoding for two numerical values using different spatial frequencies and offsets. Therefore, we define the following auxiliary functions

$$f_{(m,i)} : \mathbb{R}^2 \longrightarrow \mathbb{R}^4, (x,y) \longmapsto \left(\cos \frac{m \cdot \pi + x}{i+1}, \sin \frac{m \cdot \pi + x}{i+1}, \cos \frac{m \cdot \pi + y}{i+1}, \sin \frac{m \cdot \pi + y}{i+1} \right), \tag{4.7}$$

$$\psi_i : \mathbb{R}^2 \longrightarrow \mathbb{R}^4, (x,y) \longmapsto \left(f_{(0,i)}(x,y), f_{\left(\frac{1}{2},i\right)}(x,y), f_{(1,i)}(x,y), f_{\left(\frac{3}{2},i\right)}(x,y) \right) \tag{4.8}$$

and obtain the final vector representation of two-dimensional values via the function

$$\lambda : \mathbb{R}^2 \longrightarrow \mathbb{R}^D, (x,y) \longmapsto \frac{1}{\sqrt{\frac{D}{2}}} \left(\psi_0(x,y), \cdots, \psi_{\frac{D}{16}-1}(x,y) \right). \tag{4.9}$$

The encoding $\lambda(x,y)$ leads to non-zero vectors having unit length with information distributed over all elements (in contrast to a simple encoding like $(x,y,0\ldots,0)$). Although for both trigonometrical encoding schemes shown here there is a way of decoding back out the input values exactly from resulting vector by using either the inverse trigonometrical functions or the atan2 function, both approaches have certain drawbacks. For instance, the simple encoding (cf. Equation (4.3)) uses only two of a typically large number of dimensions and thus somewhat neglects the architectural strength of VSAs being *distributed* representations. On the other hand, the encoding of two-dimensional values based on different spatial frequencies and offsets of the trigonometrical functions uses all of the available dimensions, but is constructed for vector space dimensions being a multiple of 16. Although this is true for all sufficiently large powers of 2 such as $512 = 2^9, 1024 = 2^{10}, \ldots$ it still imposes significant restrictions on the potential dimensions of the vector spaces to be used.

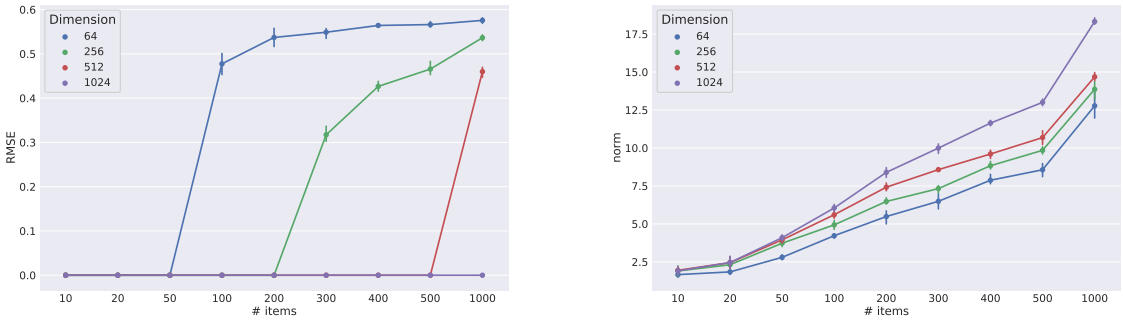
Scalar multiplication encoding

Apart from encoding numerical values in vectors as the only non-zero elements within a vector containing only zero elements elsewhere, possibly the simplest encoding is to (for instance, randomly) choose vectors representing the desired entity/unit to be encoded and simply multiply all elements of that vector with the number the vector should represent. Hence, to encode a sequence $a_1, \ldots, a_n$ of numerical values in vectors, we create a vocabulary of vectors $\mathscr{V} = \{\mathbf{X}_1, \ldots, \mathbf{X}_n\}$ representing the corresponding units and multiply them by the scalar values with the vectors representing the units $a_i \cdot \mathbf{X}_i$. Finally, summing up all of these vectors generates one single vector encoding all of the numerical values $a_1, \ldots, a_n$

$$\mathbf{V} = \sum_{i=1}^{n} a_i \cdot \mathbf{X}_i. \tag{4.10}$$

For $\mathbf{X}_i = (x_{i0}, \ldots, x_{iD-1})^\mathsf{T}$, this encoding is equivalent to the linear map given by the matrix $\mathbf{M}$ consisting of the elements of the vectors $\mathbf{X}_i$ column-wise concatenated

$$\mathbf{V} = \underbrace{\begin{pmatrix} x_{10} & \cdots & x_{n0} \\ \vdots & \ddots & \vdots \\ x_{1D-1} & \cdots & x_{nD-1} \end{pmatrix}}_{=:\mathbf{M}} \cdot \begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix} \tag{4.11}$$



(a) **(b)**

Figure 4.10: Properties of the simple scalar multiplication encoding of numerical values in vectors. (a) shows the RMSE when decoding back out an approximation of the original numerical values from the vector representation. (b) shows the norm of the representation vectors.

Hence, we can use the inverse matrix $\mathbf{M}^{-1}$ to decode back out an approximation $\hat{a}_1, \ldots, \hat{a}_n$ of the original numerical values $a_1, \ldots, a_n$ from the vector $\mathbf{V}$ by $\mathbf{V} \cdot \mathbf{M}^{-1}$.
The error of that approximation depends on the dimension D of the underlying vector space and the number of numerical values encoded in the representation. To analyze this error, we randomly choose numerical values $a_1, \ldots, a_n$ uniformly from the interval $[0, 1)$ for $n = 10, 20, 50, 100, 200, 300, 400, 500, 1000$ to be encoded as well as random, unit-length vectors $\mathbf{X}_1, \ldots, \mathbf{X}_n$ representing the numerical units, decode back out the approximations $\hat{a}_1, \ldots, \hat{a}_n$ of the original values and calculate the RMSE between the actual values and their decoded approximations. Figure 4.10a visualizes the results of this analysis for four different random trials for each vector dimension $D = 64, 256, 512, 1024$. We observe that we can reliable decode back out up to $50, 200, 500$ and 1000 numerical values from the scalar multiplication encoding for vectors of dimension $64, 256, 512$ and 1024 respectively with an error in the order of magnitude of 10^{-14}, which is more than enough for our purposes. However, Fig. 4.10b shows one of the drawbacks of the scalar multiplication encoding scheme. Although we choose the numerical values to be encoded to be lower than 1, the norm of the resulting vectors becomes comparatively large for increasing vector dimension and number of numerical values encoded. This makes sense, as we sum up an increasingly large number of vectors having the numerical values as their length. This behavior will be reinforced if we use larger numerical values to be encoded. Another potential drawback is that we need to calculate an entire inverse matrix to decode back out the original information from the vector representation, whereas one of the key strengths of VSAs is that of decoding back out information using the VSA' binding operation and (pseudo-) inverse vectors (see Equation (3.11)).

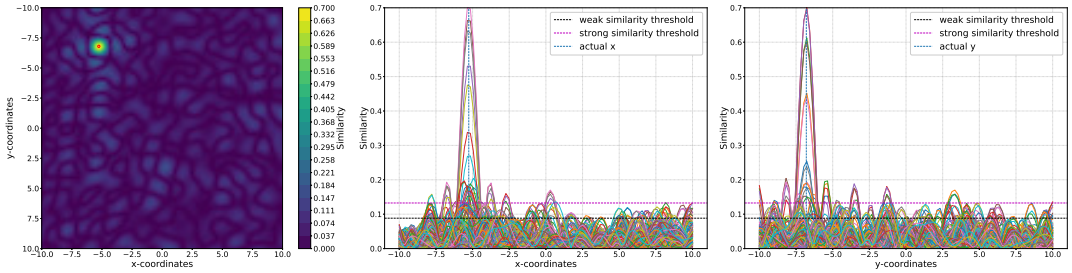
Convolutive power encoding

In this section, we present the final and, for this work most important, encoding scheme for numerical values, which we call the *convolutive power encoding*. The basic idea is to encode a numerical value $p \in \mathbb{R}$ as the power of a vocabulary vector $\mathbf{X}$ representing the corresponding unit, i.e., $\mathbf{X}^p$. While defining the power of a vector $\mathbf{X}^n$ is straightforward for an integer exponent $n \in \mathbb{N}$,

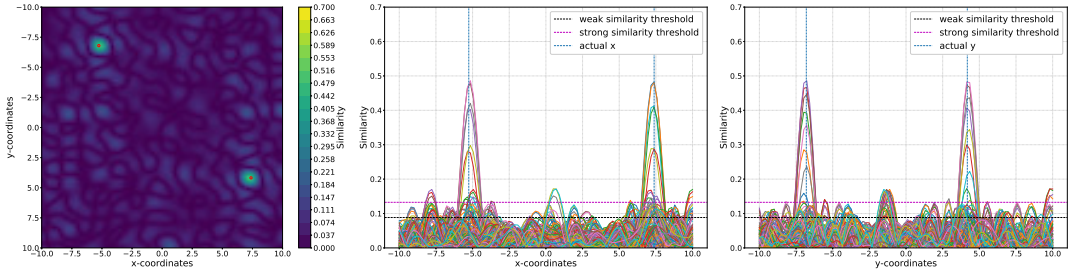
$$\mathbf{X}^n := \underbrace{\mathbf{X} \circledast \mathbf{X} \circledast \ldots \circledast \mathbf{X}}_{n \text{ times}}, \tag{4.12}$$

it is not as intuitive for real-valued exponents $a \in \mathbb{R}$. Therefore, we revisit the convolutive vector power given in definition 3.19

$$v^p := \Re\left(IDFT\left(\left(DFT_j(v)^p\right)_{j=0}^{D-1}\right)\right), \tag{4.13}$$



(a) Convolutive power encoding for one two-dimensional numerical entity



(b) Convolutive power encoding for two two-dimensional numerical entities

Figure 4.11: Visualization of the convolutive power encoding scheme for 512-dimensional representation vectors depicting the similarity between the representation vector and auxiliary comparison vectors created from a sequence of discrete values. The left plot in both rows shows a two-dimensional grid of the similarities, while the middle and right plot show the individual entities respectively. The red circles in the left plot and the dashed blue lines in the middle and right plots indicate the actual encoded values.

where $\Re$ denotes the real part a of a complex number $a+ib \in \mathbb{C}$. Equation (4.13) enables us to encode numerical, real-numbered values $p \in \mathbb{R}$ as convolutive power $\mathbf{X}^p$ and multiple numerical values $p_1, \ldots, pn$ as

$$\mathbf{V} = \mathbf{X}_1^{p_1} \circledast \mathbf{X}_2^{p_2} \circledast \ldots \circledast \mathbf{X}_n^{p_n}. \tag{4.14}$$

However, as we aim to avoid the issue of growing vector lengths when encoding a increasing number of numerical values mentioned for the scalar multiplication encoding, we restrict the vectors representing the units of the numerical values to be *unitary* vectors. Revisiting definition 3.18 and lemma 3.20, a vector $\mathbf{U}$ whose inverse element $\mathbf{U}^{-1}$ is equal to its pseudo-inverse element $\tilde{\mathbf{U}}$ is called unitary. Regarding the encoding of numerical values through their power, unitary vectors have some desirable properties (cf. lemma 3.20):

- unitary vector have unit length, i.e., $\|\mathbf{U}\| = 1$,
- the power of a unitary vector is again unitary, i.e., the set of unitary vectors $\mathscr{U}$ is closed under convolutive exponentiation,
- the product of two unitary vectors is again unitary,
- convolution with unitary vectors preserves the norm of the vector they are convolved with.

Given these properties, any vector created using Equation (4.14) representing numerical values is a unitary vector with all its desirable properties. Hence, we indeed avoid the problem of growing vector norm posed by the scalar multiplication encoding.
As mentioned previously, we are primarily interested in a way of encoding two-dimensional values in vectors, which is why we focus our analysis of the convolutive power encoding scheme on this case.

Hence, we encode two numerical values x, y, i.e., a two-dimensional entity, by generating two random, unitary vectors $\mathbf{X}, \mathbf{Y}$ representing the corresponding units and applying Equation (4.14)

$$\mathbf{V} = \mathbf{X}^x \circledast \mathbf{Y}^y. \tag{4.15}$$

To encode a sequence (x_i, y_i) for $i = 1, \ldots, n$ of two-dimensional numerical values all sharing the same units, we simply sum up a their individual encoding vectors generated via Equation (4.15), which leads to

$$\mathbf{V} = \sum_{i=1}^{n} \mathbf{X}^{x_i} \circledast \mathbf{Y}^{y_i} \tag{4.16}$$

Figure 4.11 visualizes vectors encoding one (cf. Fig. 4.11a and Equation (4.15)) and two numerical entities (cf. Fig. 4.11b and Equation (4.16)) given by two units within one vector. To generate the similarities shown in Fig. 4.11, we calculate the dot product between the vectors actually representing the encoded values and vectors $\tilde{\mathbf{V}}_i = \mathbf{X}^{\tilde{x}_i} \circledast \mathbf{Y}^{\tilde{y}_i}$ encoding a sequence of discrete sample values $(\tilde{x}_i, \tilde{y}_i)$ for $i = 1, \ldots, m$. The left plot in each row of Fig. 4.11 depicts the similarities as heat map over a two-dimensional grid. The middle and right plots in Fig. 4.11 visualize the similarities of each unit, which is similar to plotting the heat map in three dimensions as ridges and slicing them through one of the ground axes. In both rows, we observe high similarity peaks way above both similarity thresholds at the actual encoded values and significantly lower similarity values everywhere else. However, we already encounter one of the problems of this encoding scheme. Comparing the similarities at the positions of the encoded values, we observe a drop of similarity values from roughly 0.7 to 0.5 when encoding two two-dimensional numerical values instead of only one. Thus, there is a limit of how many values can effectively be encoded in such a representation before noise becomes predominant and the encoded values can not be properly recovered anymore. Such limitations regarding the number of concepts that can be represented in one vector depending on its dimension are a recurrent theme in the field of VSAs. Furthermore, we picked a 20×20 grid to encode numerical values using the convolutive power, which shows promising representational features here. Although we are dealing with real-world sensor measurements in our applications, which are naturally limited by the sensors' range or other physical constraints and/or can be re-scaled to a suitable numerical range, we still need to analyze further, if and why the chosen size of the grid is reasonable. We will further investigate these limiting factors of our representation in section 4.2.3.

4.2.2 Structured representations

Assuming we have generated a suitable vocabulary of atomic vectors encoding the entities and concepts of interest in an automotive context (cf. section 4.1) as well as several options of how to encode numerical values in vectors, we are now in the position to encapsulate the content and context of driving situations in semantic vectors. In general, we employ the SPA's algebraic operations to bind connected items and concepts together through circular convolution and to sum up independent concepts appearing alongside one another. More particularly, we combine all principles for structured representations presented so far in this thesis: simple superposition to generate unordered sets of items by summing the representational vectors, encoding of bound concepts through role-value pairs as shown in section 3.4.2 and the several approaches to encode numerical values in structured vector representations as shown in section 4.2.1. However, the setup of such a vector representation is highly dependent on the actual task to be solved as well as the data available to be encoded. Thus, we will present and investigate our structured representations in detail for two particular tasks in automotive context, namely driving context classification and vehicle trajectory prediction, in separate chapters 5 and 6. However, before we proceed to applying our vector representations to specific automotive tasks, we need to analyze their systematical limitations regarding the amount of information that can effectively be encoded in semantic vectors.

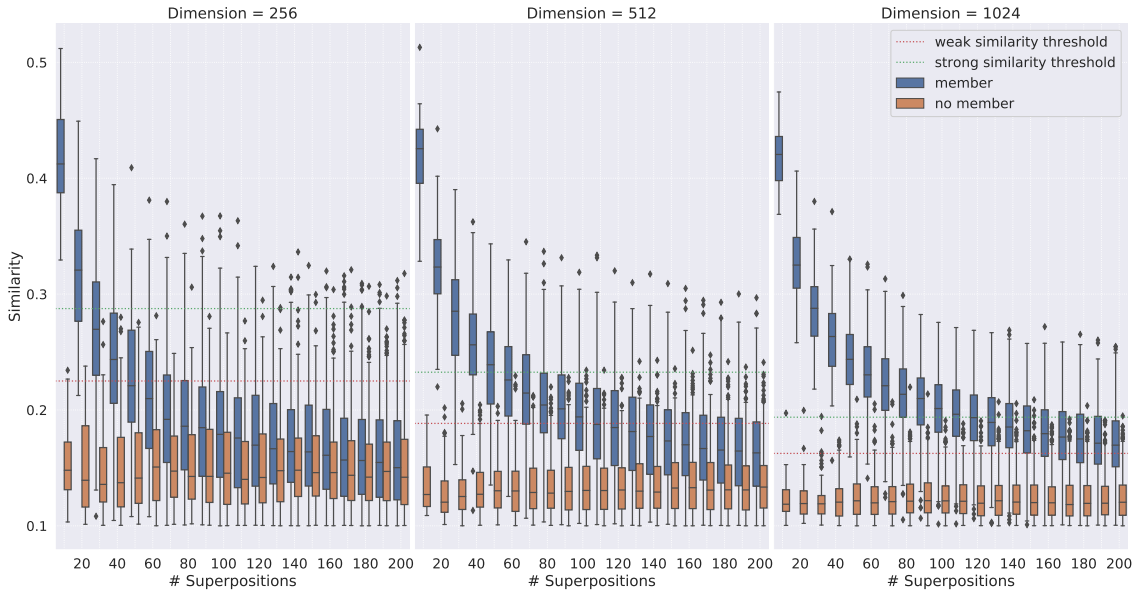


Figure 4.12: Visualization of the SPA's superposition capacity for vector dimensions 256, 512 and 1024. The blue boxes indicate the similarity between the superposition vector and its summands, the orange boxes illustrate the similarity between the superposition vector and other randomly generated vectors. The dotted lines visualize the similarity threshold based on the vector dimensionality for reference.

4.2.3 Capacity analysis - limiting factors to vector representations

In section 4.2.1 and on several other occasions, we have already seen that there are general and systematical limits for the amount of information that can be encoded in and effectively decoded from vector representations in VSAs. Such limits are a reoccurring theme and an essential feature of such architectures as they allow for the modeling of realistic cognitive phenomena. Considering human subjects for instance, the capacity to process and store information or concepts in short-term memory as well as other cognitive tasks is subject to numerical restrictions (Miller, 1956). Hence, numerical limitations of cognitive architectures like the SPA or VSAs in general are one way of modeling the numerical restrictions to cognition observed in human subjects. In the context of automated driving however, we need to analyze these restrictions imposed by the cognitive architecture applied to provide upper borders regarding the amount of information that can be stored in our vector representation. In this section, we analyze these limits with the goal of finding bounds for, e.g., the number of concepts that can effectively be stored in a single vector before the accumulation of noise makes it impossible to retrieve the original individual vectors.

Superposition capacity

First, we evaluate the capacity of superposition, i.e., the addition operation of the SPA. Superposition is used to store and combine several concept vectors $\mathbf{v}_i$ for $i = 0, \ldots, n$ in an unordered set

$$\mathbf{s} = \sum_{i=0}^{n} \mathbf{v}_i. \tag{4.17}$$

Given the properties of the SPA, we can determine if a vector of interest $\mathbf{w}$ belongs to that ordered set by calculating the similarity $\phi(\mathbf{s}, \mathbf{w})$ between the superposition vector and the vector of interest. For sufficiently high-dimensional vectors, the similarity $\phi(\mathbf{s}, \mathbf{w})$ will be close to 0 in case the vector $\mathbf{w}$ is not part of the sum. However, the more vectors we add to the superposition vector $\mathbf{s}$, the more noise accumulates in the representation and thus decreases the similarity between the superposition vector $\mathbf{s}$ and

its individual ingredients $\mathbf{v}_i$. In order to analyze how many vectors can be added together by superposition before individual vectors become irretrievable, we conducted the following experiment: assuming we want to add n vectors $\mathbf{v}_i$ for $i = 1,\ldots,n$ into a superposition vector $\mathbf{s}$ as in Equation (4.17), we randomly generate a vocabulary of $2n$ vectors $\mathbf{v}_i$ for $i = 1,\ldots,2n$ and sum up the first n members to create our superposition vector $\mathbf{s}$. Then we calculate the cosine similarity $\phi(\mathbf{s},\mathbf{v}_i)$ between the superposition vector $\mathbf{s}$ and every vector $\mathbf{v}_i$ for $i = 1,\ldots,2n$ in the vocabulary. A similar but slightly different experiment has been conducted by Wahle et al. (2012): the atomic vocabulary vectors, referred to as elemental vectors in Wahle et al. (2012), are sparse in the sense, that they mostly contain 0 elements, and the superposed vectors are normalized after adding them. Furthermore, Wahle et al. (2012) only compare the similarity $\phi(\mathbf{s},\mathbf{v}_1)$ between the superposition and the original vector with the similarity $\phi(\mathbf{v}_1,\mathbf{v}_n)$ between the original vector $\mathbf{v}_1$ and the most recently added random vector $\mathbf{v}_n$ as baseline for the expected similarity between randomly chosen vectors. In contrast, although we have already analytically derived a threshold $\varepsilon = \frac{c}{\sqrt{D}}$ for the expected similarity of randomly chosen vectors in definition 3.8, we calculate the similarity between the superposition vector and n other random vectors for reference.

Figure 4.12 shows the result of our experiment for 3 random vocabularies per superposition length containing vectors of dimension 256, 512 and 1024. The blue boxes in each figure illustrate the similarity between the superposition vector $\mathbf{s}$ and each of the individual vectors $\mathbf{v}_i$ for $i = 1,\ldots,n$ it contains, i.e., the members of the unordered superposition set. The orange boxes depict the similarity between $\mathbf{s}$ and the other vocabulary vectors $\mathbf{v}_i$ for $i = n+1,\ldots,2n$ it does not contain, i.e., the non-members. The dotted red and green lines indicate the SPA's weak and strong similarity threshold depending on the dimension of the vector space. Considering the weak similarity threshold $\varepsilon_{weak} = \frac{2}{\sqrt{D}}$, we observe that for a vector dimension of 256 the SPA allows roughly 50 items to be stored in a superposition vector. For higher vector dimensions 512 and 1024, the number of items that can be superposed increases to roughly 100 and 200 respectively. Considering the strong similarity threshold $\varepsilon_{strong} = \frac{3}{\sqrt{D}}$, the upper borders for the number of items being stored in a superposition vector are slightly more conservative with 25, 50 and 100 for vector space dimensions of 256, 512 and 1024 respectively. We also observe in our experiments that the similarity between the superposition vector and non-member random vectors is consistently below the weak similarity threshold ε_{weak} for the majority of the samples. However, once the similarity between the superposition vector and its members drops below either of the similarity thresholds for the majority of the samples, we can not distinguish between members and non-members with a sufficiently high probability. For 256 dimensional vectors for instance, we even observe that the members and non-members become nearly indistinguishably when adding more than 80 vectors. Thus, we have to choose rather conservative bounds for the number of items to be encoded in a superposition vector.

Capacity of structured representations involving convolutive powers

In the previous section, we have analyzed the SPA's capacity regarding the number of items that can be stored in an unordered set using superposition. For encoding driving situations in a semantic vector substrate, we will most likely employ more complex representation than superposition of single items alone. In this section, we thus analyze the SPA's capacity regarding the structured representations involving the convolutive vector-power representation presented in section 4.2.1. Figure 4.11 has already shown, that unbinding positions back out by querying the representation vector with sample vectors encoding discrete position examples yields high similarities for samples in the region of the encoded values and low similarities elsewhere. These low similarities for the particular examples shown in Fig. 4.11a and 4.11b are below or at most in the order of magnitude of the SPA's similarity thresholds. However, we also observe, that encoding several entities of the same type in one position vector as in Fig. 4.11b, the similarities of the true positive positions decrease compared to the encoding of only one item as in Fig. 4.11a. Hence, our capacity analysis not only has to cover the amount of objects encoded in one vector, but also the number of items per object class.

Therefore, we conduct the following experiment: assuming we want to encode n spatial entities, i.e.,

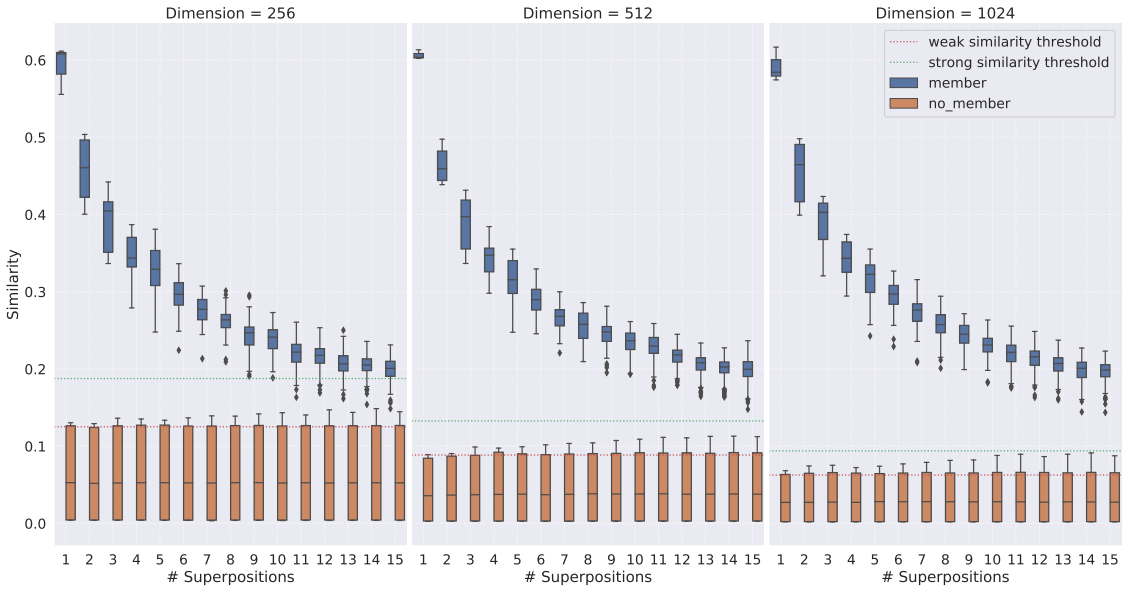


Figure 4.13: Capacity analysis for the superposition of vectors encoding spatial positions using the convolutive vector-power for varying vector dimensions.

objects o_i with two-dimensional location information (x_i, y_i) for $i = 1, \ldots, n$ as shown in Fig. 4.11 for $n = 1$ (Fig. 4.11a) and $n = 2$ (Fig. 4.11b), into a single representation vector $\mathbf{s}$, we generate a vocabulary of random vectors $\mathbf{v}_i$ for $i = 1, \ldots, n$ encoding object class labels and random unitary vectors $\mathbf{X}, \mathbf{Y}$ to encode the units of the spatial information. In contrast to the previous experiment, where we simply summed up a certain number of random vectors, we are interested in a more specific analysis, since there are several possibilities to distribute the positional values (x_i, y_i) over the available object class vectors $\mathbf{v}_i$. For instance, for a total number of two superpositions, i.e., $n = 2$, there are two possibilities to generate our representation vector, namely

$$\mathbf{s}_1 = \mathbf{v}_1 \circledast \mathbf{X}^{x_1} \circledast \mathbf{Y}^{y_1} + \mathbf{v}_1 \circledast \mathbf{X}^{x_2} \circledast \mathbf{Y}^{y_2}, \tag{4.18}$$

$$\mathbf{s}_2 = \mathbf{v}_1 \circledast \mathbf{X}^{x_1} \circledast \mathbf{Y}^{y_1} + \mathbf{v}_2 \circledast \mathbf{X}^{x_2} \circledast \mathbf{Y}^{y_2}. \tag{4.19}$$

The vector $\mathbf{s}_1$ in Equation (4.18) encodes two objects of the same type, while the vector $\mathbf{s}_2$ encodes occurrences of two different object types at the given locations. As we are working with random vectors in this experiment, we can, without loss of generality, skip the vector encoding two objects of type represented by the vector $\mathbf{v}_2$, which would yield a result equivalent to Equation (4.18). More generally, we are interested in all sets

$$C_{m,j} = \left\{ 0 < k_1, \ldots, k_m \leq n \quad \middle| \quad m \leq n \text{ and } \sum_{i=1}^{m} k_i = n \right\} \tag{4.20}$$

of natural numbers k_i summing up to n ignoring permutations of the k_i. We index the sets with j, since there potentially exist several possibilities to decompose n into sums of m natural numbers.
In our experiments, for each number n of total objects to be encoded in the vector representation, we calculate all possible sets $C_{m,j}$ (ignoring permutations) and generate random position values (x_i, y_i) for $i = 1, \ldots, n$ and a random vocabulary as described above. For each set $C_{m,j}$, we generate a representation vector

$$\mathbf{s}_{m,j} = \sum_{i=1}^{m} \sum_{l=1}^{k_i} \mathbf{v}_i \circledast \mathbf{X}^{x_i} \circledast \mathbf{Y}^{y_i} \tag{4.21}$$

as well as query vectors $\mathbf{P}_i = \mathbf{X}^{\tilde{x}_i} \circledast \mathbf{Y}^{\tilde{y}_i}$ encoding a sequence of discrete sample values $(\tilde{x}_i, \tilde{y}_i)$ for $i = 1, \ldots, M$ evenly distributed over the length of the positional encoding grid. In other words, Equation (4.21) states, that each class label $\mathbf{v}_i$ appears k_i times yielding a sum of n objects. We query the

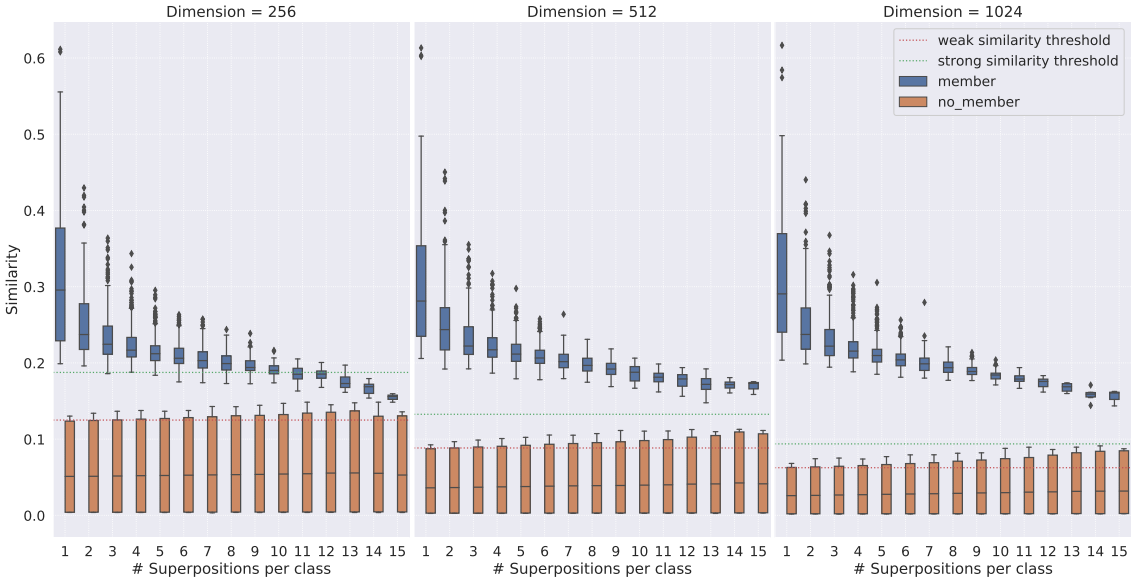


Figure 4.14: Capacity analysis for the superposition of vectors encoding spatial positions using the convolutive vector-power for varying vector dimensions. In contrast to Fig. 4.13, this figure illustrates the similarity for vectors containing spatial information for several objects of the same class.

representation vector for the position of each class by binding it to the pseudo-inverse element $\bar{\mathbf{v}}_i$ for each class label vector, i.e.,

$$\mathbf{s}_{m,j} \circledast \bar{\mathbf{v}}_i \approx \sum_{l=1}^{k_i} \mathbf{X}^{x_i} \circledast \mathbf{Y}^{y_i}, \tag{4.22}$$

and calculate the similarity with the discretized position vectors $\mathbf{P}_k$ to get

$$s_{i,k} = \left| \phi \left(\mathbf{s}_{m,j} \circledast \bar{\mathbf{v}}_i, \mathbf{P}_k \right) \right|. \tag{4.23}$$

For samples close to the originally encoded positions, i.e., $|x_i - \tilde{x}_i| < \varepsilon$ and $|y_i - \tilde{y}_i| < \varepsilon$ for a certain threshold ε, we label $s_{i,k}$ as positive similarity denoting a member of the representation vector. Otherwise, we consider the similarity $s_{i,k}$ at position $(\tilde{x}_i, \tilde{y}_i)$ not a member of the representation vector $\mathbf{s}_{m,j}$.
Figure 4.13 shows the results of this capacity analysis regarding the total number of superposed objects within the representation vector for varying vector dimensions. The blue boxes in each column illustrate the positive similarity values, i.e., for positions considered members of the representation vector, whereas the orange boxes indicate the negative similarity values of the non-member positions. The dotted red and green lines visualize the weak and strong similarity threshold for each dimension respectively. Similar to Fig. 4.11, we observe that the similarity of the non-members is in the order of magnitude of the similarity thresholds while the similarity for the member position decreases with a growing number of spatial items encoded in the vector.
Figure 4.14 shows a different evaluation of the same data showing the number of addition operations per class on the x-axis. In contrast to Fig. 4.13, Fig. 4.14 illustrates the similarity for vectors containing spatial information for several objects of the same class. That is, Fig. 4.14 illustrates the similarity of vectors containing a specific number n of superpositions per class on its x-axis independent of the total number of superpositions. We observe, that not only the similarity of the members decreases with growing number of superpositions per class, but the similarity of the non-member increases beyond the weak similarity threshold. Similar to the simple superposition capacity analysis, we consider the point in the plots where the member similarities fall below the strong similarity threshold the upper border for the maximal number of spatial objects per class to be encoded in this representational substrate. For

instance, this upper bound for 256 dimensional vectors is 10 superpositions per class, which is roughly half of the upper bound for the number of simple superpositions.

4.3 Summary

In this chapter, we presented the general approach of creating structured vector representations to encode automotive scenes. After showing the theoretical tool set in chapter 3 independent of particular applications, we focused our attention to scene representation in automotive context. The chapter's structure followed the process established by Gallant and Okaywe (2013) and is split into two main subsections focusing on the creation of vocabularies of atomic vectors (section 4.1) and generating structured, more complex representations from the atomic vectors in the vocabulary (section 4.2). We showed several approaches to generate vector vocabularies of entities encountered in automotive scenes such as traffic participants and traffic signs based on either manual design or automated learning. Thereby, we were able to encapsulate different similarity structures such as visual similarity, semantic similarity and a fusion of both. We were able to demonstrate, that we can successfully encapsulate the desired similarity structure within our vocabulary of atomic vectors. Finally, we pointed out advantages and problems for both approaches to generate these structured vocabularies with the bias imposed by human engineers and limits regarding sufficiently large and available data sets being the main issues for manual and learned vocabularies respectively. Since we consider the choice of a suitable vocabulary to be highly dependent on the particular task to be solved, we analyze the influence of varying the underlying vocabulary for the specific task of classifying the current driving context based on a vector representation of the current scene in section 5.3.4.
In section 4.2, we proceeded to investigate the creation of more complex, structured representations from atomic vocabulary vectors. One of the most important aspects was the representation of numerical values in high-dimensional vectors, where we presented three different approaches and showed their individual strengths and limitations. Particularly, we presented a novel approach to encode numerical values based the convolutive power of vectors. The biggest advantage of this representation of numerical values is that it can be bound further to other vectors as well as be used to query representation vectors about location information using pseudo-inverse elements. This approach is the main building block of the scene representation used as input to the learning models for vehicle trajectory prediction proposed in chapter 6. Finally, we analyzed the capacity of structured vector representations based on simple superposition and superposition combined with the convolutive power encoding of spatial information. Thereby, we found upper bounds for the amount of information that can effectively be encoded in such representations. These bounds have to considered in subsequent chapters to evaluate if the amount of information to be encoded in the vector representation is compliant with the bounds. This will allow a conclusive assessment of the limits of structured vector representations in automotive context. In the next chapter, we now turn our focus towards a concrete application scenario, namely the classification of the current driving context based on a structured vector representation of the current scene.

5 Instantiating a cognitive model for driving context classification

In this chapter, we describe the first application task of our driving scene representation based on semantic vectors: the classification of the current driving context. We use the tools of the SPA and the representation approaches shown in section 4.2.2 to generate semantic vectors describing the current scene. Given this representation of the driving situation, we aim to classify the current driving context, i.e., if we are currently driving on a highway, in a city or on an interurban road. We have already discussed in section 2.3.3, that high-accuracy digital maps could be used in combination with the vehicle's current position measured using GPS to detect the driving context, which is probably the most accurate approach to driving context classification. However, inferring contextual information from on-board sensory data is appealing as either a fallback option in situations when GPS is not available or if keeping an updated map with driving context information is not feasible. Furthermore, it is an interesting option and a good candidate to firstly investigate our vector representation for automotive scenes, as it seems to be a moderately complex task in the context of automated driving, yet still challenging enough, since we need to combine symbol-like processing with numerical data.
Fig. 5.2b shows a schematic overview of our approach to driving context classification and the general flow of information within the system. Environment perception happens through the ego-vehicle's on-board sensors such as cameras, RADAR and LIDAR sensors providing data either in the form of either already preprocessed object-lists or unprocessed raw sensory data. From this data, we generate our vector representation encapsulating the structure and semantics of the current scene, which will be used as input for the classification system. In the subsequent sections, we will describe the data sets used for driving context classification alongside the preprocessing steps performed to prepare the data for the task at hand, our vector representation for the current driving scene and the models used for classifying the current driving context. Finally, we will evaluate the systems performance and compare it to two performance baselines. In the evaluation stage, we will also investigate the influence of variations in the vocabulary of atomic vectors on the performance of the classification system.

5.1 Data and preprocessing

The input data used for training and evaluating the driving context classification models is real-world data gathered during test drives in the region of Munich, Germany. Depending on the test vehicle's sensor setup (Aeberhard et al., 2015), a subset of the following extrospective sensor systems is available: cameras, RADAR and LIDAR sensors. Furthermore, the dynamics of the ego-vehicle such as velocity, acceleration or the steering angle are measured using introspective sensors. While all extrospective sensors provide preprocessed lists of dynamic objects such as cars or pedestrians, the camera system additionally detects static objects such as traffic signs. Furthermore, the raw images of both, the front and rear camera are also available.
In this chapter, we focus on the ego-vehicle's dynamics and the information provided by the camera-system as the only extrospective sensor, since there are no object-lists with information fused from several sensor sources available. Furthermore, the camera-system is present in all available test traces and its data is most informative regarding categories of dynamic objects while being the only system that provides information about traffic signs. The camera provides class labels for each object such as *car* or *pedestrian* for dynamic objects as well as the type of each detected traffic sign. Apart from these object

(a) (b)

Figure 5.1: Two example images illustrating a change of the current driving context as indicated by (a) a traffic sign marking the exit of city and (b) a traffic sign marking the entrance to a highway. The traffic signs are highlighted by a red rectangle.

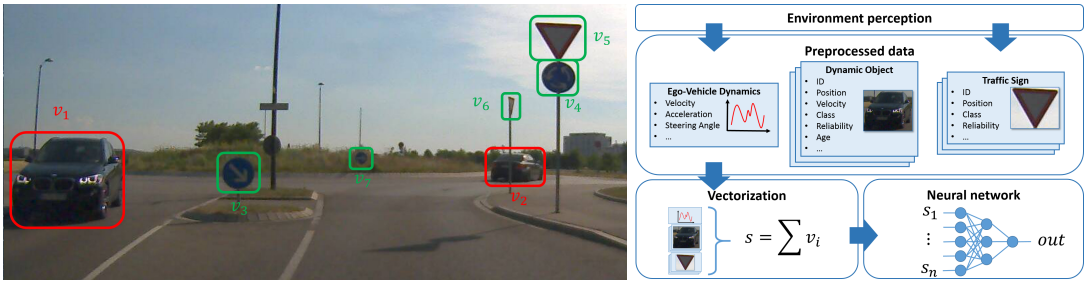
classification labels, the camera sensor provides estimations for other object features such as position and orientation relative to the ego-vehicle and furthermore dynamic information such as velocity or acceleration for moving objects only. We divide the available data into a training and a test set, which contain roughly 27 min and 18 min of driving data respectively.

5.1.1 Data labeling

To enable automated training of any supervised learning system, the training data needs to be labeled. Since there is no labeled data set publicly available neither was the driving data labeled regarding the current driving context, we had to label our own data set. In this chapter, we focus on three driving context labels only, namely *city*, *interurban* and *highway*. Hence, the goal is to assign one of these labels to all data points in our driving data set. We achieved this goal by manually labeling the driving context by inspecting the images provided by the ego-vehicle's on-board camera systems. To avoid including human bias into the labeling regarding what sorts of situations belong to each of the labels, we restricted the labeling process to finding traffic signs indicating a change of driving context and assign the respective labels to all data points between these context switches. Figure 5.1 shows two example situations where the ego-vehicle transitions from one driving context to another, namely from city driving to interurban driving as indicated by the city exit traffic sign in Fig. 5.1a and from interurban context to highway driving as indicated by the highway entrance traffic sign in Fig. 5.1b. This manual labeling process yields 53.6 %, 18.9 % and 27.5 % of the samples in the training set belonging to *city*, *interurban* and *highway* driving respectively while 23.3 %, 5 % and 25.9 % of the samples in the test set belong to the same driving context labels.

5.2 Representation and models

In this section, we describe our approach to encode driving situations in semantic vectors and to classify the current driving context from these vectors. We present the different types of information we encode in the vector representation using several underlying vocabularies to evaluate their impact on the model classification performance. Finally, we describe the models trained to produce the actual context classification.



(a) **(b)**

Figure 5.2: Overview of the driving context classification system. (a) shows one example scene with objects of interest such as cars and traffic signs highlighted by colored bounding boxes. (b) illustrates the learning system's architecture and flow of information.

5.2.1 Scene representation in vectors

For the task of classifying the current driving context from measurements provided by the ego-vehicle's on-board sensors, we encapsulate three types of information in our vector-based scene representation, namely certain dynamics of the ego-vehicle, dynamic objects and traffic signs. Information on all of these entities is provided as preprocessed object-lists as described in section 5.1. In subsequent sections, we describe the process of converting the input data into a vector representation for each of these categories of information.

Ego-vehicle dynamics

Regarding the dynamics of the ego-vehicle, we encapsulate the current velocity v, acceleration (a_x, a_y) split in lateral and longitudinal direction with respect to the ego-vehicle's coordinate system as well as the angle $\alpha_{steering}$ of the steering wheel, and the steering angle α_{axle} of the front axle. Since all of these units are intangible concepts, we encode them by assigning a random vocabulary vector to each of them. To represent the numerical value corresponding to each unit, we employ the simple scalar multiplication encoding introduced in section 4.2.1. We illustrate this procedure for the current velocity of the ego-vehicle: let $\mathbf{VELOCITY} = (v_0, \ldots, v_{D-1})$ with $v_i \in \mathbb{R}$ for all $i = 0, \ldots, D-1$ be the randomly chosen, normalized vocabulary vector representing the unit velocity, we encode the value v of the ego-vehicle's velocity as $v \cdot \mathbf{VELOCITY}$. Furthermore, we normalize all scalar values to the range $[-2, 2]$ to keep the length of our vectors limited. For vectorization of the two-dimensional acceleration values in longitudinal (x) and lateral (y) direction, we use the encoding based on sine and cosine functions with different spatial frequencies and offsets employing the function λ introduced in Equation (4.9) in section 4.2.1. Hence, to obtain a vector representation of acceleration in longitudinal (x) and lateral (y) direction (a_x, a_y), we use the encoding $\lambda(a_x, a_y)$, which leads to normalized, nonzero, similar vectors with information distributed over all elements. To achieve the encoding of all dynamics of the ego-vehicle, we sum up the vectors representing the individual values, i.e.,

$$\begin{aligned} \mathbf{EGO}_t = {} & v \cdot \mathbf{VELOCITY} + \alpha_{steering} \cdot \mathbf{STEERING} \\ & + \alpha_{axle} \cdot \mathbf{AXLE} + \lambda(a_x, a_y) \circledast \mathbf{ACCELERATION} \end{aligned} \tag{5.1}$$

Traffic participants

The camera-based classification system is able to distinguish five different traffic participant categories, namely *bicycle*, *car*, *motorcycle*, *pedestrian* and *truck*. Furthermore, there are additional categories *stationary* for stationary objects and *unknown* for objects, where none of the aforementioned labels can be assigned to. We generate vocabulary vectors for each traffic participant class. In the simplest form

of a vector representation, we could simply add the vocabulary vector once for each appearance of the corresponding object in the scene to the current representation vector. However, this representation just encodes that there are certain objects present somewhere in the current scene without any additional information. To encode a more detailed representation of the current scene with additional information on each traffic participants position, we use the function λ from Equation (4.9) again to map each dynamic object's position (p_x, p_y) in longitudinal (x) and lateral (y) direction relative to the ego-vehicle's coordinate system to vector form $\lambda(p_x, p_y)$. Subsequently, we bind the resulting vector encoding the object's position to the vector representing the object's category. However, positional information for each traffic participant might not be informative enough regarding the task of distinguishing between several driving contexts. One quite unique feature of, for instance, highway driving is the fact that almost all other traffic participants drive in similar direction as the ego-vehicle. Therefore, we create additional vocabulary vectors encoding the orientation of dynamic objects relative to the ego-vehicle for three discretized categories, namely **SAME**, **OPPOSITE** and **OTHER**, and bind this orientation information to each dynamic object as we did for position information. If we want to jointly attach those two pieces of information to one object, we need to generate two additional vocabulary vectors **POSITION** and **ORIENTATION** to impose structure. For instance, a car detected at position (p_x, p_y) with approximately the same orientation as the ego-vehicle would lead to the following vector representation

$$\mathbf{CAR} + \mathbf{CAR} \circledast \mathbf{POSITION} \circledast \lambda\,(p_x, p_y) + \mathbf{CAR} \circledast \mathbf{ORIENTATION} \circledast \mathbf{SAME}. \tag{5.2}$$

Let all traffic participants in the current scene be given as obj_i for $i = 0, \ldots, n$, we get the vector encoding traffic participants as

$$\begin{aligned}\mathbf{OBJ}_t = \sum_{i=0}^{n} &\mathbf{TYPE}_{obj_i} + \mathbf{TYPE}_{obj_i} \circledast \mathbf{POSITION} \lambda\,(p_{x,i}, p_{y,i}) \\ &+ \mathbf{TYPE}_{obj_i} \circledast \mathbf{ORIENTATION} \circledast \mathbf{ORIENT}_{obj_i},\end{aligned} \tag{5.3}$$

with $\mathbf{TYPE}_{obj_i}$ denoting the vector representing the object's category and $\mathbf{ORIENT}_{obj_i}$ denoting the vector representing the object's orientation.

Traffic signs

The ego-vehicle's camera-system is able to recognize a significant amount and variety of traffic signs. We assign a vocabulary vector to each possible traffic sign label representing the corresponding traffic sign. However, to generate a vector representation of all traffic signs relevant for the current driving context, it is not sufficient to simply add each currently visible sign to the current scene representation. In contrast to traffic participants, there are many traffic signs, which are not only valid while being visible but stay relevant for the current driving context until withdrawn or overwritten by another sign. For instance, if we observe a traffic sign indicating a speed limit of 30 km/h, that speed limit is valid and relevant until we encounter another traffic sign indicating a different speed limit. Therefore, we implemented a simple form of memory for a certain subset of traffic signs relevant to the task of driving context classification even after disappearing from the field of view such as traffic signs indicating speed limits or traffic signs indicating the entrance or exit of a city or highway. Due to the fact that the camera system is not immune to false detections, we implemented a decaying memory, to avoid relying too much on false detections and to allow the system to consider other cues as well. We realized this decay by the function

$$\sigma(t, \tilde{t}) = \gamma^{(\tilde{t}-t)}, \tag{5.4}$$

with a scaling factor $\gamma < 1$ and $\tilde{t} > t$. Furthermore, we include a simple logic to replace traffic signs in the representation if they are overwritten by more recently perceived ones. For instance, recently observed traffic signs indicating a new speed limit overwrite previously seen speed limits and a sign indicating a

city entrance withdraws a memorized highway sign. Assuming we have encountered a particular sign $\mathbf{S}_i$ at time t_i, we achieve the vector encoding the traffic signs at time t through

$$\mathbf{SIGNS}_t = \sum_{i=0}^{n} \sigma(t,t_i) \cdot \mathbf{S}_i + \sum_{j=0}^{m} \tilde{\mathbf{S}}_j, \tag{5.5}$$

for traffic signs $\mathbf{S}_i$, which need to be memorized and traffic signs $\tilde{\mathbf{S}}_i$, which are only valid while they are visible.

Putting it all together

In the previous sections, we have presented the encoding process at time t to generate vectors representing the dynamics of the ego-vehicle in $\mathbf{EGO}_t$, the traffic participants or dynamic objects in the scene in $\mathbf{OBJ}_t$ as well as the traffic signs either currently visible or memorized ones from previous observations in $\mathbf{SIGNS}_t$. To finally generate the vector representing the current scene, we simply sum up these individual pieces of information to

$$\mathbf{SCENE}_t = \mathbf{EGO}_t + \mathbf{OBJ}_t + \mathbf{SIGNS}_t. \tag{5.6}$$

We us these scene vectors as input for our classification model to classify the current driving context based on the current representation of the driving scene.

5.2.2 Classification model

In this section, we describe the model we use for classifying the driving context based on the vector representation of the current scene. Our main model is a simple single-layer neural network implemented using the Nengo (Bekolay et al., 2014) software suite, which is typically used to create large-scale neural models (Eliasmith, 2013), but also allows the implementation of traditional feed-forward neural networks. For the task of driving context classification, we use the LIF neuron model and N neurons in the hidden layer. We employ supervised learning, i.e., we present the model with the vector representation as input and our manually generated driving context labels as output. That is, we input the vectors encoding the current driving scene $\mathbf{v} = (v_0, \ldots, v_{D-1})$ into N neurons (encoding), and the output of the network (decoding) will be the classification $\mathbf{c}$ of the current driving context. To encode the current scene vector in the activity $\mathbf{a}_i$ of the N neurons, we employ the principles of the NEF:

$$\mathbf{a}_i = \mathbf{G}\left(\sum_j \mathbf{e}_{i,j} v_j + \beta_i\right), \tag{5.7}$$

where $\mathbf{G}$ is the non-linearity of the neuron model and $\mathbf{e}_{i,j}$ and β_i are randomly generated to produce a uniformly distributed range of maximum firing rates and intercepts. To decode the classification of the current driving context from the neurons' activity $\mathbf{a}_i$, we employ

$$\mathbf{c} = \sum_{i=1}^{N} \mathbf{d}_i \mathbf{a}_i \tag{5.8}$$

We leave $\mathbf{e}_{i,j}$ and β_i at their randomly initialized values and use Nengo's default least-squares optimization to calculate the optimal decoder values $\mathbf{d}_i$.

5.3 Experiments

In this section, we describe the experimental setup, models and metrics to analyze and evaluate our driving context classification model. We present several reference models and human performance on the task of driving context classification in order to compare our model to. Furthermore, we analyze the influence of variations in the underlying vector vocabularies on the classification accuracy.

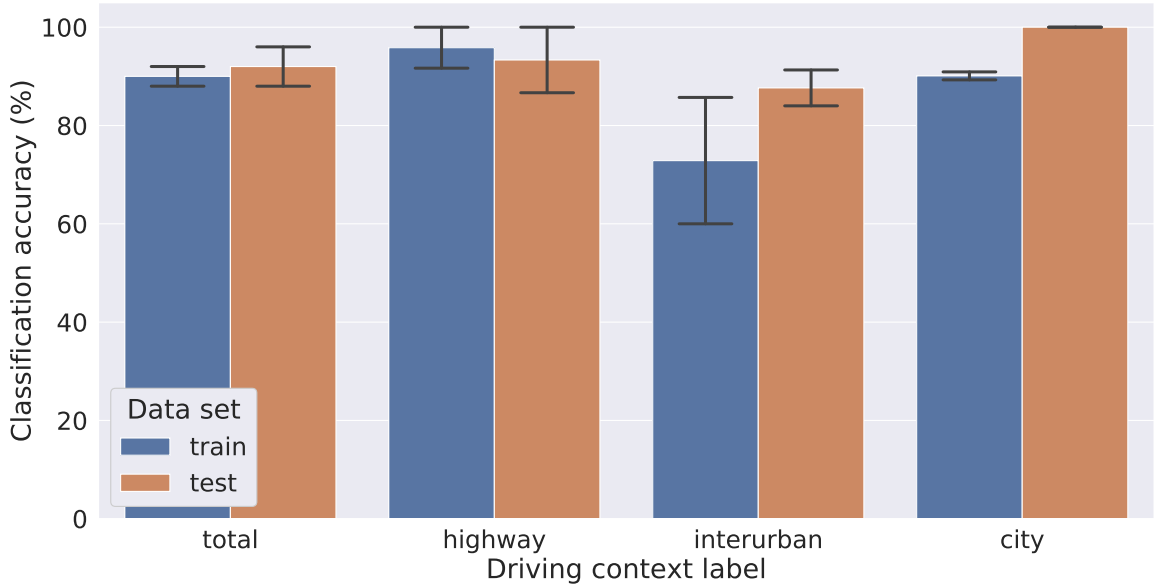


Figure 5.3: Classification performance of 2 human subjects on 50 examples selected randomly from each the training and test set.

5.3.1 Performance baselines

To get a better understanding of the quality the driving context classification model achieves based on our vector representation, we compare it to several other performance baselines. In this section, we describe these reference models and the kind of data they use in comparison to the information encoded within our vector representation of the driving scene. We use three baselines to compare our models' performance to: human performance, a multi-layer neural network implemented using the Keras library (Chollet, 2015) using our vector representation as input data and a CNN using raw images to classify the current driving context.

Human performance

One of the best and most competitive baselines for any learning model is the performance of humans in the task given to the learning system. Hence, we also aim to compare our classification models' performance to that of human subjects on the same task. However, we need to make sure that the data provided to the human subjects is as similar as possible to the information exposed to the learning system to make the results as comparable as possible. The most intuitive way for human subjects to perceive and hence classify the current driving context would be through visual information, i.e., raw camera images. However, the vector representation of the driving scene abstracts away most of the unnecessary visual features provided by the camera images and thus is very different information. On the other hand, the raw numerical values of the vector representation are hard, if not impossible, to comprehend for human subjects. Another intuitive way for humans to perceive information is language. Therefore, we created human-readable versions of our input vectors in the form of written text by storing the label of the vocabulary vector followed by the encoded numerical value. We presented a subset of 50 random samples for each data set to two human subjects asking for their classification guess. To make this process as similar to the automated learning process pursued by the neural network, the human subjects had to classify samples from the training data set first. After every sample, the subject was informed if the provided classification was correct or, in case of wrong classifications, which label would have been the correct answer. In this way, we aim to provoke a steep learning curve for the human subjects

before switching to the test set and replicate the learning process of the neural models as closely as possible. Figure 5.3 shows the performance for the two human subjects on both the training and test sets on the individual labels and their overall performance. These results indicate that an overall classification performance of roughly 92 % and at least 80 % for the individual driving context labels are a reasonable baseline for the neural models to be compared to.

Multi-layer neural network

The first reference model to compare our spiking neuron based driving context classification model to is a more traditional feed-forward, multi-layer neural networks of rate neurons. This model also uses our vector representation of the current driving scene as input and is trained in supervised fashion to classify the current driving context. Here, we use a network consisting of several fully-connected hidden layers followed by a classification layer producing the model's predictions. This model is quite similar to our classification network with the most significant differences being the neuron models, i.e., ReLU (Rectified Linear Unit) in contrast to the spiking LIF neuron model in the Nengo network version and the training procedure. Both models employ supervised learning techniques and while the Nengo model employs simple least squares optimization, the multi-layer Keras-model employs the classical backpropagation algorithm using gradient descent.

Visual driving context classification using a CNN

As mentioned earlier, a very intuitive way for us as human to classify the current driving context is by using visual information, i.e., raw images of the ego-vehicle's camera systems. Many deep learning models, especially CNNs, are inspired by the structure of the human visual cortex and have achieved remarkable results on visual classification tasks. Hence, we use a CNN using the raw camera images as input data as our third and final reference model to compare our context classification network to. We use a network architecture similar to the one employed in section 4.1.3 to classify traffic signs based on visual input since this network architecture has been used successfully for classification tasks based on visual input. The model is a multi-layer neural network with three convolutional layers each followed by a pooling layer with one additional fully connected layer and the final classification layer. The network is trained using backpropagation and the classical gradient descent.

5.3.2 Model training

In this section, we describe the process of training our context classification model as well as the aforementioned reference models.

Nengo model training

As mentioned in section 5.2.2, we use Nengo's default least squares optimization to solve for the decoders $\mathbf{d}_i$ in Equation (5.8). However, in order to generate the optimal decoders for the complete training set, we need to solve Equation (5.8) for all samples, i.e., each pair of scene vectors and true context labels $(\mathbf{v}_j, \mathbf{c}_j)$ for $j = 0, \ldots, M$ in the training data set. By default, this results in a matrix $A = (\mathbf{a}_{i,j})$ for $i = 0, \ldots, N$ and $j = 0, \ldots, M$, with N being the number of neurons in the population encoding the current scene vector and M being the number of samples in the training set, which in our case is roughly 350 000 samples. Hence, in order to solve for the optimal decoders, Nengo needs to generate a giant matrix A of neural activities and store it in the system's memory in addition to the high-dimensional vectors in the training set. That results in high memory requirements for the training process, which imposes restrictions on the computational hardware and could thus be prohibitive. Therefore, we also implemented a variant of the training process, that calculates the matrix A in blocks for subsets of the training samples of size $b < M$ and thus solves for the decoders $\mathbf{d}_i$ for $i = 0, \ldots, b-1$ and then for the next block $i = b, \ldots, 2b-1$ until we have calculated all decoders. That slight adjustment is mathematically equivalent to the default least

squares optimization process to solve for the decoders, but only consumes memory resources in the order of $\mathcal{O}(N \cdot b)$ instead of $\mathcal{O}(N \cdot M)$. The only other difference is that we use regularization in the default least squares optimization method. This is typically used to impose a certain amount of noise to make the learning population generalize better. In our case however, we found that dropping this regularization leads to improved classification results (see section 5.3.3)

Multi-layer neural network training

We implemented the multi-layer, feed-forward neural network as a reference model to compare our Nengo classification model to using the Keras library (Chollet, 2015). We chose a simple feed-forward network architecture with two fully connected hidden layers consisting of 1500 and 500 ReLU neurons respectively each followed by a dropout layer and a final classification layer. In contrast to the Nengo model, we use backpropagation and stochastic gradient descent to adjust the networks neural weights.

CNN training

The CNN for image-based context classification is implemented using the Tensorflow library (Abadi et al., 2016). We use a similar architecture as the one used for traffic sign classification in section 4.1.3 with 9 layers.

Layer	Type	# maps and neurons	kernel
0	input	3 maps of 256×144 neurons	
1	convolutional	100 maps of 250×138 neurons	7×7
2	pooling	100 maps of 125×69 neurons	2×2
3	convolutional	150 maps of 123×67 neurons	3×3
4	pooling	150 maps of 61×33 neurons	2×2
5	convolutional	250 maps of 59×31 neurons	3×3
6	pooling	250 maps of 29×15 neurons	2×2
7	fully connected	300 neurons	1×1
8	fully connected	3 neurons	1×1

Table 5.1: Layer by layer architecture of our reference CNN for driving context classification.

This architecture is shown in table 5.1 consisting of three consecutive convolutional layers each followed by a max-pooling layer and two fully-connected layers for the final classification. The network has two additional dropout layers, one after the convolutional layer part of the network dropping 25 % of the neurons activity and one before the final classification layer dropping 50 % activity. The model is trained using backpropagation and the classical gradient descent algorithm. We also employed early stopping since the model's validation loss did not significantly decrease after roughly 3500 epochs and the model's performance did not improve for longer training phases.

5.3.3 Evaluation of the classification performance

In this section, we analyze and evaluate the performance of our context classification model and compare it to the several baselines established in section 5.3.1. Figure 5.4 shows the classification performance for the Nengo model, the multi-layer neural network implemented in Keras, the CNN network based on raw camera images as well as the performance of our human subjects for reference. In this section, we use random vocabularies of dimension 512 as basis for all the context classification models using our vector representation as input data. The two variants of the Nengo model referred to as *nengo* and *nengo improved* in Fig. 5.4 differ in calculating the decoders on the complete set of training samples (*nengo*) and on blocks of sample subsets (*nengo improved*). Furthermore, the *nengo* variant uses regularization in the least squares optimization, namely 10 % of the neurons activity, instead of no regularization in the

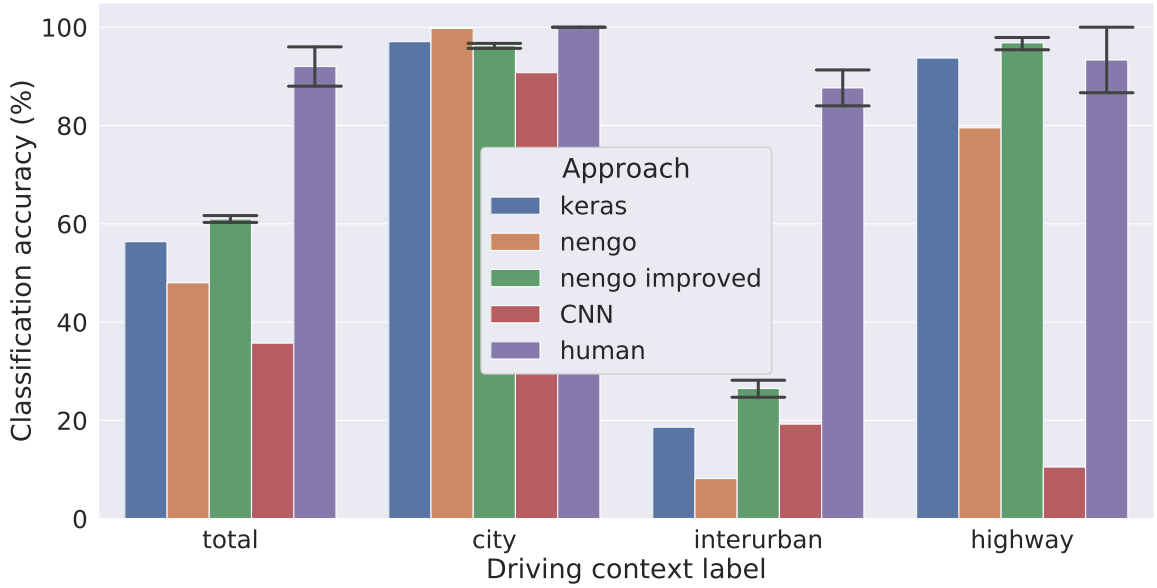
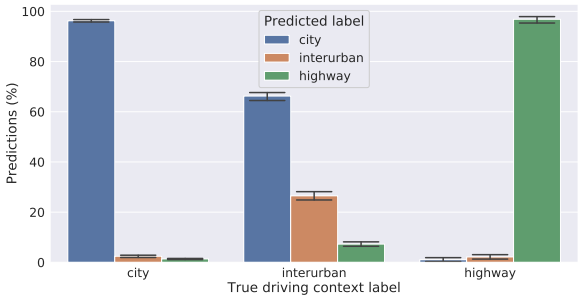


Figure 5.4: Visualization of the performance of our driving context classification model and the comparison baselines for reference.

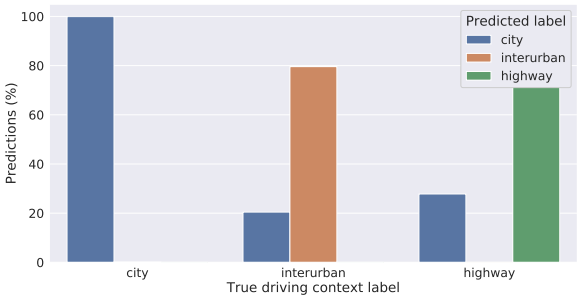
nengo improved model. We observe that all automated learning models perform overall worse than the human performance baseline with the Nengo model with improved and memory-efficient decoder calculation performs best with roughly 62.6 % classification accuracy overall. The second best classification performance achieves the Keras multi-layer neural neural network trained also on our vector representation performing just slightly worse than the best Nengo model with a total classification accuracy of 56.4 %.

However, analyzing these results in more detail, we observe that the decreased overall classification performance for both, the Nengo and Keras models is due to their poor classification performance on the *interurban* context category achieving only 27 % and 18.6 % in comparison to the 87 % classification accuracy achieved on average by the human subjects. For the other context categories, *city* and *highway*, both models achieve competitive results comparable to the human classification performance baseline. In order to further analyze this performance issue, we have a look at the composition of our data set. The most critical problem is that our data is limited. There are only 18 min of test data available, which are recorded from just two test drives. The first are 4 min are continuous, mostly in the *city* environment, with 30 s of *interurban*, that are perfectly recognized. The second part of the test set, again a continuous drive, contains, after starting in the *city*, a long stretch of about 8 min *interurban* with heavy stop-and-go traffic at low speed. The training set, however, does not contain data samples with driving situations comparable to that *interurban* part in the test set. Furthermore, the *interurban* category is probably the most difficult to predict since there are *interurban* samples that could be mistaken with either of the other two context classes depending on the situation. Figure 5.5a visualizes the amount of predictions of the Nengo network for all the context classes compared to the true label of the data sample and supports these observations. Although the majority of false classifications of the *interurban* category is misclassified as *city*, there is also a significant amount of samples mistaken for *highway*. On the other hand, *interurban* is the category with the least amount of samples in the training data set as only 18.9 % of the training samples belong to the *interurban* class. Thus, we assume that a more balanced and/or larger data set could be essential to tackle this issue.

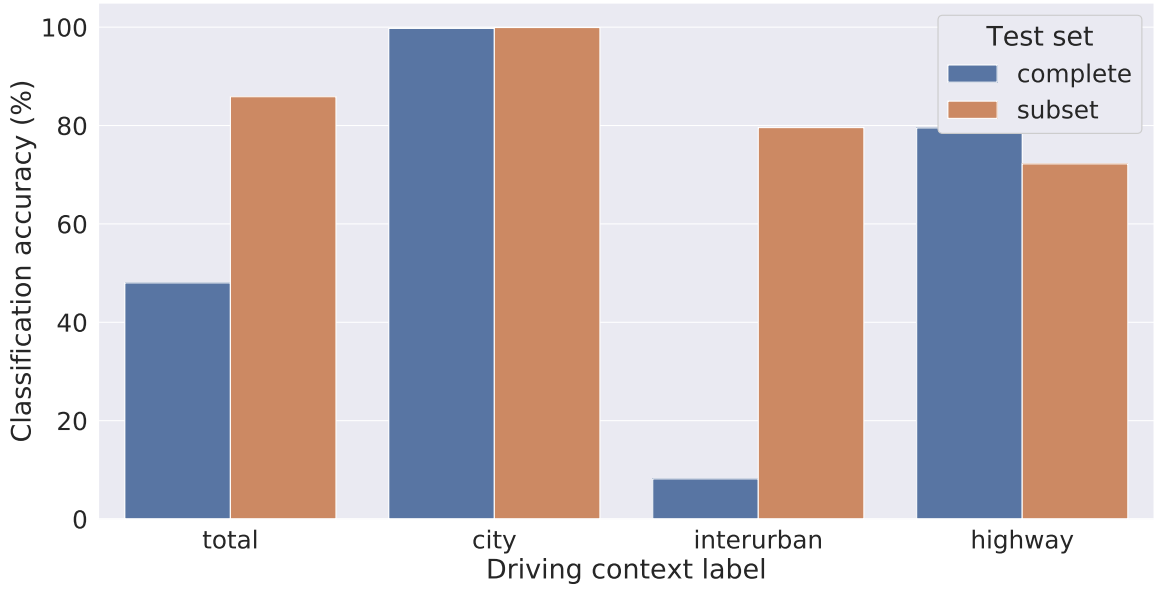
Hence, we evaluated our classification model on a subset of the test set simply removing that *interurban* part both model variants struggle to predict. Figure 5.5c shows the results of the original Nengo



(a) Complete test data set



(b) Subset of the test data set



(c) Complete vs. test subset

Figure 5.5: Visualization of the driving context predictions made by the Nengo model compared to the true labels. Figure (a) shows the results for the complete test data set, while Fig. (b) shows the model's predictions on the test subset compared to the true labels. Figure (c) shows the model's performance on the subset in comparison to the complete test set.

classification network (i.e., trained without the memory-efficiency and regularization improvements) on the complete test set as well as on the subset, while Fig. 5.5b shows a similar evaluation for the subset of the training set as Fig. 5.5a for the complete test set. We observe that, even for this non-optimized model variant, the classification accuracy of the *interurban* category significantly increases from 8.1 % to 79.5 % when switching from the complete test set to the smaller subset. Thus, we consider this a strong hint that a more balanced and larger training data set will be able to improve the poor classification performance on the *interurban* driving context category and therefore the overall accuracy of the model.

Revisiting Fig. 5.4, the CNN classification model based on the raw camera images performing poorly in comparison to the other models achieving only 35.7 % overall classification accuracy is sort of an exception. While the model achieves 90.7 % classification accuracy on the *city* context label, its performance deteriorates down to 19.2 % and 10.5 % for the *interurban* and *highway* labels respectively. This is even the best classification performance we were able to achieve with this model: when training the network for a larger amount of epochs, which, in theory, should improve the model's performance, its

(a) Interurban **(b)** City

Figure 5.6: Examples of similar looking images in the test set with different driving context labels.

performance deteriorates even further. Instead of generalizing better, the model tends to overfit and just learns to simply assign the same context label to all samples. We assume, that there are too few and too similar images in the training data set for the model to sufficiently generalize beyond these samples. For instance, Fig. 5.6 shows two raw images from the training data set with rather similar visual features but with different labels. Figure 5.6a shows a training sample of the *interurban* category while Fig. 5.6b shows a *city* sample. Hence, we conclude that for the particular task of driving context classification performed on a rather small data set, it is beneficial to omit unnecessary visual features and complexity and instead use our abstract vector representation of the scene as input for an automated learning system.

5.3.4 The influence of varying vocabularies

In section 4.1, we have already seen several ways to generate a vocabulary of atomic vectors for scene representation in automotive context. For the evaluation in section 5.3.3, we focused solely on randomly generated vector vocabularies for the sake of simplicity. Here, we are interested how varying vector vocabularies encapsulating a similarity structure of the encoded entities influence the performance on the task of driving context classification.

Random vocabulary baseline

The evaluation in section 5.3.3 was conducted on randomly generated vocabularies containing 512 dimensional vectors. However, the structured vocabularies generated in section 4.1 consist of 300 dimensional vectors. Therefore, we first analyze how the reduced dimensionality of the underlying vector vocabulary influences the classification performance of our model. Figure 5.7 illustrates the performance of our Nengo driving context classification model for 10 different randomly chosen vocabularies containing 300-dimensional and 512-dimensional atomic vectors. Although we observe a slight deterioration of the classification accuracy when reducing the dimension of the vocabulary vectors, the difference is not statistically significant. Hence, we use the classification performance on the 300-dimensional random vector vocabularies as baseline for comparison for models built upon structured vocabularies encoding different types of similarity.

Structured vocabularies

In this section, we analyze the impact of encapsulating a similarity structure within the vocabulary of atomic vectors used to generate the scene representation vectors. We hypothesize, that encapsulating a similarity structure within the vocabulary of atomic vectors is beneficial in terms of performance for our driving context classification model based on the scene representation built upon these vocabularies. We use the different vocabularies presented in section 4.1 to impose different similarity structures,

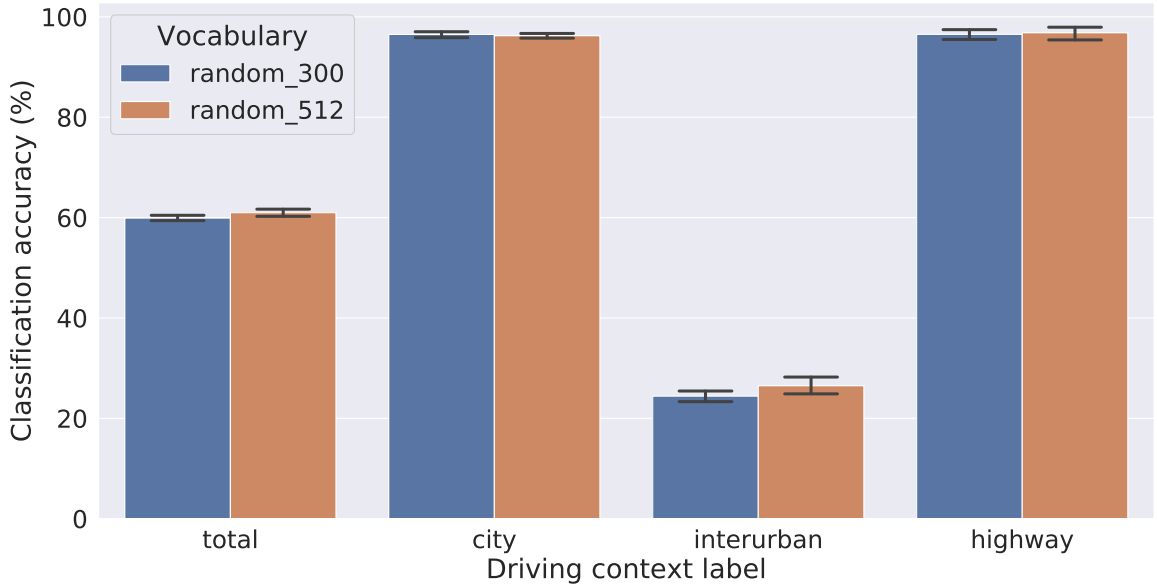


Figure 5.7: Comparison of the driving context classification model for 300-dimensional and 512-dimensional vectors.

namely a manually generated semantic vocabulary (see section 4.1.4), a learned visual vocabulary (see section 4.1.3) as well as a fused visual-semantic vocabulary combining the visual and semantic similarity structures in one vocabulary (see section 4.1.5). However, the aforementioned structured vocabularies do not contain all entities encoded in the random vocabularies. For instance, the GTSRB data set, which was used to generate the visual vocabulary for traffic signs, contains only 42 traffic sign classes, which covers only $\frac{1}{6}$ of the traffic signs the ego-vehicle's camera system is able to detect. This allows us to encode the structure of, for example, speed limit signs as well as priority signs, which occur in both, our structured vocabulary and the driving data. On the other hand, we do not encode the traffic sign indicating a speed limit of 130 km/h (as it is not contained in the GTSRB data set), nor the signs indicating a highway entrance, which occur in the driving data and have a significant impact on the driving context classification. Hence, we adapt the baseline random vocabularies by replacing the vectors with their counterpart within the structured vocabulary if such a vector exists and otherwise leave it unchanged. Thus, we are only able to encode the desired structure for the subset of entities encoded in the vocabularies.

Figure 5.8 illustrates the classification performance of our driving context classification model for the three different approaches to generate structured vocabularies as well as the random vocabularies as baseline. It shows the performance over 10 randomly generated vocabularies used as baseline as well as the starting point for structured vocabularies. We observe the general trend that both, the manually generated semantic vocabulary as well as the vocabulary encapsulating visual similarity deteriorate the model's classification accuracy. While the semantic similarity is able to slightly improve the classification accuracy at least for the *highway* category, the classification performance for the model using the visual vocabulary decreases significantly for all context categories. On the other hand, the vectors combining visual and semantic similarity in one vocabulary are able to slightly improve the model's accuracy over the random vocabularies. Although the model's performance based on this visual-semantic vocabulary decreases slightly for the *city* category, its performance on the *highway* class is comparable and, on average, slightly improved for the *interurban* context category. For some individual vocabularies, the additional structure significantly improves results. In the most extreme case, we observe an unchanged 96 % prediction accuracy of *city*, while *interurban* increases from 23 % to 30 % and *highway* from 95 % to 99 %. On the other hand, there are also vocabularies without substantial changes and, in some cases,

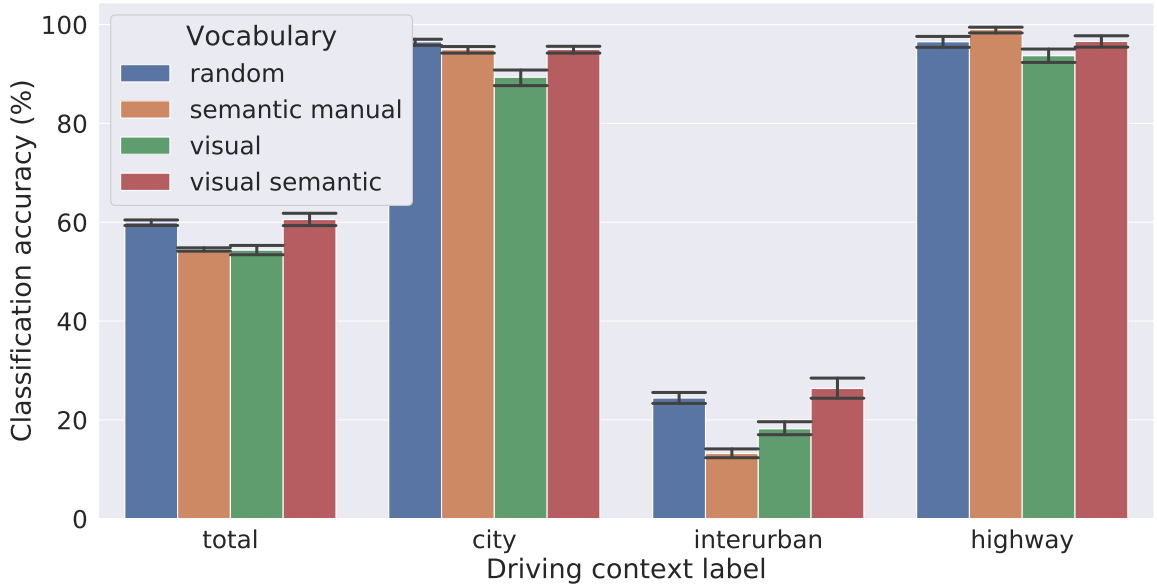


Figure 5.8: Visualization of the Nengo model's classification performance for several structured vocabularies compared to the baseline performance on randomly generated vocabularies.

even decreased classification performance. For the most pronounced example, accuracy decreases in all three classes with a deterioration of 1 and 2 percentage points for *city* and *highway* respectively and even 5 percentage points for the *interurban* category.
In conclusion, encapsulating visual-semantic structure produces a measurable effect on the model's classification performance for several vocabularies and improves average results compared to the vocabularies encoding individual parts of the structure (i.e., only visual or semantic). We observe a difference in classification accuracy of up to 7 %. However, the impact varies depending on the underlying original vocabulary: for some vocabularies, accuracy decreases over all classes, in others it increases. On the other hand, since this average is only based on 10 vocabularies (due to computational limitations), this visual-semantic structure does not perform significantly better than the baseline vocabulary. Furthermore, there is no clear pattern observable such as an increase in accuracy especially for vocabularies that underperformed without additional structure, or a decrease for vocabularies that performed well using the original randomly generated vectors.

5.4 Summary

In this chapter, we have presented a novel approach to a data-driven driving context classification model based on our vector representation of driving scenes. We proposed a vector representation of the current driving situation encapsulating a mixture of symbol-like information such as traffic signs or the type of traffic participants as well as numerical information such as position or velocity of dynamic objects. We showed that our learning model using SNNs in the Nengo framework outperformed the feed-forward reference neural network implemented in Keras as well as a CNN based on raw camera images. We assume that we successfully abstracted away unnecessary and too complex visual features from the representation for being able to learn driving context classification from a rather small vocabulary. The CNN would most likely require a much larger training data set containing a great variety of different examples of driving context examples under several conditions (e.g., daylight and night, heavy or flowing traffic etc.). All data-driven models presented here performed worse than the baseline of human level performance, but mainly because of the *interurban* category. For this category, the test data set contains

a significant amount of samples, which are unlike any samples present in the training data such that all models, unlike our human subjects, are unable to make sense out of them. Therefore, we evaluated the models' performance on a subset of the test set removing this chunk of data boosting the classification accuracy of our model. Thus, we conclude that on the one hand, our data set is too small and too limited for our model to generalize sufficiently from it. On the other hand, we assume that a larger and more balanced data set will improve the performance of all models (including the CNN). However, there are also general issues regarding the process of encoding the current driving scene into a vector representation. For instance, while we encode many traffic signs (243 of the total 258 atomic entities), the great majority of them is not that useful for driving context classification, as they occur rather rarely. Although our implementation of a traffic sign memory partially absorbs this effect, there is still a significant amount of traffic signs, which are not memorized and furthermore are not that informative regarding the current driving context. Considering this fact, we are left with a very small vocabulary since only 5 classes of traffic participants are encoded. Therefore, we might want to include more categories such as the number of lanes or road markings. Another method could be to try to encode at a higher level directly, for instance, entities with motion properties such as a vector for a pedestrian that will cross the road or for a *fast car driving towards us*.

Regarding the impact of structured vocabularies on the model's classification accuracy, we achieved a partial success. We showed that a visual-semantic structure can be successfully encoded and, in principle, be learned given suitable data. We were able to show that different vocabularies have a measurable effect on the resulting classification accuracy. However, we could not find a guiding principle to generally improve results such as using visual-semantic vocabularies improves classification accuracy over all random vocabularies by a certain margin. The reasons for these results are manifold: While we do argue that knowledge about visual similarity might improve scene classification, this kind of information alone is rather limited especially with the current implementation. That is especially true for traffic participants, since encoding their visual similarity simply does not add beneficial information to better distinguish between driving contexts. Trying to expand with semantic content, we are relatively successful with traffic signs, as their meaning is clear, explicit and unchanging. However, semantics of traffic participants, even if we succeed in encoding a structure, are again not that helpful regarding the task of driving context classification as performed here. Although the combined visual-semantic vocabulary is successful in encapsulating the underlying information, this information is too little to improve performance significantly with the given tasks. However, other types of semantic information can be conceived but would require a more complex algorithm as well as a large data set to be learned automatically.

Furthermore, we only encode structure for a part of the original vocabularies and leave vectors representing entities, for which we have no suitable structured data available, unchanged at their random initialization. The measured impact on classification performance might be larger if all entities were part of the structured representation. On the other hand, there is no data set available to learn a semantic embedding of traffic signs. Therefore, the semantics had to be manually designed. Due to these data limitations, we do not succeed in establishing a comprehensive learning algorithm. Learning would make the approach more independent from human design choices and biases, which would not only enable generalization, but also allow the algorithm to learn the kind of structure it "needs" rather than us as humans imposing our understanding on it.

6 Instantiating a cognitive model for predicting vehicle behavior

Predicting future behavior and positions of other traffic participants from observations is a key problem, that needs to be solved by human drivers and automated vehicles alike to safely navigate their environment and to reach their desired goal. Therefore, we picked behavior prediction as another task for investigating the potential of vector representations in automotive context. In contrast to the task of classifying the current driving context presented in chapter 5, predicting future positions of vehicles is a regression problem, i.e., we are predicting continuous values such as spatial positions instead of discrete class labels. However, future positions of vehicles not only depend on each vehicle's own past positions and dynamic data such as velocity and acceleration but also on the behavior of the other traffic participants in the vehicle's surroundings. For instance, in a situation as shown in Fig. 6.1, the behavior of the target vehicle, as depicted by a solid blue and dotted orange line for past and future positions respectively, is influenced by the surrounding vehicles, as illustrated by solid and dotted gray lines for past and future positions respectively. The target vehicle is approached from behind by a faster vehicle causing it to eventually change its lane to the right, which, for now, is still occupied by the ego-vehicle and another vehicle. All of these external influences have an impact on the target vehicle's motion (and vice versa). We hypothesize that structured vector representations will be able to capture these relations and mutual influence between traffic participants, which is necessary for reliable predictions. As we aim for a model-free data representation, we avoid introducing any physical constraints or restrictions regarding our data-representation or the predicting models. Although this allows for physically unrealistic or even impossible trajectory predictions, we want our neural network models to figure out the best predictions on their own without biasing them in any direction. In this section, we present a representation of spatial information for multiple objects in semantic vectors of fixed length using the convolutive power introduced in Definition 3.19. In contrast to other representations of spatial data, the dimensionality of our vectors is independent of the number of encoded entities. We use this representation as input for simple feed-forward neural networks and more sophisticated LSTM-based models and compare them against each other and a linear prediction model as the simplest baseline. We conduct a thorough and detailed analysis and evaluate our models on two different data sets, namely one proprietary data set recorded with an automated vehicle prototype and one publicly available trajectory data set.

We analyze our models with respect to the context, i.e., which model performs best depending on the current driving situation. Furthermore, we investigate the influence of the composition of the training data on the models' performance. Additionally, we show that by using our vector representation with a simple network architecture we can achieve results comparable to more complex models.

6.1 Data and preprocessing

In this chapter, we use two different data sets for training and evaluation of our behavior prediction models, which we describe in more detail in subsequent sections. We refer to those data sets as *Onboard* or D_1 (see section 6.1.1), which is our main data set, and *NGSIM* or D_2 (see section 6.1.2), which is a publicly available data set used for reference and comparability. In this section, we describe both data sets regarding available features, available sources of information as well as their key differences and the preprocessing procedure.

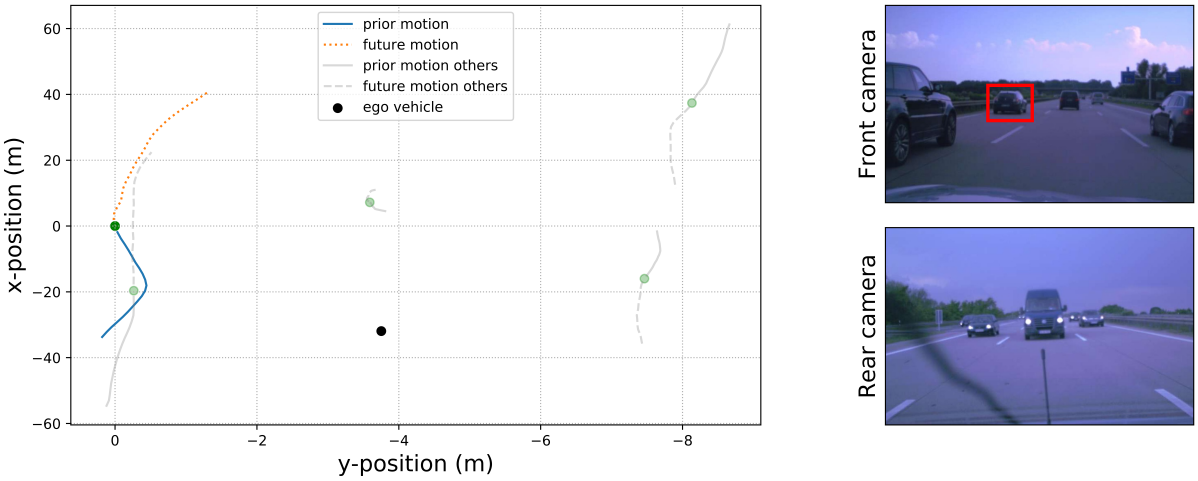


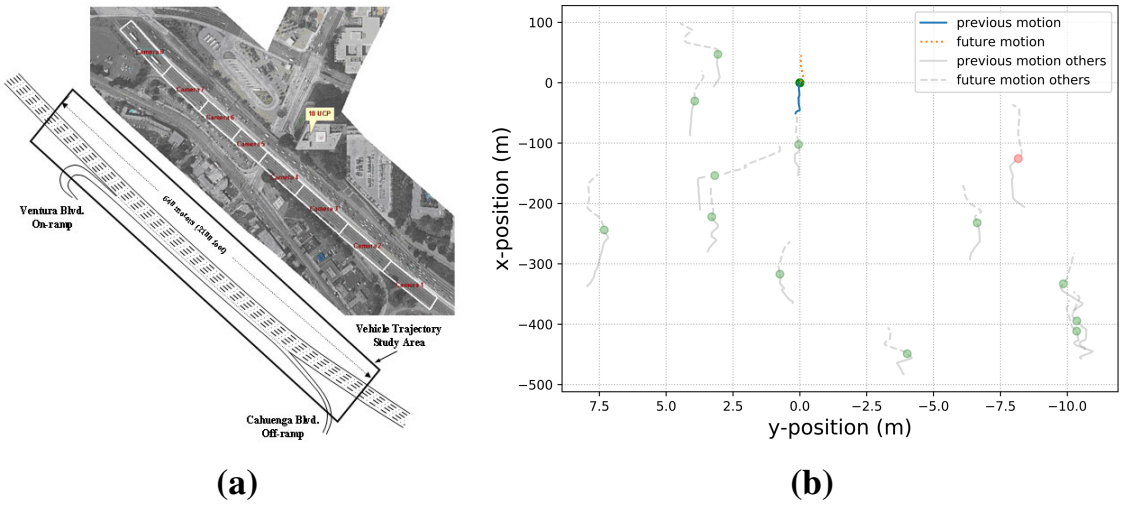
Figure 6.1: Data visualization of one driving situation example from the *On-board* data set D_1. The dots in the left plot indicate the position of the vehicles and color-code the vehicle type (red=motorcycle, green=car, blue=truck, black=ego-vehicle), blue and orange lines show past and future motion of the target vehicle whereas gray lines depict the other vehicles' motion. The right figures show raw images of the ego-vehicle's front and rear camera with the target vehicle highlighted by a red rectangle.

6.1.1 On-board-sensors data set

This is our main data set used in this chapter. It contains real-world data gathered using the on-board sensors of an automated vehicle prototype, which we refer to as the *ego-vehicle*, during test drives mainly on highways in the area of Munich, Germany. Therefore, we refer to this data set as the *On-board* data set. The data set contains object-lists with a variety of features obtained by fusing different sensor sources. Apart from features about motion and behavior of the dynamic objects in the scene such as position, velocity and acceleration, which are estimated from LIDAR sensors, there is also visual information like object type probabilities or lane information, which is acquired from additional camera sensors (see Aeberhard et al., 2015, for further information on the sensor setup). Table 6.1 gives an overview and detailed description of the data features available in this data set, which are relevant for our models.

Data label	**Description**
Position	Lateral/Longitudinal position absolute/relative to the ego-vehicle's position estimated from range sensor readings
Velocity	Lateral/Longitudinal velocity absolute/relative to the ego-vehicle's velocity estimated from range sensor readings
Acceleration	Lateral/Lateral acceleration absolute/relative to the ego-vehicle's velocity estimated from range sensor readings
Lane	Information about the lane relative to the ego-vehicle estimated from fused sensor reading (camera and range sensors)
LaneBorderDistance	Distance to left/right border of the current lane estimated from fused sensor reading (camera and range sensors)
LaneCurvature	Information about the lane curvature estimated from fused sensor reading (camera and range sensors)
TypeProbability	Probability for the object being a of certain type (e.g., car or truck) estimated from camera sensors

Table 6.1: Table depicting different features for dynamic objects within the training data



(a) **(b)**

Figure 6.2: Visualization of *NGSIM* data set: (a) depicts the highway segment from top view perspective indicating the camera's position. Image source: Colyar and Halkias (2018). (b) visualizes the data of one particular driving situation from the data set.

In contrast to the data set used in chapter 5, the object-lists of the data set used here contain already preprocessed information as a fusion from the different available sensor sources. This fused information about objects is available at a frequency of roughly 5 Hz. The main feature of this data set is that all information (position, velocity, etc.) about vehicles other than the ego-vehicle is measured with respect to that ego-vehicle and its coordinate system. Figure 6.1 shows one example situation from this data set: the left plot depicts the already preprocessed) positional information of all vehicles detected by the ego-vehicle's on-board sensors. The dots indicate the current position of the vehicles and color-code the vehicle type (red=motorcycle, green=car, blue=truck, black=ego-vehicle). The blue and orange lines illustrate 5 s of past and future motion of the target vehicle respectively. Solid and dashed gray lines depict the other vehicles' past and future motion respectively. On the right-hand side, the raw images from the front and rear camera give an impression of the driving situation at hand with the target vehicle highlighted by a red box. In total, the *On-board* data set contains 3891 vehicles, which yield a total length of roughly 28.3 h when adding up the time each individual vehicle is visible.

6.1.2 NGSIM US-101 data set

The NGSIM US-101 data set (Colyar and Halkias, 2018) is a publicly available data set recorded on a segment of approximately 640 m length with 6 lanes on the US-101 freeway in Los Angeles, California. Although the data set was originally intended for driver behavior analysis and traffic flow models (He, 2017), it has also been used to train trajectory prediction models, for instance proposed by Altche and La Fortelle (2017) and Deo and Trivedi (2018b). The data set was recorded using cameras observing freeway traffic from rooftops with trajectory-data being extracted later from the obtained video footage. Figure 6.2 shows the recorded highway segment from top view perspective indicating the camera's position (Fig. 6.2a) as well as the visualization of one example driving situation (Fig. 6.2b). The data set contains a total of 45 min of driving data split into three 15 min segments of mild, moderate and congesting traffic conditions. Apart from positional information in lateral and longitudinal direction (in a global and local coordinate system), additional features such as instantaneous velocity, acceleration, vehicle size as well as the current lane are available for each vehicle. The trajectory data is sampled with a frequency of 10 Hz. The main difference to the *On-board* data set is the fact that the *NGSIM* data set is recorded with external stationary cameras instead of a driving vehicle's on-board sensors. Thus, there is no ego-vehicle present in the data and all information is available in absolute coordinates instead of being

measured relative to one particular ego-vehicle. In total, the *NGSIM* data set contains 5930 vehicles and therefore a total time of roughly 91.3 h when adding up the time each individual vehicle is visible.

6.1.3 Preprocessing

In this section, we describe the preprocessing steps performed before training our models to prepare the trajectory information contained in our two data sets as input for neural networks. Although we aim to keep these preprocessing steps as consistent as possible across both data sets, there are some mild differences, which we will point out here. We aim to anticipate positions of dynamic objects 5 s into the future based on past positions 5 s prior to their current location for one object at a time. As the two data sets are sampled at different frequencies, we interpolate the available data over 20 equidistant steps to achieve intervals of 0.25 s to improve consistency and comparability. Furthermore, we translate the current position of the target vehicle, i.e., the vehicle to be predicted, to be the origin of the reference coordinate system. That is, the current position of the target vehicle will be at position $(0,0)$ for all data samples consistently across both data sets (see Fig. 6.1 and 6.2b). The reason for this design choice is two-fold: on the one hand, we make samples from both data sets, which originally have different coordinate frames (global vs. ego-vehicle) as reference, more comparable and consistent. On the other hand, by moving the current position of the target vehicle to the origin of the reference coordinate system of the sample, we prevent our models from treating similar trajectories differently due to positional variations. Finally, to improve suitability of the data as input for neural networks, we divide all x-positions by a factor of 10 such that x-/y-values are scaled to a similar order of magnitude. Since the *NGSIM* data set D_2 is sampled at a high frequency of 10 Hz and contains more data than the *On-board* data set, we use only every 10th data point. Thereby, we avoid the creation of too many overlapping, and therefore too similar, data samples for the *NGSIM* data set. Furthermore, we swapped the dimensions of the positions in the *NGSIM* data set such that for both data sets x- and y-direction correspond to longitudinal and lateral positions respectively. For training and evaluating our models, we split both data sets into two distincts subset containing training $T_i \subset D_i$ and validation data $V_i \subset D_i$. Those training and validation subsets contain 90 % and 10 % of the objects within the data sets respectively with $T_i \cap V_i = \varnothing$ to avoid testing our models on vehicles they have been trained with.

6.1.4 Data set peculiarities

In this section, we analyze the composition of our data sets regarding the amount of "interesting" behavior of the target vehicle. Both, the *On-board* and *NGSIM* data set consist of mainly highway driving, where we would expect mainly straight driving with the most interesting situations being the target vehicle, i.e., the vehicle whose motion we aim to predict, performing a lane change. Hence, we are interested in the amount of situations where the target vehicle actually performs a lane change and how much more prominent normal straight driving is in both our data sets. For the *On-board* data set, we have information about the current lane as well as the distance to the lane borders estimated from the ego-vehicle's cameras available for all vehicles. The *NGSIM* data set contains information about the current lane for each vehicle extracted from the external camera's video footage. Thus, the selection process for the examples containing a lane change is straightforward for both data sets. Figure 6.3 shows one data sample from the *On-board* data set containing a lane change performed by the target vehicle in its future motion to be predicted. Comparing this example to the one shown in Fig. 6.1, which shows mainly straight driving for all vehicles present in the scene, we observe that a lane change mainly influences the vehicle's motion in lateral (y) direction.

Figure 6.4 shows the amount of situations where the target vehicle performs a lane change in comparison to the amount of situations where no such behavior occurs for both, the *On-board* and *NGSIM* data set. For the *On-board* data set, in 86.1 % of all data samples the target vehicle does not perform a lane change, i.e., only 13.8 % of all data samples contain a lane change performed by the target vehicle. We further distinguish between lane changes performed during the trajectory history, i.e., the past 5 s before

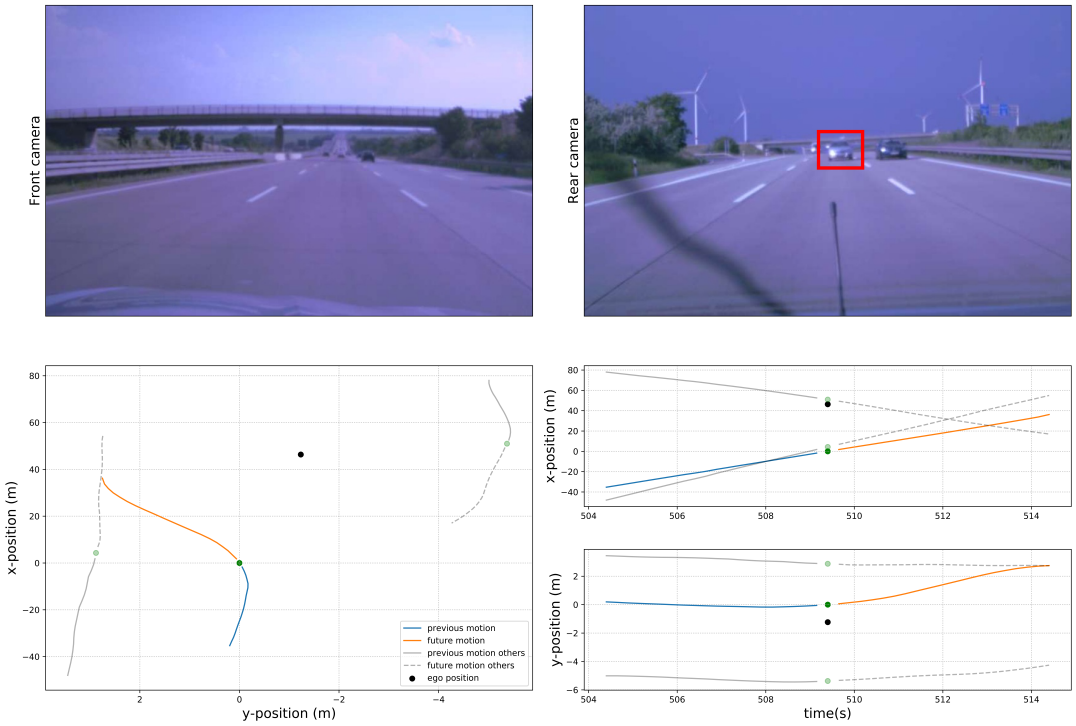


Figure 6.3: Data visualization of one data sample from the *On-board* data set D_1 containing a future lane change of the target vehicle. The dots in the left plot indicate the position of the vehicles and color-code the vehicle type (red=motorcycle, green=car, blue=truck, black=ego-vehicle), blue and orange lines show past and future motion of the target vehicle whereas gray lines depict the other vehicles' motion. The images in the top row show raw images recorded using the ego-vehicle's front and rear camera with the target vehicle highlighted by a red bounding box.

the current time step (labeled as *past* in Fig. 6.4) and lane changes that are performed in the future, i.e., during the future 5 s from the current time step (labeled as *future* in Fig. 6.4). We consider the lane changes in the future part of data samples to be the most interesting and challenging ones, since any model making predictions about the future trajectory needs to be able to anticipate these lane changes. For the *On-board* data set, 7 % of all data samples contain a lane change in the trajectory history, while 8.2 % of the samples contain a future lane change performed by the target vehicle. In comparison to the 86.1 % of data samples not containing a lane change, the amount of samples with interesting behavior other than straight driving within the data set is significantly less present. For the *NGSIM* data set, the discrepancy between the amount of samples without the target vehicle performing a lane change and the number of samples containing a lane change is even more significant. The percentage of samples without a target vehicle lane change is 95.1 % while only 4.9 % of the samples contain a lane change performed by the target vehicle at all. The amount of samples containing a future lane change performed by the target vehicle is only 2.6 % of all samples in the *NGSIM* data set. Hence, there is a significant imbalance in both data sets between examples containing mainly straight driving by the target vehicle, namely 86.1 % and 95.1 % of all samples in the *On-board* and *NGSIM* data set respectively, where most likely already simple prediction approaches are able to achieve reasonable results.

6.1.5 Performance baselines

In this section, we present and discuss the models we aim to use as performance baselines for our SPA-based trajectory prediction approaches. Therefore, we begin with an example: Figure 6.5 shows an overtaking maneuver in a highway situation from the *On-board* data set at four different selected time

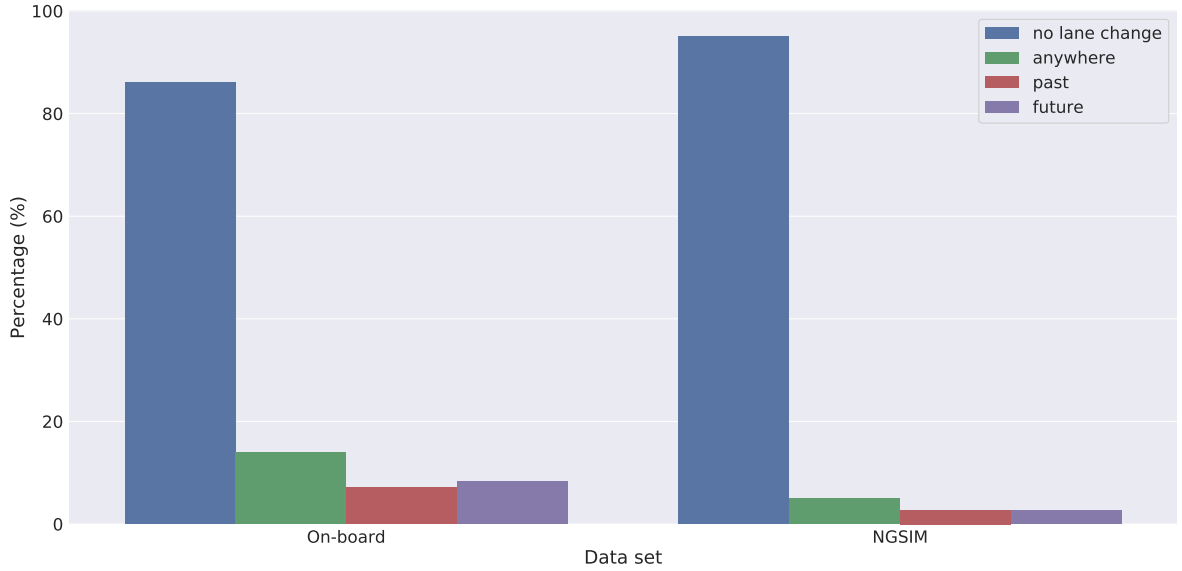
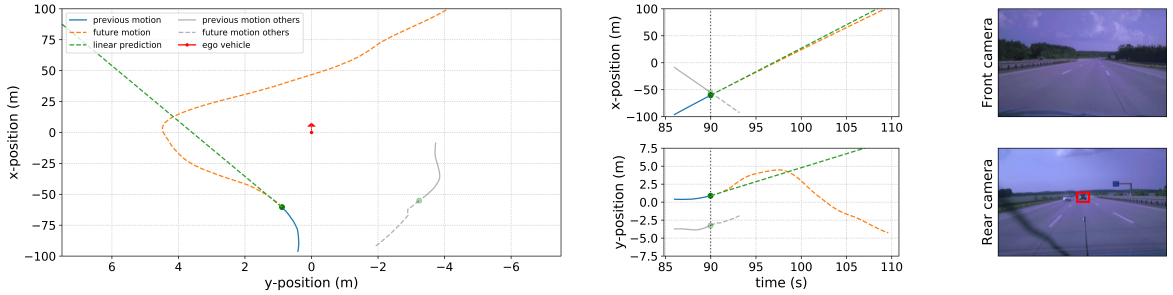


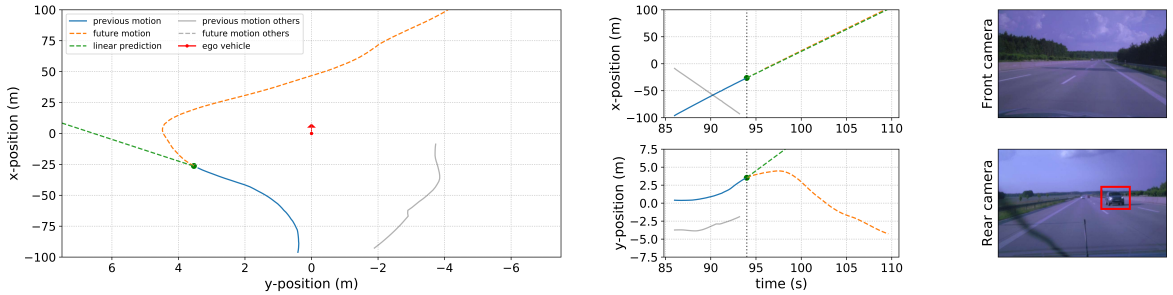
Figure 6.4: Visualization of the composition of both data sets regarding lane changes of the target vehicle.

steps. The ego-vehicle is overtaken by another car, the target vehicle to be predicted, approaching from behind. During that overtaking maneuver, the ego-vehicle itself performs a lane change from the middle to the most right of the three lanes. Figures 6.5a– 6.5d show different times of the situation. Solid lines indicate previous positions whereas dashed lines indicate future positions or predictions. The solid blue line depicts the past motion of the target vehicle overtaking the ego-vehicle, while solid gray lines visualize the past positions of other vehicles in the scene. The dashed green line illustrates predictions from a simple linear model based on a constant velocity assumption. This example illustrates the general trend we observe for both data sets used in this chapter that already simple linear prediction models achieve solid prediction accuracy, especially in x-direction. This makes sense as both data sets almost exclusively contain highway driving situations, which in turn consists of significantly more straight driving and rather rare lane-change maneuvers as analyzed in section 6.1.4. For straight driving, linear prediction based on a constant velocity assumption is already a solid prediction approach, especially if all dynamic information (position, velocity etc.) is given relative to an already moving ego-vehicle like with the *On-board* data set D_1. Hence, we use these linear prediction models based on a constant velocity assumption as the simplest baseline models to compare our neural models using our distributed vector representations as input to.

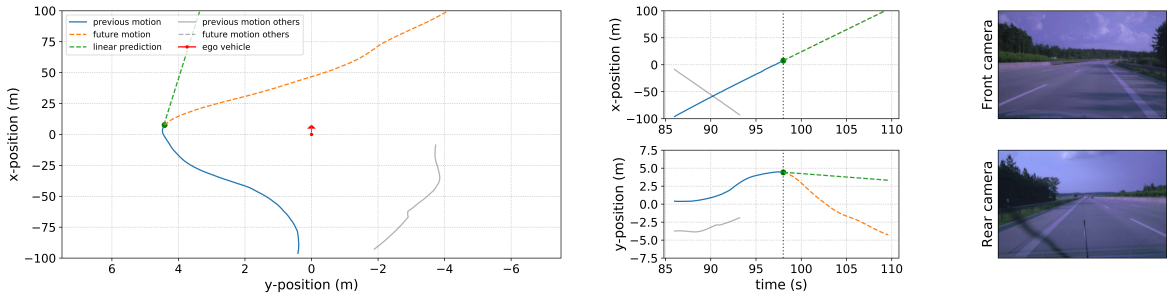
The analysis of related research on trajectory prediction in automotive context given in section 2.3.5 suggests, that the most successful current state-of-the-art approaches rely mainly on LSTM-based neural network architectures. They are typically combined with other neural networks such as convolutional layers or classification networks (Deo and Trivedi, 2018a) to improve model performance. In this chapter, we use LSTM-based models as well as simpler feed-forward neural networks to predict trajectories based on our semantic vector representation of the current driving situation. Hence, we use the same models just employing a different encoding (see section 6.2.1 for further details on the reference encoding schemes) of the input data avoiding further complexity of additional networks or layers to make the models as comparable as possible.



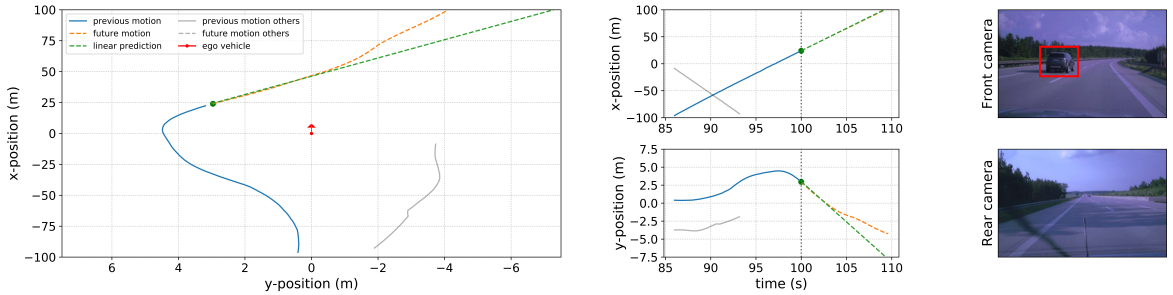
(a) Overtaking maneuvre at $t = 90\,\mathrm{s}$



(b) Overtaking maneuvre at $t = 94\,\mathrm{s}$



(c) Overtaking maneuvre at $t = 98\,\mathrm{s}$



(d) Overtaking maneuvre at $t = 100\,\mathrm{s}$

Figure 6.5: An example scene visualizing the data of an overtaking maneuver in a highway situation at selected time steps.

6.2 Representation and models

In this section, we describe the models we use for the behavior prediction task. The input data for our models are sequences of positional data either as raw numerical values or in the form of semantic vectors as described in section 6.2.1. LSTM-based neural network architectures have proven to be powerful tools for sequential data analysis and are widely used for behavior, or more generally, motion prediction. We also investigate much simpler feed-forward neural networks constructed using the principles of the NEF (Neural Engineering Framework) (c.f. section 3.3) to evaluate the performance gains achieved by

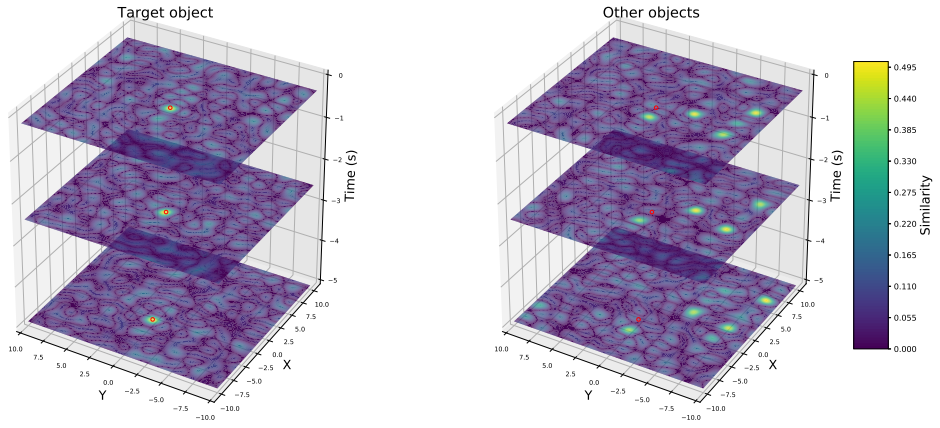


Figure 6.6: Visualization of the convolutive vector-power representation of one particular driving situation over time at selected time-steps as a heat map of similarity values for 512-dimensional vectors. The red circles indicate the measured position of the target vehicle.

the more complex LSTM models. To encode spatial information of driving situations in sequences of semantic vectors of fixed length, we employ the convolutive vector power introduced in Definition 3.19 and analyzed in sections 4.2.1 and 4.2.3. For reference, we also describe a simpler vector representation employing the scalar multiplication encoding of numerical values also shown in section 4.2.1 as well as a raw numerical representation encoding only the positional information of the target vehicle. Furthermore, we describe the architecture of the learning models used for behavior prediction from that input data. Importantly, here we use our models to predict one particular target vehicle at a time instead of trying to jointly predict the progress of the entire scene. To achieve a forecast of the development of all vehicles in the scene, we would deploy several instantiations of the same network. Using this approach, we only have to train one model while we can use each detected vehicle as training example, which significantly increases the amount of training data.

6.2.1 Scene representation in vectors

We use three different encoding schemes of the positional input data in this chapter. Our main encoding scheme is the semantic vector representation as depicted in the following section making use of the convolutive vector power to encode numerical values (see also section 4.2.1). We also apply two other encoding schemes using the scalar multiplication encoding of numerical values in vectors (see also 4.2.1 as well as simply using the raw numerical values of the input data.

Convolutive power representation

In this section, we investigate the expressive power of encoding the spatial positions of multiple vehicles using the convolutive vector-power introduced in Definition 3.19. Given the results of section 5.3.4 that the impact of similarity structures in rather small vocabularies is neglectable, we create a random vocabulary V of atomic vectors here. We assign a random real-valued vector from the unit sphere to each category of dynamic objects (e.g., car, motorcycle, truck) as well as random unitary vectors (c.f. Definition 3.18) $\mathbf{X}$ and $\mathbf{Y}$ to encode the units of spatial positions in vectors. We use unitary vectors $\mathbf{X}$ and $\mathbf{Y}$ since they have unit length and are closed under convolutive exponentiation as shown in Lemma 3.20. Therefore, by encoding spatial positions with powers of unitary vectors, we avoid exploding lengths of

our final scene vectors, which would lead to additional noise and unwanted behavior when using them as input for neural networks. Furthermore, we use additional random ID-vectors **TARGET** and **EGO** representing the target object to be predicted and, if applicable, the ego-vehicle. Given a situation as shown in Fig. 6.1 with a sequence of prior positions (x_t, y_t) for the target vehicle at time step $t \in \{t_0, \ldots, t_N\}$ and equivalent sequences $(x_{obj,t}, y_{obj,t})$ for other traffic participants, we encapsulate this information in a scene vector

$$\mathbf{S}_t = \underbrace{\mathbf{TARGET} \circledast \mathbf{TYPE}_{target} \circledast \mathbf{X}^{x_t} \circledast \mathbf{Y}^{y_t}}_{\text{target-vehicle}} \oplus \underbrace{\sum_{obj} \mathbf{TYPE}_{obj} \circledast \mathbf{X}^{x_{obj,t}} \circledast \mathbf{Y}^{y_{obj,t}}}_{\text{other objects}}, \tag{6.1}$$

This yields a sequence of semantic scene vectors $\mathbf{S}_t$ for $t \in \{t_0, \ldots, t_N\}$ encoding the past spatial development of objects in the current driving situation. An alternative option could be to simply sum up the vectors at each past time step to encode the complete motion history within a single vector. However, given the results regarding the capacity of the convolutive power encoding shown in section 4.2.3, we decided for a sequence of individual vectors instead, which is also a more suitable input to neural networks employing LSTM units. Figure 6.6 depicts the aforementioned scene vector representation: the left plots show similarities (depicted as heat map) between the vector $\mathbf{S}_t$ encoding the scene from Fig. 6.1 and the vectors $\mathbf{v}_i = \mathbf{TARGET} \circledast \mathbf{TYPE}_{target} \circledast \mathbf{X}^{\bar{x}_i} \circledast \mathbf{Y}^{\bar{y}_i}$ for a sequence of discrete position samples $\bar{x}_i, \bar{y}_i$. Similarly, the right plots show similarities between $\mathbf{S}_t$ and $\mathbf{CAR} \circledast \mathbf{X}^{\bar{x}_i} \circledast \mathbf{Y}^{\bar{y}_i}$ visualizing all other objects in the scene of type *car*. Importantly, each vector in the sequence $\mathbf{S}_t$, i.e., each plane in the sequence of heat maps shown in Fig. 6.6 is a spatial encoding vector of the type depicted Fig. 4.11 Hence, we can encode spatial information of several different objects in a sequence of semantic vectors and reliably decode it back out. This allows us to encode automotive scenes with varying number of dynamic objects in a vector representation of fixed dimension. Note that by using this vector representation as input data for a neural network (or any other predictor), we predict the future position of one other traffic participant at a time. The indication vector **TARGET** bound to the target vehicle, i.e., the object we want the model to predict, indicates the network the current focus. To predict all objects present in a scene during deployment, multiple instantiations of the same network can be used. Thereby, the amount of training data generated per file increases with the number of objects while we only need to train one network.
To avoid accumulation of noise in the vectors (cf. section 4.2.3) while focusing on the vehicles most relevant for prediction, we only use objects closer than 40 m to the target vehicle in the *On-board* data set. For the *NGSIM* data set D_2, we additionally include only objects on the same lane as the target vehicle and on adjacent lanes. Thereby, we aim for consistency across both data sets and we keep the input data as comparable as possible to what a driving vehicle could be able to detect using its on-board sensors.
For the *On-board* data set D_1, we use two different variants of this representation, which differ in that the ego-vehicle's position is used or excluded in the *other objects* part of Equation (6.1), yielding two sequences $(\mathbf{S}_t^{ego})_{t_0}^{t_N}$ and $(\mathbf{S}_t)_{t_0}^{t_N}$. We used Nengo's SPA package for implementation and therefore refer to these two encoding schemes $(\mathbf{S}_t)_{t_0}^{t_N}$ and $(\mathbf{S}_t^{ego})_{t_0}^{t_N}$ as *SPA-power* and *SPA-power-with-ego* respectively. As the *NGSIM* data set D_2 does not contain an ego-vehicle, we only investigate the *SPA-power* encoding scheme there.

Reference encoding schemes

For a simple reference vector-representation, we employ the scalar multiplication encoding for numerical values shown in section 4.2.1. Therefore, we add the positional vectors $\mathbf{X}$ and $\mathbf{Y}$ scaled with the target vehicle's prior positions (x_t, y_t) at each time step t, yielding the sequence $\tilde{\mathbf{S}}_t = x_t \cdot \mathbf{X} + y_t \cdot \mathbf{Y}$. We refer to this simpler encoding scheme based on the scalar multiplication encoding as *SPA-simple*. Finally, we also use plain numerical position values $p_t = (x_t, y_t)$ as another reference encoding scheme of the input data to our learning models, which we refer to as *numerical*. Importantly, only the *SPA-power*

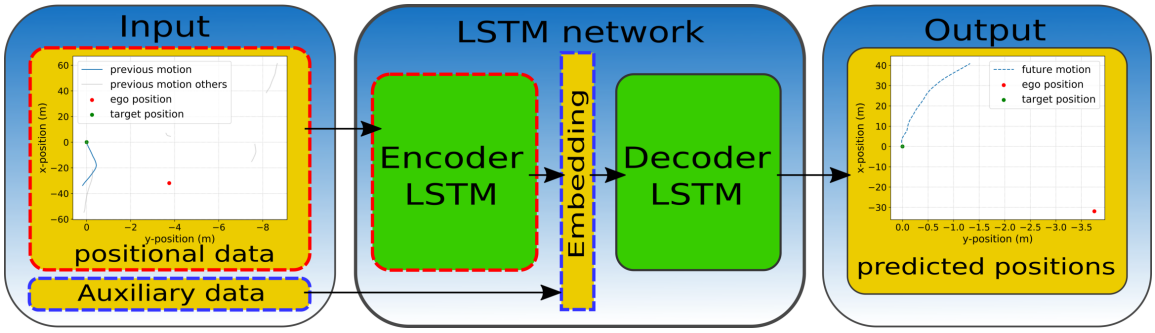


Figure 6.7: Visualization of our LSTM-based learning architecture. Modules that change with varying encoding scheme of the input data are highlighted through dashed red borders whereas parts that change when varying the data set are highlighted through dashed blue borders.

representation variants $(\mathbf{S}_t)_{t_0}^{t_N}$ and $(\mathbf{S}_t^{ego})_{t_0}^{t_N}$ contain positional information about vehicles other than the target.

6.2.2 LSTM-based prediction models

In this section, we use a LSTM (Long Short-Term Memory) (Hochreiter and Schmidhuber, 1997) network-architecture for the prediction of vehicle positions. Our network consists of one LSTM encoder and decoder cell for sequence to sequence prediction, which means that the input and the final result of our model is sequential data. The encoder LSTM takes positional data for 20 past, equidistant time frames as input. That is, the input data is a sequence of 20 items of either positions of the target vehicle or a sequence of high-dimensional vectors encoding this positional data. The resulting embedding vector encodes the history of the input data over those time frames. This embedding vector is concatenated with additional auxiliary information to aid the model when predicting the future trajectory of the target vehicle. This auxiliary data is information, that is available to the system when the prediction is to happen, i.e., sensory data available at prediction time or future data about the ego-vehicle such as its own planned trajectory (see section 6.3.1 for further details on this auxiliary data). Finally, the embedding vector is used as input for the decoder LSTM to predict future vehicle positions. The output of each model is a sequence of 20 positions of the target vehicle predicted over a certain temporal horizon into the future. We use the same network architecture for all encoding schemes of the input data and for both data sets. However, the dimensionality of the input and the information used as auxiliary information to enrich the embedding vector vary over different encoding schemes and data sets respectively. Figure 6.7 visualizes the architecture of our LSTM models indicating modules that change when varying the encoding scheme by a dashed red border whereas parts that change with the data set are highlighted through a dashed blue border.

6.2.3 Simple feed-forward NEF-based prediction models

As an alternative to the LSTM-models, we also considered a much simpler single-hidden-layer network defined using the NEF (Neural Engineering Framework) (Eliasmith and C. H. Anderson, 2003). While this is usually used for constructing large-scale biologically realistic neuron models (Eliasmith et al., 2012), the NEF software toolkit Nengo (Bekolay et al., 2014) also allows for traditional feed-forward artificial neural networks using either spiking or non-spiking neurons. For these NEF networks, we use a single hidden layer, with randomly generated (and fixed) input weights, and use least-squares optimization to compute the output weights. We employ the principles of the NEF as shown in section 3.3 to instantiate and train these models. As with any traditional network, we can have any number of input, output, and hidden neurons, all following this same process. The goal here is to provide a simple baseline

for comparison to the LSTM networks, to see what (if any) performance gain is produced by the more complex network approach. However, these simpler networks are unable to process sequential data in the same way as the LSTM models. Therefore, we will have to slightly adapt our data, especially the semantic vector sequences, to make it suitable as input for the feed-forward networks.

6.2.4 Excursion on unsupervised anomaly detection

In this section, we take a brief detour on anomaly detection. We are further interested in the information encapsulated within our semantic vector representation and if it can be used to detect potentially dangerous driving situations from just the vector representation. VSAs in general already have an intrinsic mechanism of comparing vectors with one another through the measure of similarity ϕ. However, it is not clear if simply comparing vectors in terms of similarity to, for instance, the mean pairwise-similarity of all known vectors, or a subset of vectors considered "normal", will differentiate outliers from the "normal data". A-priori, it might not even be clear what vectors belong to the baseline set of normal data or how to define vectors to be considered as inliers. One option could be to manually define metrics such as the number of vehicles in the scene or a threshold for the distance between the vehicles to detect crowded and potentially dangerous situations. However, such an approach suffers from the typical issues of manual engineering such as biases introduced by the human designer as well as poor scaling. Therefore, we employ an unsupervised learning approach based on fully-connected autoencoder neural networks similar to the one proposed in J. Chen et al. (2017). We train an autoencoder neural network on the latest vector in the sequence $(\mathbf{S}_t)_{t_0}^{t_N}$ of our scene vectors based on the convolutive vector-power representation in unsupervised fashion. Thereby, the network is trained to reconstruct, i.e., generate replicates of, the data it is given. Once the network is trained on a sufficiently large data set, we can calculate the element-wise error between the original vector $\mathbf{v} = (v_0, \ldots, v_{D-1})$ and the replicate vector $\tilde{\mathbf{v}} = (\tilde{v}_0, \ldots, \tilde{v}_{D-1})$ generated by the neural network autoencoder, i.e.,

$$\varepsilon_{\mathbf{v}} = \sqrt{\frac{1}{D} \sum_{i=0}^{D-1} (v_i - \tilde{v}_i)^2}. \tag{6.2}$$

Vectors exceeding a certain threshold c for this reconstruction error, i.e., $\varepsilon_{\mathbf{v}} > c$ will be considered as outliers or anomalies. The threshold c is generated from the percentage of examples we expect to be anomalies within the data set, which is typically chosen in the range of 5 % to 15 %.

6.3 Experiments and results

In this section, we describe the training process and parameters of all our models and give a detailed analysis and evaluation of the results achieved. The LSTM models are implemented in Tensorflow (Abadi et al., 2016) whereas the NEF models are implemented using the Nengo software suite (Bekolay et al., 2014). We use the RMSE as our main metric for evaluation purposes. In contrast to earlier work, we inspect the RMSE for lateral and longitudinal directions separately to give more detailed insights into the models' behavior. Calculating the RMSE of the Euclidean distance would absorb the influence of the lateral RMSE since it is an order of magnitude smaller than the longitudinal RMSE, while we consider both directions to be at least equally important. The lateral RMSE is even more informative regarding the models' performance on, for instance, lane change maneuvers. For all evaluations in this section, we refer to the longitudinal and lateral direction as x- and y-direction respectively. Furthermore, we investigate where the models show their best performance looking for correlations between prediction accuracy and specific driving situations.
For both model types, we follow the same order of analyzing steps: firstly, we perform an investigation of the model's hyperparameters to find the best possible configuration for each model. Secondly, we describe the process of training each model with all peculiarities corresponding to the data used or the training process itself. Finally, we evaluate the trained models and compare their performance. For the

hyperparameter analysis, we conduct a thorough investigation on both our network architectures using only numerical input data for simplicity and to keep the time needed for training limited. Systematically, we only analyze one parameter at a time and fix the best value for that parameter for the subsequent analysis of other parameters. However, we also inspect certain parameter pairs jointly if there are correlations or mutual influences between the parameters to be expected. All hyperparameter analyses in this section on both model types are performed on the *On-board* data set $D1$. If not declared otherwise, all figures show the performance of the investigated models on the validation part $V_1 \subset D_1$ of the *On-board* data set. The training process however is intended to be kept as coherent as possible between the data sets and differences having an impact on the training process will be highlighted where necessary. Finally, we evaluate both model types' performance on both available data sets and, especially for the LSTM-based models, we give a thorough analysis on which model performs best depending on the current driving context. Thereby, we will identify strengths and weaknesses of each particular model.

Short name	Input	Position encoding	Network architecture	Training	Number of Units/Neurons	Data set
linear	current position and velocity	-	Linear regression	-	-	both
LSTM numerical	sequence of positions	-	LSTM with one encoder/decoder cell each	Offline, backpropagation	150 units per cell	both
LSTM SPA 1	semantic vector sequence	convolutive power	LSTM with one encoder/decoder cell each	Offline, backpropagation	150 units per cell	both
LSTM SPA 2	semantic vector sequence	scalar multiplication	LSTM with one encoder/decoder cell each	Offline, backpropagation	150 units per cell	both
LSTM SPA 3	semantic vector sequence	convolutive power incl. ego-vehicle	LSTM with one encoder/decoder cell each	Offline, backpropagation	150 units per cell	*On-board*
NEF numerical	sequence of positions	-	NEF Single-layer	Offline, least-squares	3000 neurons	both
NEF SPA 1	semantic vector sum	convolutive power incl. ego-vehicle	NEF Single-layer	Offline, least-squares	3000 neurons	*On-board*
NEF SPA 2	semantic vector sum	convolutive power	NEF Single-layer	Offline, least-squares	3000 neurons	*NGSIM*

Table 6.2: Summary of the evaluated models regarding architecture, input data, encoding and training.

Table 6.2 summarizes the models evaluated in this section. The models LSTM SPA 1 - 3 as well as LSTM numerical employ the same network architecture as described in section 6.2.2 with sequential information as input data (using the different encoding schemes presented in section 6.2.1) and are analyzed in section 6.3.1. The models NEF SPA 1 and 2 employ the simpler, single-layer, feed-forward architecture as described in section 6.2.3 with a vector obtained as partial sum of vectors from the whole sequence used as input for the LSTM models (see section 6.3.2 for further details). The models will be denoted in legends of the figures in this chapter by their short name given in table 6.2.
In section 6.2.1, we have described the different encoding schemes we will use to evaluate our models. We mentioned that the models employing the convolutive power to encode the input data (i.e., LSTM SPA 1, 3 and NEF SPA 1 and 2) are the only ones having access to information about objects other than the target vehicle. Although these models therefore have access to more data than the other reference models such as LSTM numerical, we are interested in evaluating the benefits of encoding the interconnections between vehicles implicitly in the input data using our semantic vector encoding instead of introducing a more complex network architecture. Therefore, we focus on the same network architecture for all encoding schemes in this chapter and leave a comparison with more sophisticated network architectures,

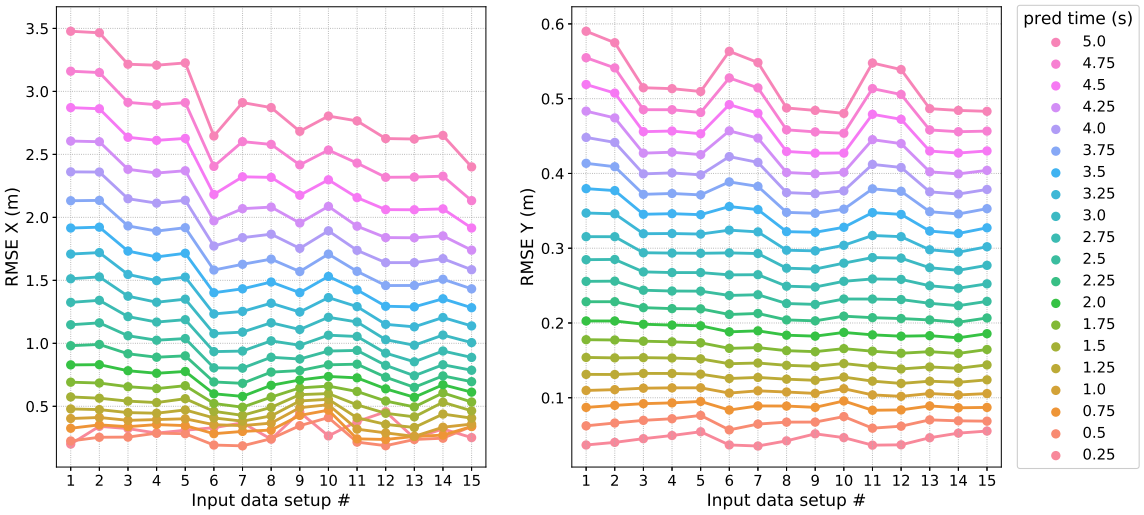


Figure 6.8: Analysis of the RMSE for different variations of numerical input to our LSTM model trained on the *On-board* data set for 8 epochs.

for instance, ones combining LSTMs with social pooling layers as in Deo and Trivedi (2018a) or Alahi et al. (2016) for future work.

6.3.1 Evaluation of the LSTM-based prediction models

In this section, we will investigate the LSTM-based prediction models.

Hyperparameter analysis

For the LSTM-models, we firstly investigated the composition of the input data to the model to get an idea, what kind of information is useful for the task of motion prediction. Therefore, we trained several instantiations of our LSTM-network architecture on the *On-board* data set D_1 for different variations of the input data, for 8 epochs each. Table 6.3 summarizes the different setups and shows what kind of data is included in each setup shown in Fig. 6.8. The simplest setting (setup 1) is using only the positional information of the target vehicle as input and its instantaneous velocity as additional information in the embedding without including any information about the ego-vehicle. We refer to this setting as the default setting in this section. In addition, we also analyze settings, where additional information like velocity (setups 6-15) and acceleration (setups 11-15) of the target vehicle are available to the system. Furthermore, if all dynamic information is available relative to the ego-vehicle, there are other features that could be useful for motion prediction such as the current curvature of the road or the current velocity or steering values of the ego-vehicle itself. For instance, if the ego-vehicle performs a lane change or the road bends, this will influence the relative motion of all other vehicles while this information most likely will not be available from just the positions of the target vehicle. On the other hand, if such information is available to the system, it would improve the model's capability of abstracting and inferring correlations between the available information in such situations. In this evaluation, we include the history of the ego-vehicle's information to the input data and future values to the embedding, whereas we include additional information about the target vehicle to the input data only.
Figure 6.8 depicts the RMSE (y-axis) for each input data setup given in table 6.3 on the x-axis at each prediction time step. Each tick on the x-axis corresponds to one input setup, whereas each group of 5 ticks from left to right corresponds to one fixed setup for the target vehicle. The left group contains only the default data about the target vehicle (setups 1-5 in table 6.3), the middle group contains the history of the target vehicle's velocity (setups 6-10 in table 6.3) and the right group contains additionally the history of the target vehicle's acceleration (setups 11-15 in table 6.3).

Setup #	Included target vehicle data			Included ego-vehicle data			
	Position	**Velocity**	**Acceleration**	**Acceleration**	**Lane Border**	**Lane Curvature**	**Steering**
1	✓						
2	✓			✓			
3	✓			✓	✓		
4	✓			✓	✓	✓	
5	✓			✓	✓	✓	✓
6	✓	✓					
7	✓	✓		✓			
8	✓	✓		✓	✓		
9	✓	✓		✓	✓	✓	
10	✓	✓		✓	✓	✓	✓
11	✓	✓	✓				
12	✓	✓	✓	✓			
13	✓	✓	✓	✓	✓		
14	✓	✓	✓	✓	✓	✓	
15	✓	✓	✓	✓	✓	✓	✓

Table 6.3: Summary of the input data setups of the different models evaluated in Fig. 6.8.

Each tick within one group corresponds to one setting for the ego-vehicle, whereas again from left to right the amount of available information increases. Therefore, the rightmost tick contains information about the ego-vehicle's acceleration, distance to the lane border as well as the curvature of the current lane and its current steering values (setup 15 in table 6.3). We observe that adding more information about both, the ego- and target vehicle, indeed improves prediction accuracy significantly: the difference between the best and worst setting is more than 1 m in x-direction and more than 0.1 m in y-direction. For both dimensions, setup 15 using all available information outperforms the simpler setups. However, there are some interesting peculiarities visible in this analysis that are worth noting. For instance, the input information improving performance in x-direction the most appears to be the target vehicle's velocity (setups 1-5 vs. setups 6-10). Furthermore, the target vehicle's acceleration does not yield further significant improvements given its velocity is available. Interestingly, setup 6 using only the target vehicle's velocity as additional information is closely behind the best setting in x-direction. Furthermore, the performance boost of the setting using all available information (setup 15 or the rightmost tick) over the prior setting comes from the ego-vehicle's steering, which only appears when its acceleration information is also available. For the y-direction, we observe similar trends in that the target vehicle's acceleration does not yield significant improvements if its velocity is already given. Here however, the information about the ego-vehicle's distance to the lane borders appears to be the input that gives the most significant improvements in y-direction (setup 2 vs. 3, setup 7 vs. 8 and setup 12 vs. 13). That makes sense, since these inputs encode information about the ego-vehicle's motion in y-direction when, for instance, performing lane changes. As setup 15 using all available information (the rightmost tick in Fig. 6.8) is by far outperforming all other settings in x-direction and is on par with the best in y-direction, we use this data setup for further analyzing the hyperparameters of the LSTM-model.

We continue our hyperparameter analysis by inspecting the number of dimensions within the LSTM cells. In our initial experiment, we used 80 dimensions in each of the LSTM encoder and decoder cell. Here, we investigate if adding more dimensions improves the models' prediction performance. Again, we train the model for 8 epochs. Figure 6.9a depicts the RMSE for models with 80, 150, 200 and 500 dimensions

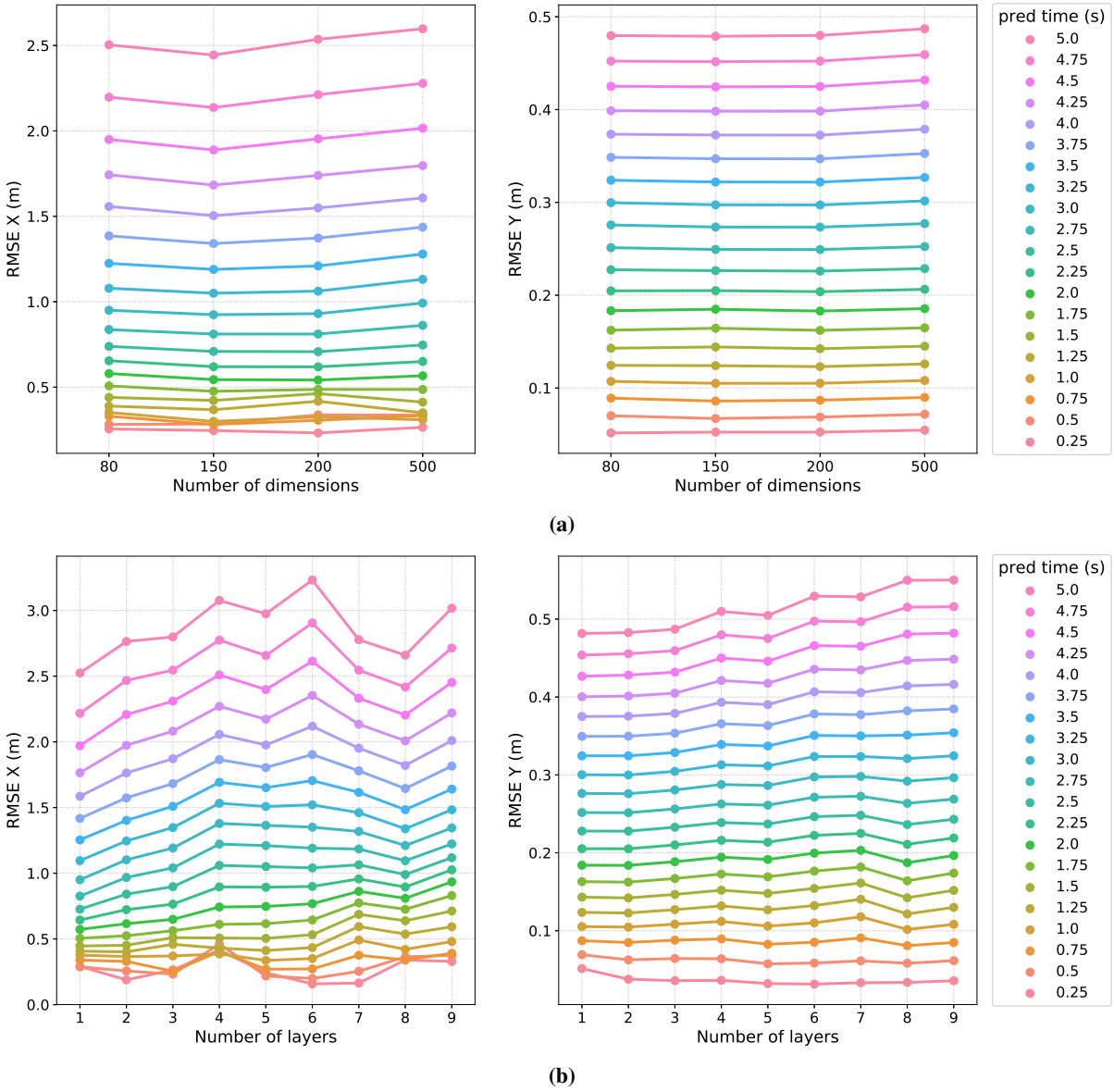


Figure 6.9: Visualization of the RMSE for different parameter tests of our LSTM-model trained on the *On-board* data set for 8 epochs: (a) depicts the RMSE when varying the number of dimensions in each LSTM cell (b) visualizes the RMSE when varying the number of layers, i.e., the number of encoder and decoder LSTM cells are used in the network.

in the LSTM cells. We observe that the model with 150 dimensions performs best in x-direction whereas all models show comparable performance in y-direction. However, increasing the number of dimensions beyond 150 per LSTM does not improve the models' accuracy but rather deteriorates the performance. Therefore, we fix the number of dimensions within the LSTM cells to 150 for further investigation.

In the next step, we inspect how the number of layers in our network architecture influences the model's performance. Here, one layer is a pair of one LSTM encoder and decoder cell each. Thus, a model with 2 layers consists of a sequence of 2 LSTM encoder cells followed by a sequence of again 2 LSTM decoder cells. Figure 6.9b visualizes the RMSE of models with 1 up to 9 layers trained for 8 epochs. This analysis shows that the model using only one layer performs best and that increasing the number of layers and thus using a deeper network architecture does not improve the model's performance. On the contrary, more layers lead to worse accuracy in both dimensions. However, we trained all models for a fixed number of 8 epochs whereas deeper network architectures might demand a longer training process.

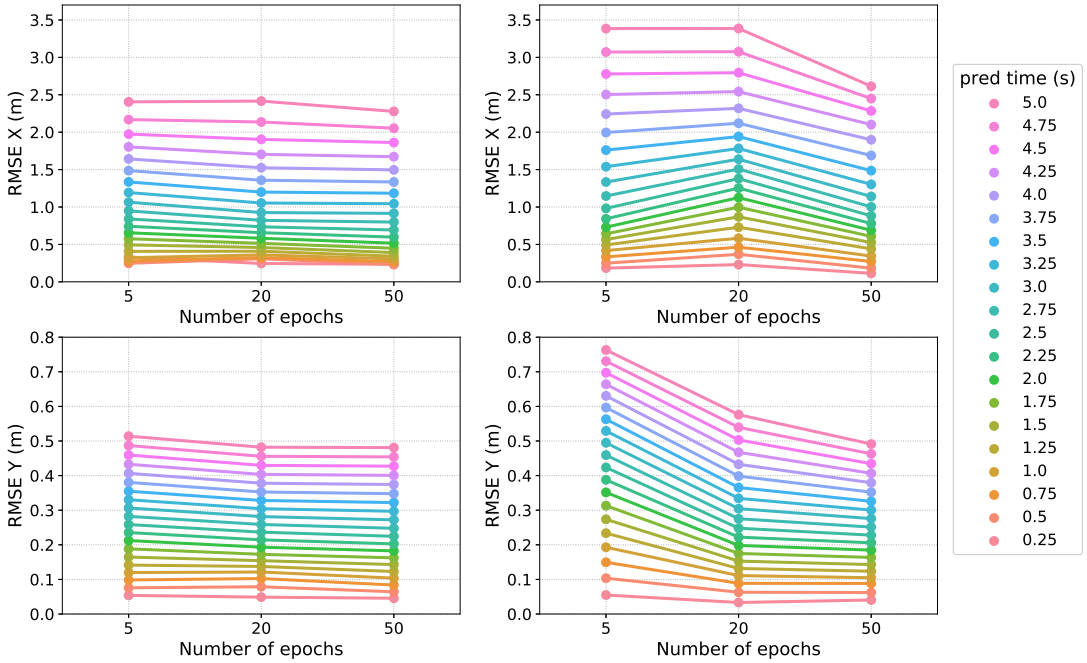


Figure 6.10: Analysis of the RMSE varying the number of layers and epochs of our LSTM model trained on the *On-board* data set. The left column shows the RMSE of a model with only one layer trained for 5, 20 and 50 epochs, while the right column shows the RMSE of a model with 10 layers trained for 5, 20 and 50 epochs.

In the next step, we therefore analyze the number of layers and number of epochs jointly to investigate if larger network architectures trained for more epochs improve prediction performance. Figure 6.10 visualizes the results of this experiment: the left column depicts the RMSE of a model with only one layer trained for 5, 20 and 50 epochs, whereas the right column shows the RMSE of a model with 10 layers trained for 5, 20 and 50 epochs. We observe that training a deeper model for more epochs does improve its accuracy. However, if we compare the left and right plots in Fig. 6.10, we also find that by training a model with 10 layers for 50 epochs we only achieve the accuracy of the simpler single-layer LSTM model trained for 5 epochs. We conclude, that using more layers even with a longer training process (i.e., increased number of epochs) does not lead to improved prediction results. Thus, a LSTM model with one encoder and decoder cell each is not only the simplest network architecture but also the best in terms of accuracy and time needed for training.
We briefly summarize the findings of this section and fix the following set of parameters for subsequent sections: we use a LSTM model with one encoder and one decoder cell with 150 hidden dimensions each.

Model training

Using the aforementioned network architecture and hyperparameter set, we train one model instantiation for each encoding scheme mentioned in section 6.2.1, whereas only the input dimensionality of the encoder cell changes when varying the representation of the input data. Importantly, we focus on positional information as the only input for our LSTM models in this work for reasons of consistency to make all models as comparable as possible. Hence, we neglect for example the history of the target (or ego-) vehicle's velocity or acceleration as input here. Between the two data sets, the only difference between models is the auxiliary data, that is used as additional input to the LSTM decoder cell at each time step. For both data sets, we use the instantaneous velocity of the target vehicle to aid the model predicting the future trajectory at every time step. As there is no ego-vehicle present in the *NGSIM* data set D_2, we use

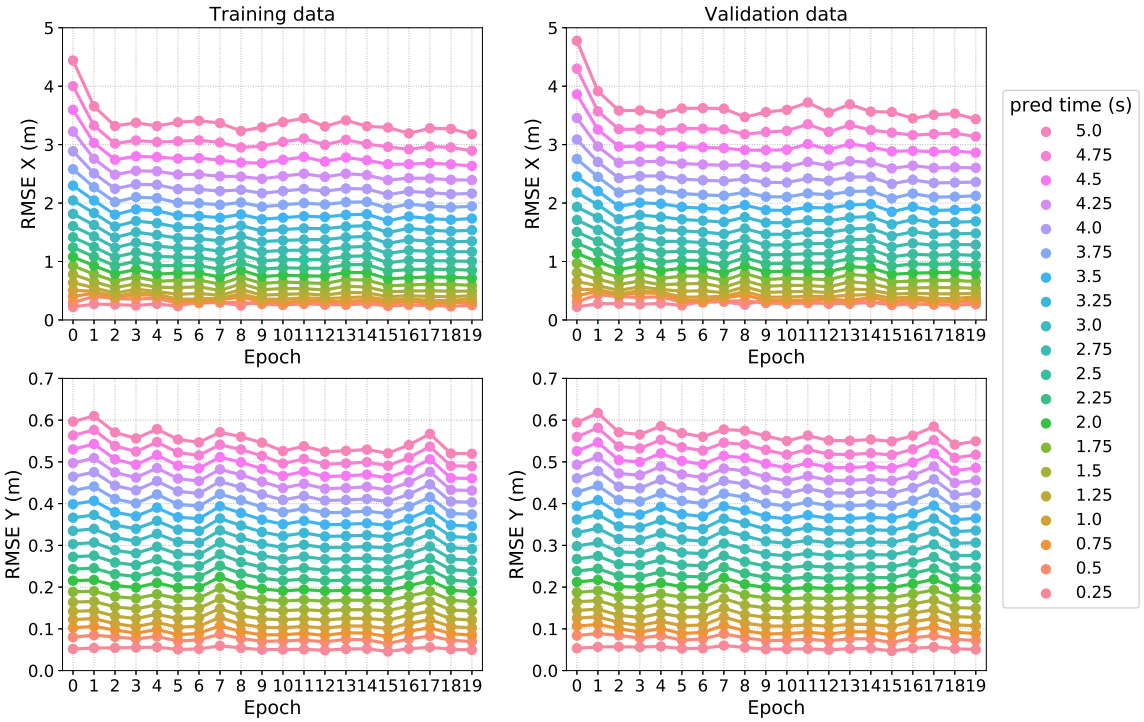


Figure 6.11: Development of the RMSE at every prediction time step during the training process of the LSTM SPA 3 model for each epoch on the training (left column) and validation part (right column) of the *On-board* data set. One observes comparable trends on both, training and validation set and that the RMSE does not significantly decrease after 10 epochs.

no further auxiliary data. For the *On-board* data set D_1, we use the ego-vehicle's predicted acceleration and the estimated curvature of the ego-vehicle's current lane. Although this is future information, we argue that it is solely about the ego-vehicle, which we expect to be available at the time the prediction is to happen. We assume, that an automated vehicle, in order to safely navigate, will have an estimation of the future lane curvature as well as the acceleration values of its own planned trajectory. Furthermore, we employed early stopping, that is, we trained our models for 10 epochs as we found that the models' performance stagnate on both, training and validation data sets, when training for up until a total 20 epochs. Figure 6.11 visualizes this result by showing the development of the RMSE of the LSTM SPA 3 model using the SPA-power-with-ego vector representation $(\mathbf{S}_t^{ego})_{t_0}^{t_N}$ as input for the training (Fig. 6.11 left column) and validation part (Fig. 6.11 right column) of the *On-board* data set D_1. On the y-axis of each sub-figure, we have the RMSE while the x-axis from left to right depicts the result after each epoch during the training process. Each colored line illustrates the RMSE of the model for one particular prediction time step while all points with the same value on the x-axis depict the model's performance after the respective epoch during the training process.

Evaluation

Figure 6.12 visualizes the RMSE of all LSTM-based models on the validation-set $V_1 \subset D_1$ of the *On-board* data set using 512-dimensional vectors. Figure 6.12a shows the performance on the complete validation-set, whereas Fig. 6.12b depicts only situations with at least 3 other vehicles present, the distance between the target and the ego-vehicle being lower than 20 m and the distance between the target and the closest other vehicle being less than 10 m. We observe that all approaches yield comparable results with notable differences in certain situations. Although the models employing the SPA-power encoding schemes tend to perform worst in x-direction, we observe that they perform better in y-direction

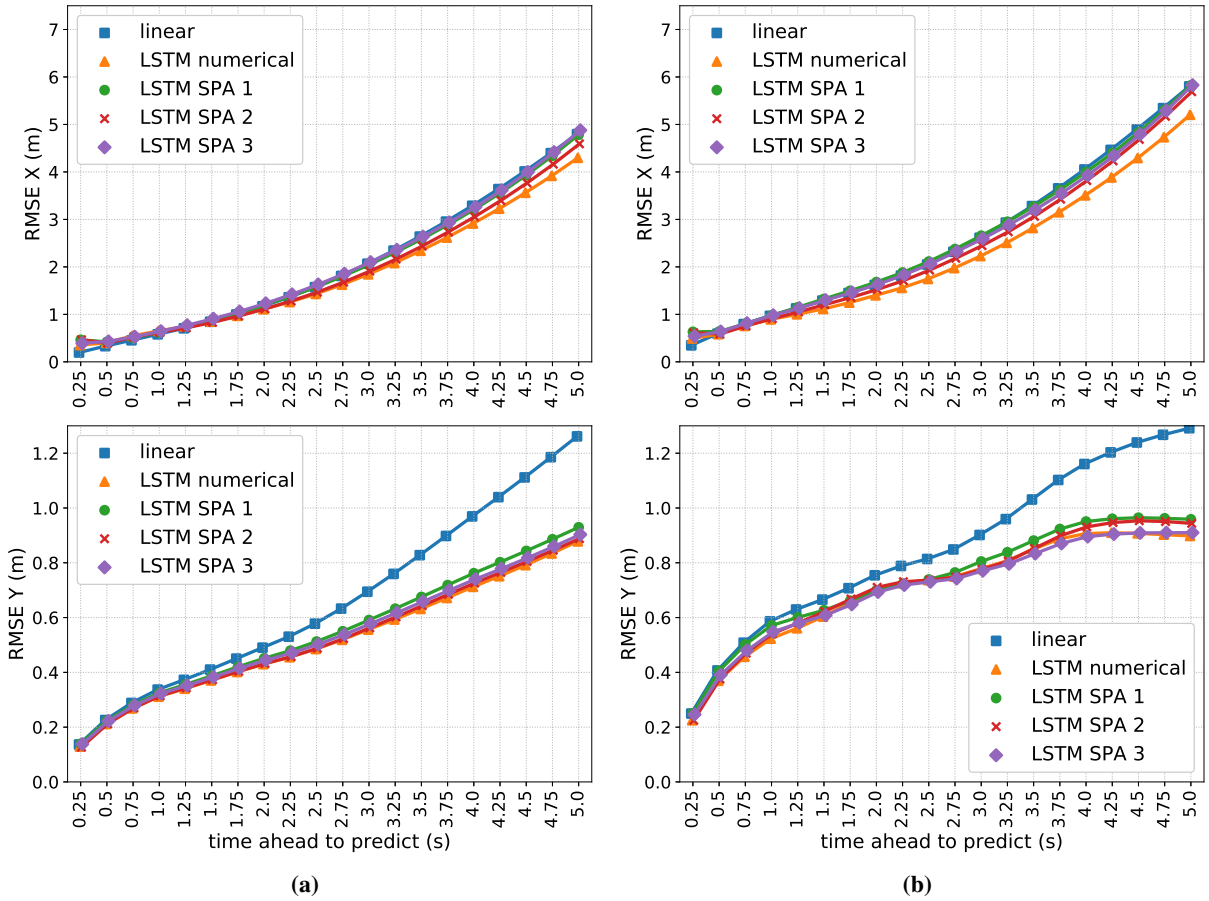


Figure 6.12: Visualization of the RMSE of all LSTM models on the *On-board* data set: (a) shows the complete validation set $V_1 \subset D_1$ (b) shows the subset of situations with at least 3 other vehicles present and distance between the target and ego-vehicle lower than 20 m and between target and closest other vehicle lower than 10 m.

in crowded situations with closely driving vehicles with the LSTM SPA 3 model ranking best along the LSTM numerical model.

To further investigate this result, we evaluated certain metrics, chosen to characterize crowded and potentially dangerous situations, for items in the validation set, where the LSTM SPA 3 model outperforms all other approaches with respect to the RMSE in y-direction (see Fig. 6.13). We observe that the number of samples, where the distance between the target and the ego-vehicle and/or the closest other object being small is significantly higher when the LSTM SPA 3 model outperforms all other approaches. For samples where the LSTM SPA 3 model performs best, the number of samples with a distance less than 20 m between the target and ego-vehicle is 50.5 % higher compared to samples where one of the other models performs best. For distances less than 20 m between the target vehicle and the closest other vehicle, the number of samples is still 11.4 % higher when the LSTM SPA 3 model performs best. Finally, the number of situations with at least 3 other vehicles present is also 7.8 % higher compared to samples where one of the other models performs best. However, we aim to investigate more sophisticated options such as clustering methods in future work to uncover other, potentially more meaningful features compared to the ones shown here, distinguishing between situations where LSTM SPA 3 performs best compared to another model showing the best performance.

Figure 6.14 visualizes the RMSE of all LSTM-based models on the validation-set $V_2 \subset D_2$ of the *NGSIM* data set for 512-dimensional vectors (Fig. 6.14a) and for 1024-dimensional vectors (Fig. 6.14b). We observe, that all LSTM models achieve a very similar performance (almost identical in y-direction) with

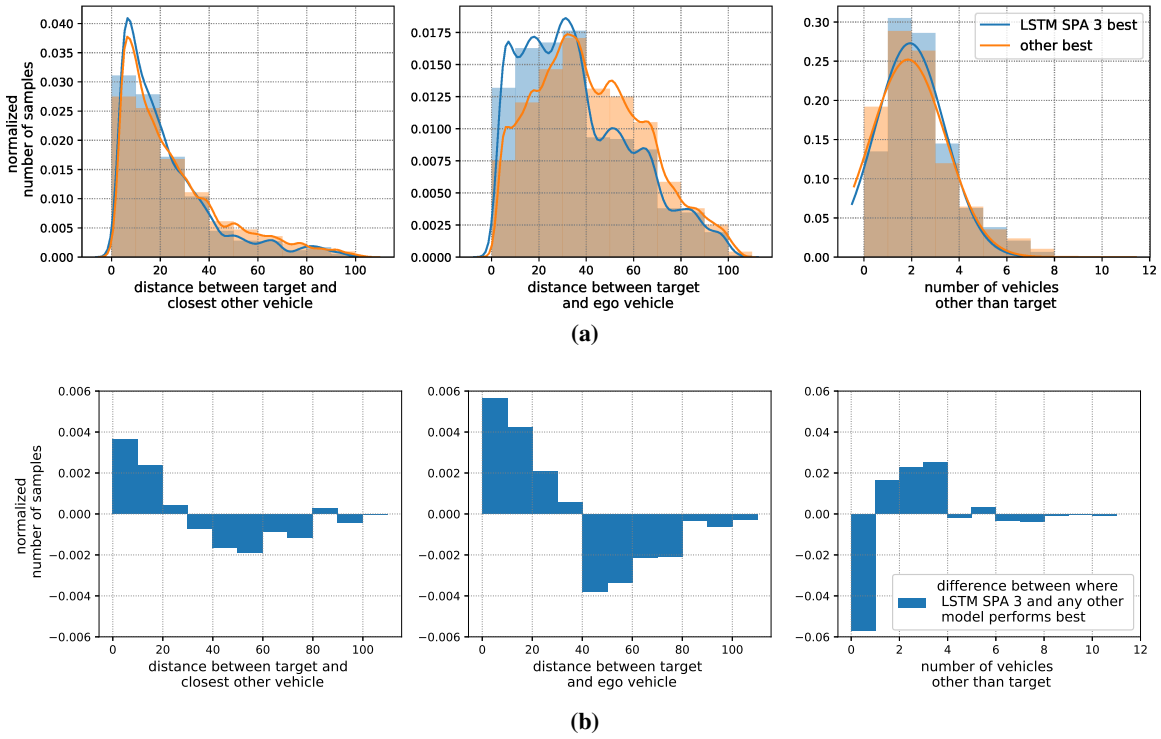


Figure 6.13: Metric evaluation specifying situations where the LSTM SPA 3 model outperforms all other approaches regarding the RMSE in y-direction on the *On-board* data set D_1. In the upper row (a), blue bars illustrate samples where LSTM SPA 3 performs better than all other models while the orange bars depict samples where one of the other models performs best. From left to right, the plots in (a) illustrate the distance between the target vehicle and the closest other vehicle, the distance between the target and the ego-vehicle and the number of vehicles other than the target. The lower row (b) illustrates the difference between the blue and orange bars in the corresponding plot in (a).

LSTM SPA 1 achieving the best performance in x-direction being on par with the numerical encoding for 512-dimensional vectors. For 1024-dimensional vectors, LSTM SPA 1 even slightly outperforms all other approaches in x-direction, whereas we do not observe significant improvements in y-direction.

Evaluation of models trained on data set variations In section 6.1.4, we have already seen that our prediction models using neural networks have to deal with imbalanced data sets containing significantly more straight driving than lane changes performed by the target vehicle. Hence, training any learning system on all the samples of both our data sets will expose the system to a significantly higher amount of data, in which most likely already a simple linear prediction model performs reasonably well. In this section, we therefore analyze the influence of the training data set on our LSTM models and if there are significant differences in the performance of models trained on the complete data sets and on subsets consisting only of such samples that contain a lane change of the target vehicle. We conduct this analysis on the *On-board* data set only. Figure 6.12 showed the performance of our LSTM-based models trained on the complete data set. Here, we train models with the exact same neural network architectures and encoding schemes of the input just on different data sets, namely the subset of samples containing a lane change performed by the target vehicle. We consider both types of target vehicle lane changes, namely those performed during the trajectory history as well as lane changes performed in the future trajectory to be predicted, as input for the models. Figure 6.15 shows a comparison of different setups regarding training and evaluation data for our LSTM-based trajectory prediction models. We also keep the simple linear prediction model based on a constant velocity assumption for reference in the figures, since it is not a data-driven learning model and is therefore invariant under the changes of the training

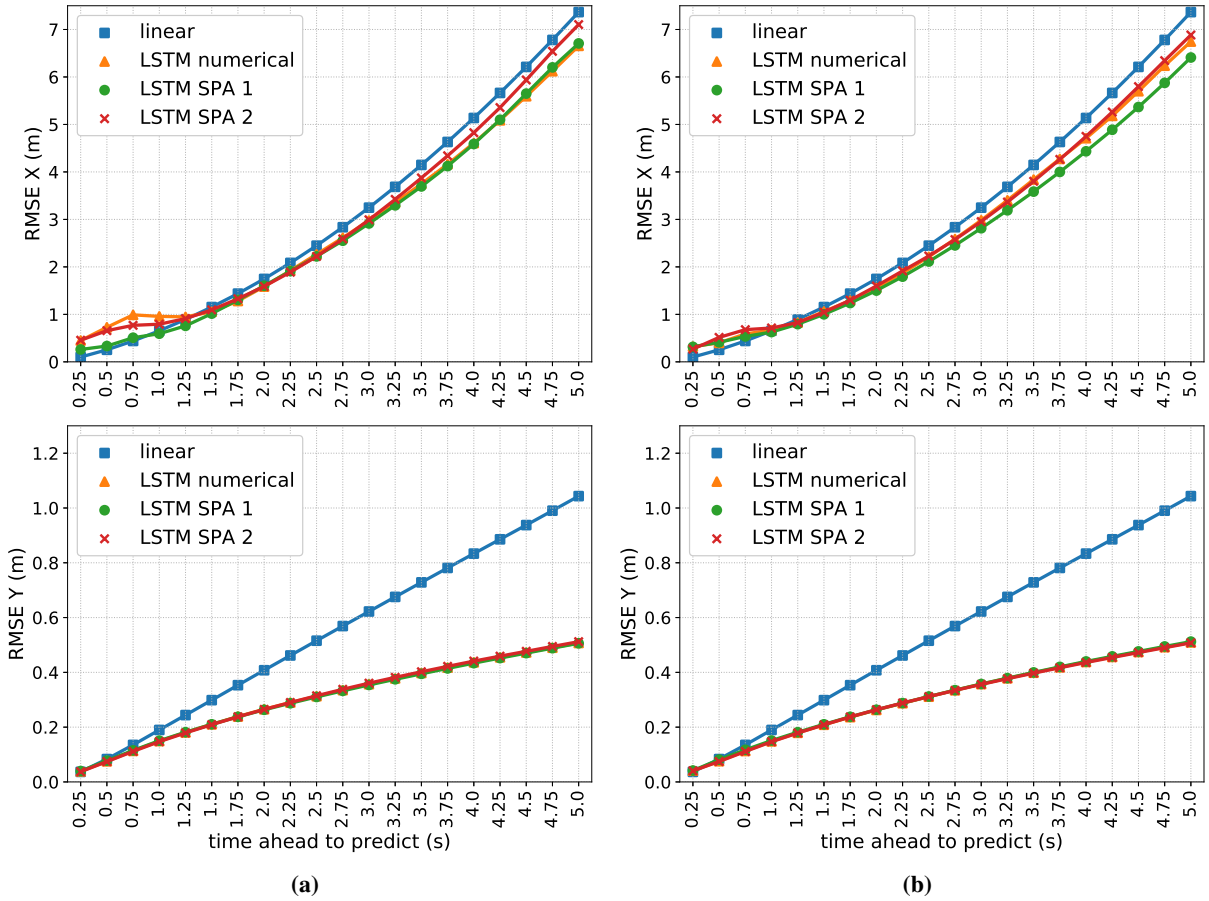


Figure 6.14: Visualization of the RMSE of all LSTM models on the *NGSIM* validation set $V_2 \subset D_2$ using (a) vectors of dimension 512 for the SPA-based models and (b) using vectors of dimension 1024 for the SPA-based models.

data. Figures 6.15a, 6.15c, 6.15e and 6.15g show the RMSE for models trained on the complete data set while Fig. 6.15b, 6.15d, 6.15f and 6.15h show the same models evaluated on the same samples but trained only on data samples including a target vehicle lane change. On the other hand, the upper row of Fig. 6.15, i.e., Fig. 6.15a - 6.15d, illustrates the performance of the models evaluated on either the complete data set or only on the lane changes in all data samples, while the lower row of Fig. 6.15, i.e., Fig. 6.15e - 6.15h, shows the performance of the models evaluated on samples containing crowded driving situations or crowded situations containing even a target vehicle lane change.

We observe that the models that have been trained only on samples containing a lane change performed by the target vehicle tend to achieve worse results than the models trained on the complete data set, when being evaluated on the entirety of all data samples (Fig. 6.15a,, 6.15c, 6.15e and 6.15g). Interestingly, the performance of the models based on the convolutive power encoding scheme (LSTM SPA 1 and 3) deteriorates more significantly compared to the other data-driven models, especially in lateral (y) direction. However, if we evaluate the same models only on the samples containing a target vehicle lane change, their performance changes significantly (Fig. 6.15b, 6.15d, 6.15f, 6.15h). We recapitulate the findings of section 6.1.4 that lane changes influence the lateral (y) position values more severely than the longitudinal (x) position compared to straight driving samples. Therefore, the performance difference between the models trained on lane changes only and models trained on the complete data set, as we would expect, is also not that significant in longitudinal direction.

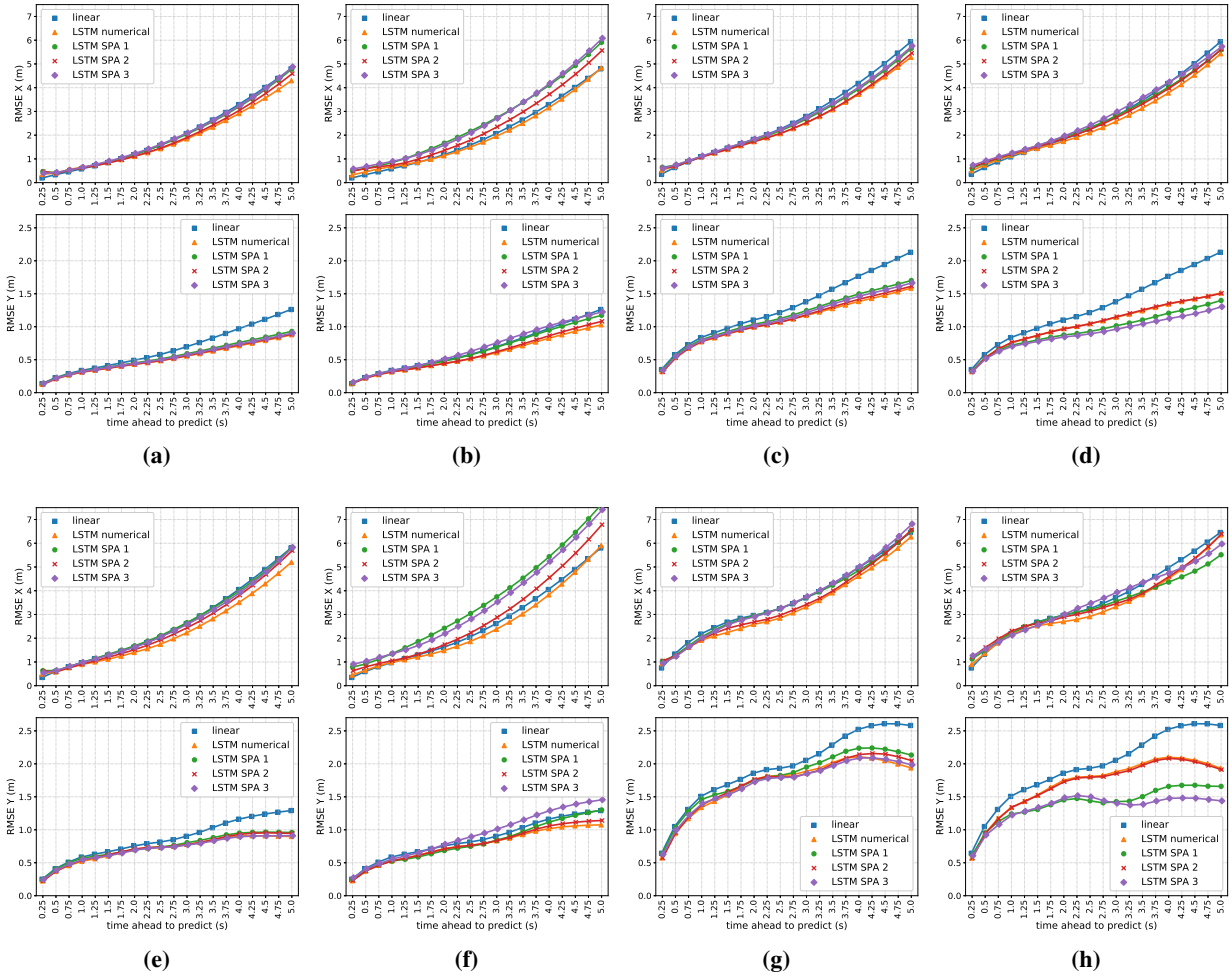


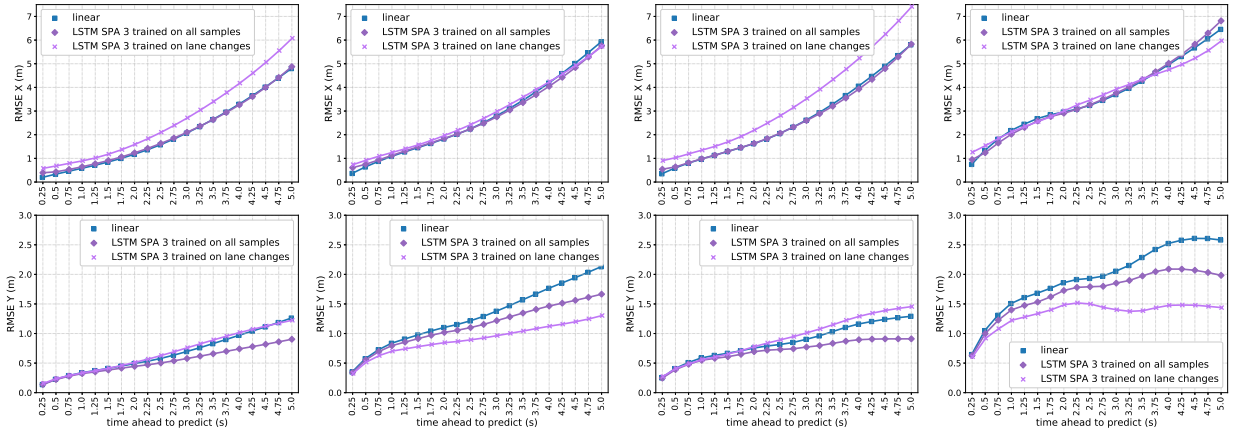
Figure 6.15: Visualization of the RMSE performance of our LSTM models for different data setups. Figures (a), (c), (e) and (g) show the RMSE for models trained on the complete data set, while Fig. (b), (d), (f), (h) show the same models evaluated on the same samples but trained only on the samples including a target vehicle lane change.

Considering the lateral (y) direction however, we observe a significant change between both model and evaluation variants. If the models are trained only on lane change samples, the LSTM SPA 1 and 3 models outperform all other models in lateral direction when evaluated only on the data samples containing a lane change while their performance in longitudinal direction does not change significantly compared to the models trained on the complete data set.

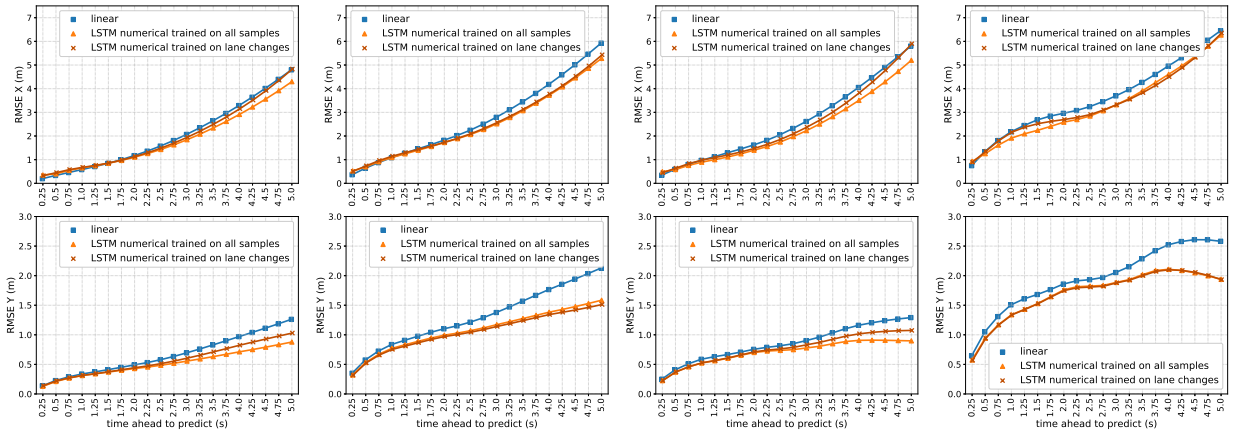
Setup ID	Training set	Evaluation set
(a)	all samples	all samples
(b)	lane change samples	all samples
(c)	all samples	lane change samples
(d)	lane change samples	lane change samples
(e)	all samples	crowded samples
(f)	lane change samples	crowded samples
(g)	all samples	crowded lane change samples
(h)	lane change samples	crowded lane change samples

Table 6.4: Summary of the data samples used for the evaluations shown in individual sub-figures of Fig. 6.15.

So far, we have only compared either all models trained on the complete data set or all models trained only on the lane change samples. However, we are also interested in how the performance for particular models changes when modifying the training data set. Figure 6.16 shows a direct comparison of the LSTM SPA 3 model (Fig. 6.16a - 6.16d) as well as the LSTM numerical model



(a) All samples **(b)** Lane changes **(c)** Crowded **(d)** Crowded lane changes



(e) All samples **(f)** Lane changes **(g)** Crowded **(h)** Crowded lane changes

Figure 6.16: Visualization of the changing RMSE performance of particular prediction models depending on the data they have been trained on. The first four figures (a) - (d) illustrate the difference between the LSTM SPA 3 models when changing their training data, while the last four figures (e) - (h) shows the same comparison for the LSTM numerical models.

(Fig. 6.16e - 6.16h) with different training data sets. In this direct comparison, we observe that there is no significant difference in the performance of the numerical LSTM models trained on different samples for both evaluation sets containing either the complete data set or only the target vehicle lane changes. For the LSTM SPA 3 model however, we observe significant improvements for the model trained on the lane change samples when evaluated on the lane changes performed by the target vehicle compared to the model trained on all data samples. This result indicates that either the numerical model trained on all samples generalizes sufficiently well to all possible situations compared to the convolutive power based model. However, we have already seen, that both trajectory data sets show a significant imbalance towards straight driving compared to lane change maneuvers (cf. section 6.1.4), which is the same for all models. On the other hand, the results shown here could also hint, that the convolutive power model encapsulating the prior motion not only of the target vehicle but also other vehicles in its surroundings is better suited to predict lane change maneuvers given a more balanced data set. These results suggest, that there is not only room for improvement for the models investigated here, but also other data-driven models used for trajectory prediction in the literature, by researching and evaluating the influence of the

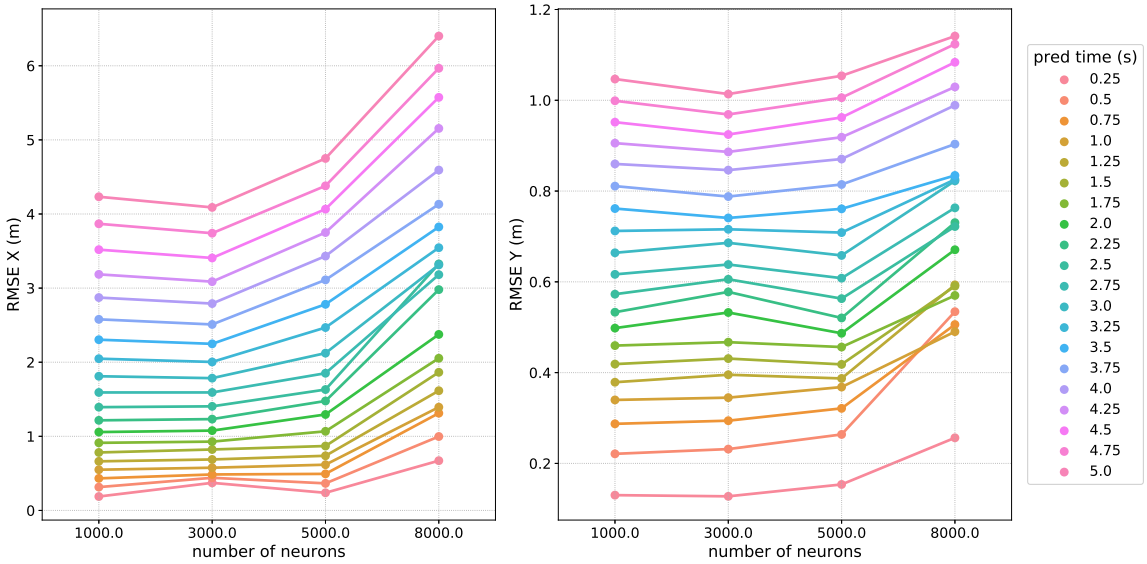


Figure 6.17: Analysis of the RMSE for a varying number of neurons in the learning population of our NEF model trained on numerical input from the *On-board* data set.

distribution of driving situations within the training data.

6.3.2 Evaluation of NEF-based feed-forward prediction models

In this section, we evaluate the simpler, feed-forward neural networks for trajectory prediction to compare our LSTM-based models to.

Hyperparameter analysis

For our NEF networks, the main parameter influencing the models' performance is the number of neurons in the learning population, i.e., the hidden layer in terms of traditional neural networks, as well as the neuron model. For simplicity, we use Nengo's rate-variant of the LIF neuron model. Here, we inspect the number of neurons in the learning population in more detail. From the NEF-theory (Eliasmith and C. H. Anderson, 2003) we know that increasing the number of neurons in a population yields a more accurate representation of the data encoded in the population's activity. Thus, we expect more accurate predictions when increasing the number of neurons. Figure 6.17 shows the RMSE of different model instantiations with varying number of neurons, namely 1000, 3000, 5000 and 8000 neurons. We observe that a number of 3000 spiking neurons is sufficient and further increasing the number of neurons does not improve the model's prediction accuracy. Note that the order of magnitude of the RMSE in Fig. 6.17 is significantly higher than for the figures visualizing the LSTM models RMSE inspected in the previous section. This is due to the fact that the NEF models investigated here receive less amount of information, namely only position data and instantaneous velocity of the target vehicle, compared to the LSTM models. Here, we are only interested to find the best possible parameter setup, whereas we will evaluate both models with comparable setups in later sections.

Model training

We train two different variants of our simpler NEF-models using numerical input (NEF numerical) as well as the SPA-power-with-ego (NEF SPA 1) and SPA-power encoding (NEF SPA 2) for the *On-board* and the *NGSIM* data set respectively. The neural weights are calculated using Nengo's default least-squares-optimization method with the exception, that we calculate the weights over smaller subsets of

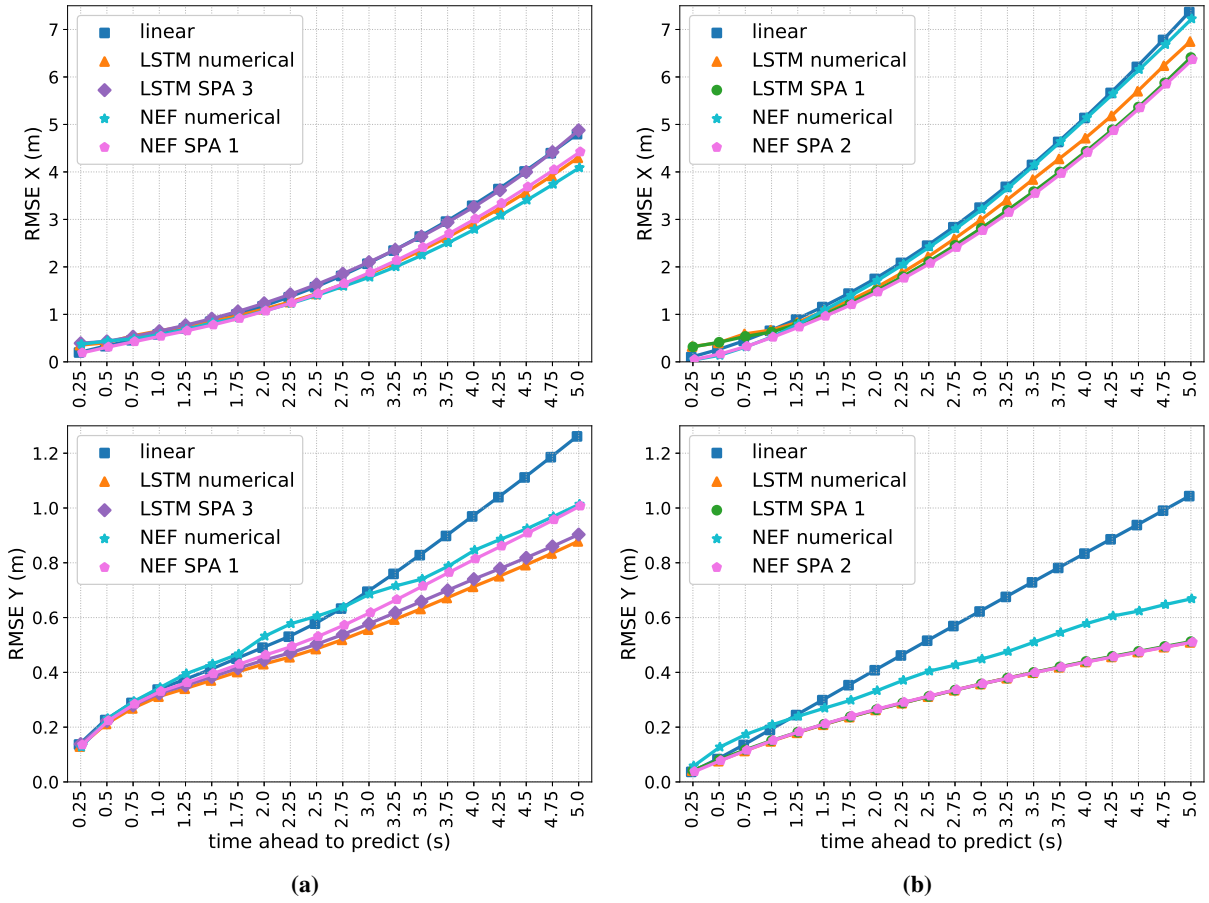


Figure 6.18: Visualization of the RMSE of all NEF-network models (a) on the *On-board* validation set $V_1 \subset D_1$ using 512-dimensional vectors for the SPA-power vectors and (b) on the *NGSIM* data set D_2 using 1024-dimensional vectors for the SPA-power vectors.

the input data for computational reasons. Here, we adapt the input data such that for the numerical NEF-model, we use the vector $(x_{t_0}, \ldots, x_{t_N}, y_{t_0}, \ldots, y_{t_N}, v)$ as input with v denoting the instantaneous velocity of the target vehicle. For the SPA-power encoding schemes, instead of flattening the whole sequence of vectors into one giant single vector, we created a single semantic vector by summing the first, middle and last element of the original vector sequences

$$\hat{\mathbf{S}} = \mathbf{S}_{t_0} \oplus \mathbf{S}_{t_{N/2}} \oplus \mathbf{S}_{t_N} = (\hat{s}_0, \ldots, \hat{s}_{D-1}). \tag{6.3}$$

We only sum up these vectors instead of the whole sequence $(\mathbf{S}_t)_{t_0}^{t_N}$ to avoid the accumulation of noise in the vector representation. Note that thereby the NEF model using the SPA-power representation does not use the full trajectory history but only selected time steps, namely those visualized in Fig. 6.6. We also include the instantaneous velocity v of the target vehicle as an additional vector element, which yields $(\hat{s}_0, \ldots, \hat{s}_{D-1}, v)$ as input of our model. To make these simpler models as comparable as possible to the LSTM models in terms of information available to the network, we add the instantaneously velocity of the target vehicle as an additional element to the input, since there is no intermediate embedding vector here where it could be included.

Evaluation

Figure 6.18 visualizes the RMSE of our NEF-network models on both data sets. The NEF-network using the SPA-power encoding schemes processes 512-dimensional for the *On-board* (NEF SPA 1) and 1024-

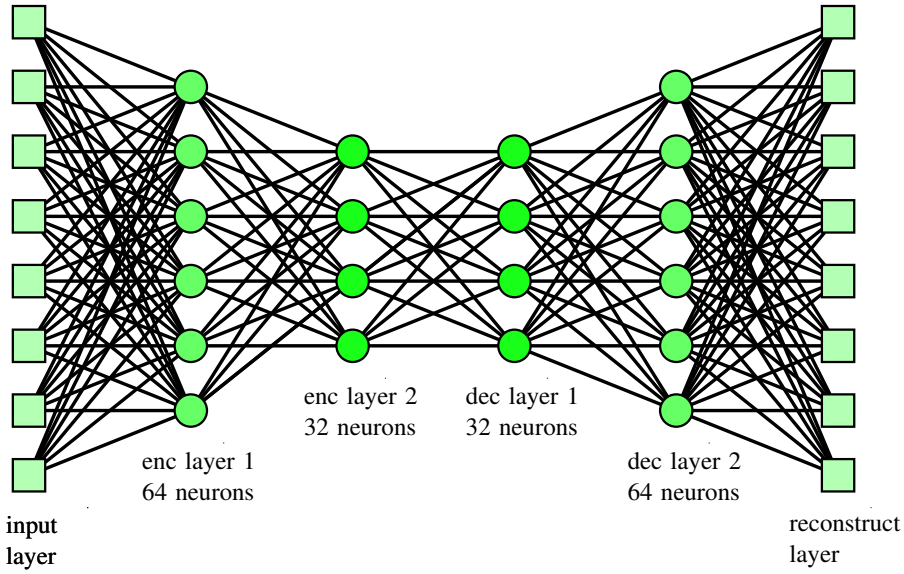


Figure 6.19: A schematic visualization of our autoencoder neural network architecture.

dimensional vectors for the *NGSIM* data set (NEF SPA 2). For reference, we included the performance of the most relevant LSTM models, namely LSTM SPA 1 and 3 for the *NGSIM* and *On-board* data set respectively as well as LSTM numerical, in Fig. 6.18 as well. We observe that, despite a simpler network architecture and learning algorithm, the NEF-networks achieve a performance comparable to the more sophisticated LSTM models on both data sets. For the *NGSIM* data set, the NEF SPA 1 model performs on par with its LSTM model counterpart LSTM SPA 3. In this case, the NEF-model is not only simpler, but also has access to less information as its input data is a sum of a subset of the input sequence used for the corresponding LSTM-model.

6.3.3 Evaluation of the unsupervised anomaly detection

In this section, we analyze and evaluate the results of the performance of the anomaly detection network introduced in section 6.2.4. We train a fully-connected autoencoder neural network with 4 hidden layers consisting of 64, 32, 32 and 64 neurons in unsupervised fashion to generate replicates of the original vectors used as input to the network. Figure 6.19 shows a schematic visualization of the network's architecture. We use the reconstruction error between the original vectors and the replicates generated by the neural network as anomaly score and calculate a threshold for the error to detect outliers based on the percentage of anomalies we expect in the data set. For this evaluation, we use 10 % for the amount of expected outliers within the data set and trained the network for 100 epochs on the vectors encoding the current scene using the convolutive power representation as described in section 6.2.1.
Since the data we are using to train the network is unlabeled, i.e., we do not have any information available which vectors belong to unusual or atypical situations, we have no way of comparing the results produced by the neural network with actual ground truth data. Hence, our only option is to analyze the anomalies detected by our neural network with respect to certain characteristic values describing the driving situation and compare them to the values of the complete data set. If this comparison uncovers significant differences between the anomalies detected by the neural network and the entirety of all data samples, we can at least conclude that the anomalies are reasonably different from the rest of the data set and furthermore, that there is sufficient information encoded in our vector representation to unravel them. For this analysis, we use the same metrics already analyzed in section 6.3.2 to characterize crowded and potentially dangerous driving situations, namely the distance between the target and the ego-vehicle, the distance between the target and the closest other vehicle and the number of other vehicles present in the scene. Figure 6.20 visualizes the distribution of these metrics on the set of anomalies produced by the

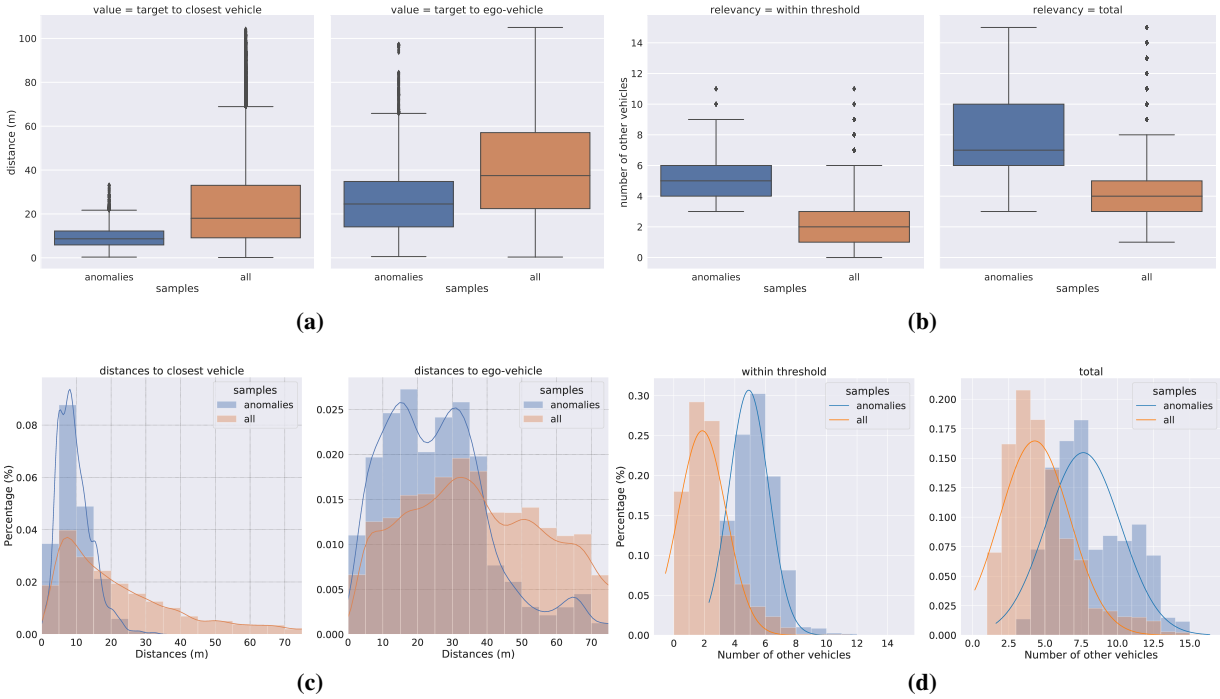


Figure 6.20: Visualization of the results of the autoencoder neural network used for unsupervised anomaly detection on the *On-board* data set. The figures show the distribution of certain characteristic values, namely distances between the target and other vehicles (Fig. (a) and (c)) as well as the number of other vehicles (Fig. (b) and (d)), for situations classified as anomalies and for the complete data set. The upper row shows box plots to visualize the difference between anomalies and the complete data set, whereas the lower row shows histograms for a more in-depth visualization of the metrics' distribution.

neural network and on all data samples in the *On-board* data set. Figure 6.20a shows box plots of the distances between the target and the closest other vehicle (left plot in Fig. 6.20a) as well the target and the closest other vehicle (right plot in Fig. 6.20a) while Fig. 6.20c shows the same data, but illustrated as histograms. Figure 6.20b and 6.20d visualize a similar evaluation for the number of other vehicles within a certain distance (left plots), namely 40 m, and all other vehicles in the scene (right plots).

We observe a clear difference for both, the evaluation of the distances and the number of other vehicles, between the anomalies detected by our neural network and the complete set of data samples. Focusing on the distance information, the number of instances with smaller distances between the ego-vehicle or the closest other vehicle and the target is significantly higher for the anomalies than for the complete data set. While the mean distance between the target and closest other vehicle is slightly below 20 m for the complete data set, the mean distance for the anomalies is 10 m (cf. Fig. 6.20a) with clear concentration between 0 m to 15 m (cf. Fig. 6.20c). We observe a similar distribution for the distance between the target and the ego-vehicle, where the distances are more or less equally distributed around the mean of 40 m for the complete data set. For the anomalies, we observe a concentration of the distances between the target and the ego-vehicle below 40 m around the mean of 25 m. Regarding the number of other vehicles, the difference between the complete data set and the anomalies detected by our neural network is even clearer. For the complete data set, the mean number of other vehicles within 40 m to the target vehicle is 2, while the total mean number of other vehicles is around 4. Both numbers are significantly higher for the anomalies with a mean number of 5 other vehicles within 40 m and a mean of 7 other vehicles in total (cf. Fig. 6.20b). Looking at the distribution shown in the histograms in Fig. 6.20d, the picture becomes even clearer. There are no situations with less then 3 other vehicles within 40 m to the target

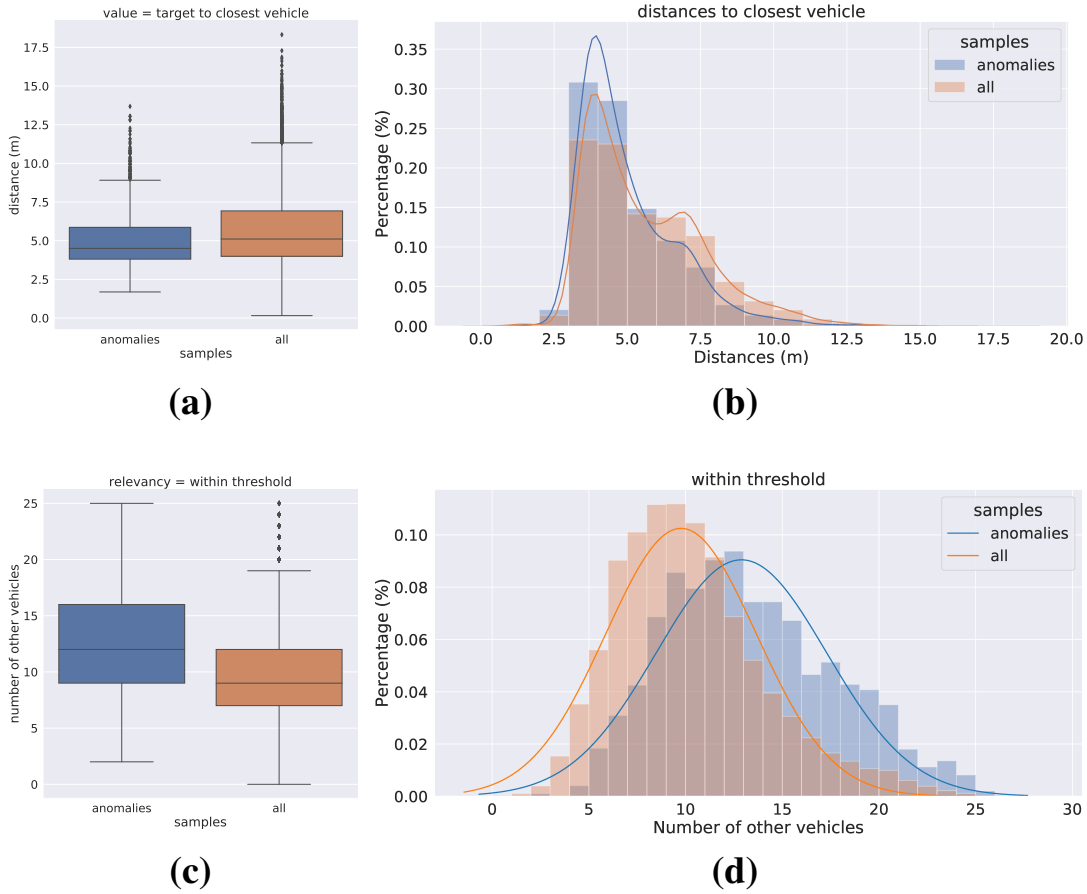


Figure 6.21: Visualization of the results of the autoencoder neural network used for unsupervised anomaly detection on the *NGSIM* data set. The figures show the distribution of distances between the target and other vehicles (Fig. (a) and (d)) as well as the number of other vehicles (Fig. (c) and (d)), for situations classified as anomalies and for the complete data set. The left figures (Fig. (a) and (c)) show box plots to visualize the difference between anomalies and the complete data set, whereas the right figures (Fig. (b) and (d)) show histograms for a more in-depth visualization of the metrics' distribution.

vehicle in the set of anomalies, whereas in this same range fall the majority of samples of the complete data set. We observe a similar distribution for the total number of other vehicles in the scene with the anomaly samples having at least 3 and the majority of examples having at least 4 other vehicles present in the scene. In contrast, the great majority of samples from the complete data set has at most 7 other vehicles present and the overall distribution is somewhat inverse to that of the anomalies.

Figure 6.21 shows a similar analysis for the *NGSIM* data set with a few systematic differences. Since the *NGSIM* data set is recorded with external cameras observing highway traffic, there is no ego-vehicle to evaluate regarding the distance to the target vehicle. Hence, we only analyze the distance between the target vehicle and the closest other vehicle (see Fig. 6.21a and 6.21b). Furthermore, since the external cameras record highway traffic on 6 lanes on a segment of 640 m length for the *NGSIM* data set, we needed to reduce the number of vehicles to be included in our vector representation to the ones most relevant for the prediction task. Therefore, we focus on vehicles within a distance of 40 m on lanes adjacent to the target vehicle's lane for the analysis of our anomaly detection network here as well (see Fig. 6.21c and 6.21d). While the differences between anomalies and the complete data set regarding the distance between the target and the closest other vehicle is not as significant for the *NGSIM* data set in comparison to the *On-board* data set, we still observe a similar tendency for the anomalies to capture

situations with smaller distances between the target and the closest other vehicle. For the number of other vehicles however, we also observe that the samples detected as anomalies by our autoencoder network tend to have more vehicles in the target vehicle's surroundings present than all the samples within the *NGSIM* data set.
In conclusion, our autoencoder neural network is able to detect a subset of anomalies consistently for both, the *On-board* and *NGSIM* data set, which show sufficiently significant differences to the complete data set regarding certain metrics. The results indicate, that the anomalies detected by the network have a tendency towards crowded situations with rather small distances between the target vehicle and the other vehicles in its surroundings. Since these are the kind of situations, where our prediction models based on the SPA-based convolutive power encoding tend to perform better than the other reference models, these results offer interesting directions for future research. For instance, we could combine the anomaly detection network based on our vector representation presented in this section with the behavior prediction networks to decide whether the current driving situation is potentially hazardous and needs more accurate predictions than less crowded or dangerous situations. We have already seen in section 6.3.1, that the LSTM models employing the convolutive power representation (LSTM SPA 1 and 3) outperform the other models in such situations particularly in lateral direction. Furthermore, one could also imagine to let the outlier detection network guide the training process of the behavior prediction networks. That is, we could train prediction models particularly on lane changes, the outliers and a similar amount of "normal" samples to create a more balanced training data set. We have already seen that the networks using semantic vectors benefit from a more focused and specific training data set, since the models trained particularly on lane changes improved especially in lane change situations compared to network variants trained on all samples. Finally, we could also investigate other anomaly detection algorithms in addition to the autoencoder models shown here to get a better idea what sort of data samples are actually outliers by evaluating how different models classify anomalies differently.

6.4 Summary

In this chapter, we showed a novel approach to encapsulate spatial information of multiple objects in a sequence of semantic pointers of fixed vector length. We used a LSTM sequence to sequence model as well as a simple feed-forward Spiking Neural Network to predict future vehicle positions from this representation. For each of those models, we implemented at least one reference model using other encoding schemes to compare their performance to. Furthermore, we compared all our models to a simple linear predictor based on a constant velocity assumption. We evaluated our models on two different data sets, one recorded with on-board sensors from a driving vehicle and one publicly available trajectory data set recorded with an external camera observing a highway segment and conducted a thorough analysis.
For the trained neural networks, simple feed-forward NEF models and more sophisticated LSTM models alike, we observe that most improvements over the linear model are achieved in y-direction. That makes sense as linear prediction is unable to account for lane-changes or driving curves, which are mainly characterized by non-linear changes in y-direction. We found that the LSTM models based on our SPA-power representation achieve promising results on both data sets. However, for the *On-board* data set, this encoding scheme achieves its best result in crowded and potentially dangerous driving situations, without clearly outperforming the other approaches on the whole data set (see section 6.3.1 and Fig. 6.12). Given these finding, we investigated situations, where the SPA-power representation does outperform all other approaches in y-direction and thereby came up with metrics characterizing such crowded situations (see Fig. 6.13). This result did not hold that clearly on the *NGSIM* data set D_2, since the LSTM models achieve an almost identical performance in y-direction on this data set. On the other hand, employing an unsupervised learning method for anomaly detection yielded a significantly higher amount of such crowded situations within the samples classified as anomalies compared to the normal samples. Additionally, we found that when training the LSTM models only on data samples containing a lane change performed by the target vehicle, that the model employing the SPA-power representation clearly outper-

forms all other approaches in y-direction when evaluated on the samples containing a lane change. These results suggest that training the whole system on a more balanced data set containing equally many lane change and straight driving samples could improve overall model performance.
Another interesting result of our experiments is the fact that the simple, feed-forward NEF networks show results comparable to the more sophisticated LSTM models. For those simple models, the SPA-power representation shows promising results comparable to the NEF model using numerical input on the *Onboard* data set and clearly outperforming it on the *NGSIM* data set (Fig. 6.18). Although the NEF models do not clearly outperform the LSTM models (which would be surprising), it is quite remarkable that they achieve results comparable to the more sophisticated models with a simpler network architecture, training procedure and, partly, less information. These results make those simple models using our proposed vector-representation as well as a numerical encoding scheme (possibly in combination with an online learning system like the one proposed in chapter 7) potential candidates to be deployed on dedicated neuromorphic hardware in mobile applications, as they can be efficiently implemented in a spiking neuron substrate. This could be an interesting, power-efficient approach in future automated vehicles.
Finally, given the results that there is not one model clearly outperforming all the others in all evaluated situations, we aim to implement an online model meant to be trained at run time to weight the predictions of several models, which have been trained offline, to improve over the individual predictors. We will introduce and evaluate such a model in the next chapter.

7 A mixture-of-experts online learning system for adaptive behavior prediction

In this chapter, we continue our work on behavior prediction from chapter 6. Although the various models developed there show already promising results, our analysis has shown that each of the developed models has its own strengths and weaknesses in particular driving situations or even perform differently regarding the direction of motion (lateral or longitudinal) of the target vehicle. Furthermore, they are intended to be trained offline and remain unchanged during deployment without any adaptation at run time. Hence, relying on just one specific predictor would lead to sub-optimal performance in situations where one of the other models is superior.

Therefore, we investigate a mixture-of-experts online learning model to select between several models, which have been previously trained offline, to achieve the best possible forecast. Importantly, this model is intended to be trained online, i.e., continuously updating its weights based on the data received at run time. This is one option to tackle the aforementioned issues with only one prediction model that has been trained offline before deployment. By training a model at run time, we expect improved performance of the mixture model over the individual input predictors. We aim to implement the online learning model to be independent of the type and number of prediction models used as input. This enables the mixture model to either use several similar prediction models only differing in the encoding of the input data such as the LSTM-based models in chapter 6 or completely different models types such as one simple linear predictor and one simple single-layer neural network. One of the advantages of such an approach is that instead of starting the model from a completely blank state, the individual predictors used as inputs for the mixture model already learned a consistent prior during their offline training. Furthermore, the possibility of the offline models being validated in advance and serving as a fallback option in case the online model fails during deployment, is an additional advantage in a safety-critical domain such as automated driving. Finally, the implementation of our approach employing the classic delta rule as well as the possibility to use spiking neuron models allows future deployment on dedicated neuromorphic hardware, which offers interesting possibilities regarding energy efficiency, especially in mobile applications and automotive context. We investigate two variants of our online learning system differing in the information the model uses to optimize its weights. In its simplest form, such a model adjusts its weights only based on the prediction error, i.e., the difference between its own prediction and the actual motion of the target vehicle. However, we have seen in section 6.3.1 that the performance of the individual models heavily depend on the current driving situation. Therefore, we also investigate online learning models, that adjust their predictions based on some sort of contextual information. We use the findings from section 6.3.1 as a first hint regarding potential context information.

However, any online learning system making predictions about the future poses additional challenges. For instance, the actual motion of the target vehicle and thus the error signal to update the neural weights is unknown at prediction time, but rather becomes available while the agent continues driving. The temporally delayed error signal potentially introduces long lags between the prediction and the update of the corresponding weights. Therefore, we need to find a mechanism to propagate the error between the actual future motion of the target vehicle and the model's prediction back in time to update the weights correctly. In this chapter, we address this issue through a temporal spreading of the error signal, i.e., we use the error of earlier prediction steps to update the weights of predictions further into the future. Furthermore, we need to deal with the problem of over-fitting meaning that the model could just learn to predict the current vehicle particularly well but when switching to another vehicle with a potentially very different situational context and/or dynamics, the model's performance could deteriorate due to

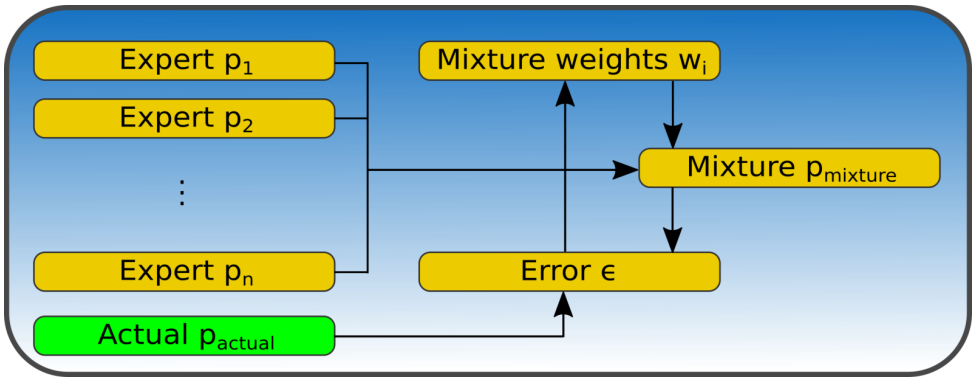


Figure 7.1: Visualization of the network architecture of the context-free mixture-of-experts online learning system with yellow boxes indicating the individual components of the model. The weights of the mixture-of-experts model depend solely on the error ϵ between the model's prediction and the actual future motion of the target vehicle.

over-fitting the previous vehicle. We evaluate our online learning model on real-world driving data and show, that the model is able to improve over the individual offline models already after being trained with just a few vehicles. Finally, while the type of online learning approach presented here has been shown to be successful in adaptive robot arm control (DeWolf et al., 2016), to the best of our knowledge, our approach is the first trajectory prediction model to be trained during deployment for choosing from several pre-trained prediction models to achieve the best possible forecast.

7.1 Mixture-of-experts online learning models

In this section, we describe the architecture and learning rules of our proposed mixture-of-experts online learning models. Importantly, all of the mixture-of-experts models shown in this chapter are meant to be trained *online*. That is, we do not pre-train them on a large corpus of data and then fix the final weights. Instead, we run the neural network training *while the system is running*. We start with two variants of the model: one adapting its weights based solely on the prediction error and a more sophisticated model, which uses additional contextual information to adapt its neural weights.

7.1.1 A context-free mixture-of-experts online learning model

Given that we have multiple models p_i for $i = 1,\ldots,n$ predicting vehicle positions, we define the set of prediction models as

$$\mathscr{P} = \{p_i \quad | \quad i = 1,\ldots,n\}. \tag{7.1}$$

We construct our mixture-of-experts model in the following way: We combine the predictions of the offline models using a simple weighted sum and we use online training to learn these weights. This core weighting algorithm is given by

$$\mathbf{v}_{mix,t} = \sum_{p\in\mathscr{P}} \mathbf{W}_{p,t}\mathbf{v}_{p,t}, \tag{7.2}$$

where $\mathbf{v}_{p,t}$ is the predicted value for time t into the future from offline model p, and $\mathbf{W}_{p,t}$ is the weight for expert p for a prediction time of t. Note that this means that the weighting between the expert predictions may be different depending on how far into the future we are predicting. That is, in some conditions one expert should be weighted more highly than in other conditions, and we want to adapt this weighting based on experience. Figure 7.1 depicts the architecture of the context-free mixture-of-experts model variant with yellow boxes indicating the individual components of the model.

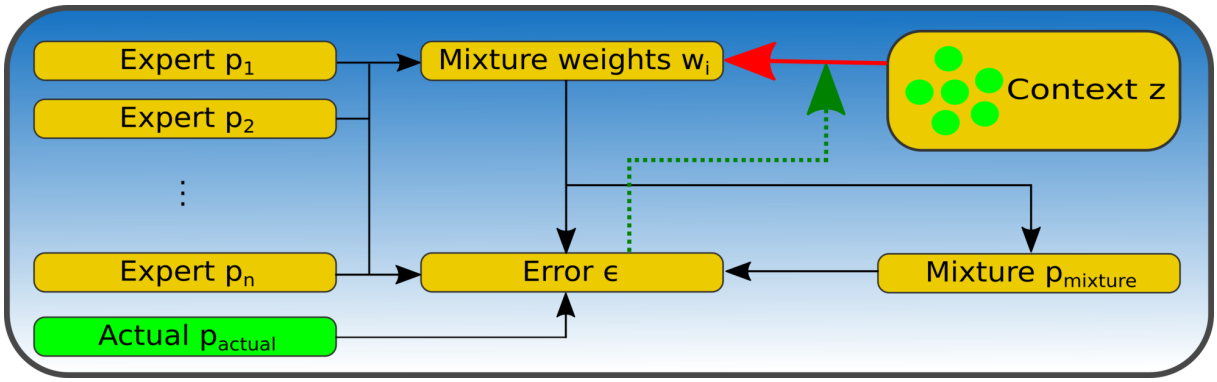


Figure 7.2: Visualization of the network architecture of the context-sensitive mixture-of-experts online learning system. Yellow boxes indicate the individual components of the model, while the solid red line depicts the connection to decode out the weights $\mathbf{W}_{p,t}$ for the individual expert predictors from the neural population encoding the context $\mathbf{z}$ as indicated by the green circles in the context component. The dotted green arrow indicates that the error signal is used to update the weights of this connection using delta rule learning.

The weights $\mathbf{W}_{p,t}$ are initialized equally over all predictors p, i.e., $\mathbf{W}_{p,t} = 1/N_p$, where N_p is the number of prediction models being combined, and then updated using online learning. While any neural network learning algorithm could be used to do this, we adapt the classic delta learning rule, which is the basis of all gradient-descent learning algorithms, for the sake of simplicity and ease of implementation:

$$\Delta \mathbf{W}_{p,t} = \kappa \nu_t \mathbf{v}_{p,t} \underbrace{(\mathbf{v}_{observed,t} - \mathbf{v}_{mix,t})}_{=\varepsilon_t} = \kappa \nu_t \mathbf{v}_{p,t} \varepsilon_t, \tag{7.3}$$

where κ is the learning rate and ν_t is a factor to scale the learning rate κ differently for each prediction time step.

7.1.2 A context-sensitive mixture-of-experts online learning model

The above model attempts to find the best weighting of the individual prediction models based solely on the prediction error of the mixture-of-experts model. However, we believe that the ideal weights will crucially depend on some aspects of the current situation (i.e., the current context). That is, instead of learning $\mathbf{W}$, we can learn the function $f_{\mathbf{W}}(\mathbf{z}) = \mathbf{W}$, where $\mathbf{z}$ is some currently available sensor information. Indeed, the context-free version shown in section 7.1.1 is equivalent to the context-sensitive models shown here if the context is kept constant.

Now our goal is to use this context information $\mathbf{z}$ to generate $\mathbf{W}$. While any neural network learning algorithm could be used here, for simplicity we use a simple single hidden-layer neural network and adapt its weights by applying the delta rule again. That is, we input the context values $\mathbf{z}$ into N neurons (encoding), and the output of the network (decoding) will be the $\mathbf{W}$ values for the current context. For this encoding and decoding, we use the principles of the NEF (Neural Engineering Framework) as described in section 3.3. The encoding process (the input weights for the neural network) is shown in Equation (7.4), converting $\mathbf{z}$ into the activity $\mathbf{a}_i$ of the i-th neuron:

$$\mathbf{a}_i = \mathbf{G}\left(\sum_j \mathbf{e}_{i,j}\mathbf{z}_j + \beta_i\right) \tag{7.4}$$

$\mathbf{G}$ is the neuron non-linearity (here we use the rate-approximation of the Leaky Integrate and Fire neuron, although any other neuron model is likely to have similar behavior). $\mathbf{e}_{i,j}$ and β_i are randomly generated to produce a uniformly distributed range of maximum firing rates and intercepts (the $\mathbf{z}$ value at which the neuron starts firing), as per Eliasmith (2013). This has been shown to be a good distribution of values

for a wide variety of situations, and is consistent with what is observed in mammalian brains (Eliasmith, 2013). Unlike most neural network models, since these input weights are a reasonable distribution, we do not change $\mathbf{e}_{i,j}$ and β_i at any time, leaving them at their initial randomly generated values. In other words, we only adjust the weights between the hidden layer and the output layer, and leave the other set of weights at their initial randomly generated values. This greatly reduces the computation cost of performing the online learning.
Given the neural activity $\mathbf{a}_i$ generated in Equation (7.4), we now use

$$\mathbf{W}_{p,t} = \sum_i \mathbf{d}_{p,t,i}\mathbf{a}_i \tag{7.5}$$

to decode the $\mathbf{W}$ values for the current context $\mathbf{z}$. As with the context-free version of the model, we initialize the weights, i.e., here the $\mathbf{d}$ values such that there is an equal weighting across all the expert predictions, i.e., $\mathbf{W}_{p,t} = 1/N_p$, where N_p is the number of prediction models being combined, for all context $\mathbf{z}$. The $\mathbf{d}$ values achieving this equal weighting are found using least-squares minimization.
Now that we have this system for generating context-dependent weights, we can use online learning to adjust $\mathbf{d}$ to change the weights $\mathbf{W}$ based on the accuracy of the predictions. As with the context-free model variant, we use the delta learning rule given in Equation (7.3)

$$\Delta\mathbf{W}_{p,t} = \kappa\nu_t\mathbf{v}_{p,t}\underbrace{(\mathbf{v}_{observed,t} - \mathbf{v}_{mix,t})}_{=\varepsilon_t} = \kappa\nu_t\mathbf{v}_{p,t}\varepsilon_t. \tag{(7.3) revisited}$$

that would determine how much to adjust $\mathbf{W}$ given the current error between the mixture-of-experts prediction and the observed actual position of the vehicle (i.e., the error in $\mathbf{v}$). κ is the learning rate and ν_t is a factor to scale the learning rate κ differently for each prediction time step. However, rather than applying that change to $\mathbf{W}$ directly, we instead use that as the error signal for the delta rule applied to the decoding network, turning an adjustment to $\mathbf{W}$ into an adjustment to $\mathbf{d}$:

$$\Delta\mathbf{d}_{p,t,i} = \kappa\mathbf{a}_i\nu_t\mathbf{v}_{p,t}\varepsilon_t = \mathbf{a}_i\Delta\mathbf{W}_{p,t} \tag{7.6}$$

Figure 7.2 shows a schematic visualization of our model's architecture. Yellow boxes indicate the individual components of the model, while the solid red line depicts the connection to decode out the weights $\mathbf{W}_{p,t}$ for the individual expert predictors from the neural population encoding the context $\mathbf{z}$ as indicated by the green circles in the context component. Finally, the dotted green arrow indicates that the error signal is used to update the weights of this connection using delta rule learning.

7.1.3 Temporal spreading of the error signal

One extremely important consideration for any predictive model updating its weights through online learning (i.e., one where it is generating anticipated future observations) is that the error signal is only available in the future. That is, we can only apply Equation (7.6) after the amount of time t has occurred. This introduces a long lag into the learning process. We illustrate this issue using an example situation showing the predictions of the individual offline models depicted in Fig. 7.3. In this example, all individual models predict the target vehicle's motion in y-direction almost perfectly until a prediction horizon of roughly 2.5 s when the predictions start to deviate from the actual motion. Assuming a similar situation for a model employing an online learning approach, the weights for the current prediction 2.5 s into the future can only be updated after 2.5 s have passed. For all prediction time-steps further into the future, we have to wait even longer while the error between the prediction and the actual motion potentially increases even more. In the meantime, the model is doomed to make predictions for these future time-steps based on sub-optimal weights based on past learning updates. However, the example depicted in Fig. 7.3 also hints, that the error at 2.5 s could already be used to update weights further into the future as, although the error increases, the general *direction* of the deviation between predictions and actual motion remains the same. In other words, we assume that if our system is currently predicting too large

a value at time t, then it is likely also to be predicting too large a value at time $\tilde{t} > t$. That is, whenever we have an observation at time t that we compare with the prediction made t time steps ago, we can also apply Equation (7.6) for all the larger values as well, i.e., we apply Equation (7.6) for all $\tilde{t}$ with $\tilde{t} > t$ once the amount of time t has passed. Since this is just an estimation, we want the predictions for time steps $\tilde{t}$ further into the future than t be less influenced by the error at t. Hence, we exponentially scale down the amount of adjustment of $\mathbf{d}$ by the difference in time $\tilde{t} - t$, leading to our final learning rule:

$$\Delta\mathbf{d}_{p,\tilde{t},i} = \kappa \mathbf{a}_i v_t \mathbf{v}_{p,t}(\mathbf{v}_{observed,t} - \mathbf{v}_{mix,t}) e^{-(\tilde{t}-t)/\tau} \qquad \text{for all } \tilde{t} \geq t. \tag{7.7}$$

7.2 Experiments and results

In this chapter, we use the output of three different prediction models as input for our mixture-of-experts online learning model. We use some of the models we developed and already evaluated in chapter 6 as input to the mixture-of-experts model. These individual predictors are a simple *linear* prediction model based on a constant velocity assumption and two LSTM-based neural networks, namely LSTM SPA 1 or 3 (depending on the data set) and LSTM numerical. Those are the models presented in section 6.2.2 and evaluated in section 6.3.1. The architecture of the LSTM models, which was already depicted in Fig. 6.7, consist of one encoder and one decoder cell each for sequence to sequence prediction. The LSTM SPA 1 and 3 models used as input to the mixture-of-experts online learning model employ the convolutive power encoding of the input data as shown in Equation (6.1), while the LSTM numerical model simply uses raw numerical values as input data (see also section 6.2.2).
For training the model, we simulate online deployment by presenting the LSTM models' predictions on the validation subsets to the system. Thereby, the individual experts perform their prediction on previously unseen data samples. We conduct individual simulation runs for each of both data sets.
We present results from two variants of our mixture-of-experts model. Firstly, we evaluate a simplified version, which applies Equation (7.6) directly at prediction time assuming that the error signal, which is future data, is actually available already at prediction time. The benefit of this prior evaluation is two-fold: on the one hand, we get an impression what benefits can be expected from using context information over the context-free variant before employing the more sophisticated, timing-sensitive model. On the other hand, a model having immediate access to the future error signal serves as an upper bound for the performance to be expected from models that have to deal with temporally delayed error signals. We present the results of this prior evaluation in section 7.2.2, whereas the evaluation of the model tackling temporally delayed error signals is shown in section 7.2.3.
All model variants in this chapter are implemented using the neural simulator Nengo (Bekolay et al., 2014), which is typically used for constructing large-scale biologically realistic neural models (Eliasmith, 2013), but also allows for traditional feed-forward artificial neural networks using either spiking or non-spiking neurons. Here, we use the rate-approximation of the Leaky Integrate and Fire neuron, although we expect any other neuron model to have similar behavior. Spiking neurons are of considerable interest for automotive applications due to the potential for reduced power consumption when deployed on dedicated neuromorphic hardware.

7.2.1 Data and preprocessing

In this section, we describe the data we use to train and evaluate our mixture-of-experts models. We use the same data sets already used in chapter 6 and detailed in section 6.1. Figure 7.3 depicts one particular data sample from the *On-board* data set showing a lane change maneuver of the target vehicle. The upper row shows the raw images from the ego-vehicle's front and rear camera to get an idea of the driving situation with the target vehicle highlighted by a red box. The middle row of plots shows the corresponding trajectory data for that particular driving situation, i.e., the positions of the vehicles during the past 5 s, the actual future motion of the vehicles as well as the offline model's predictions of the target

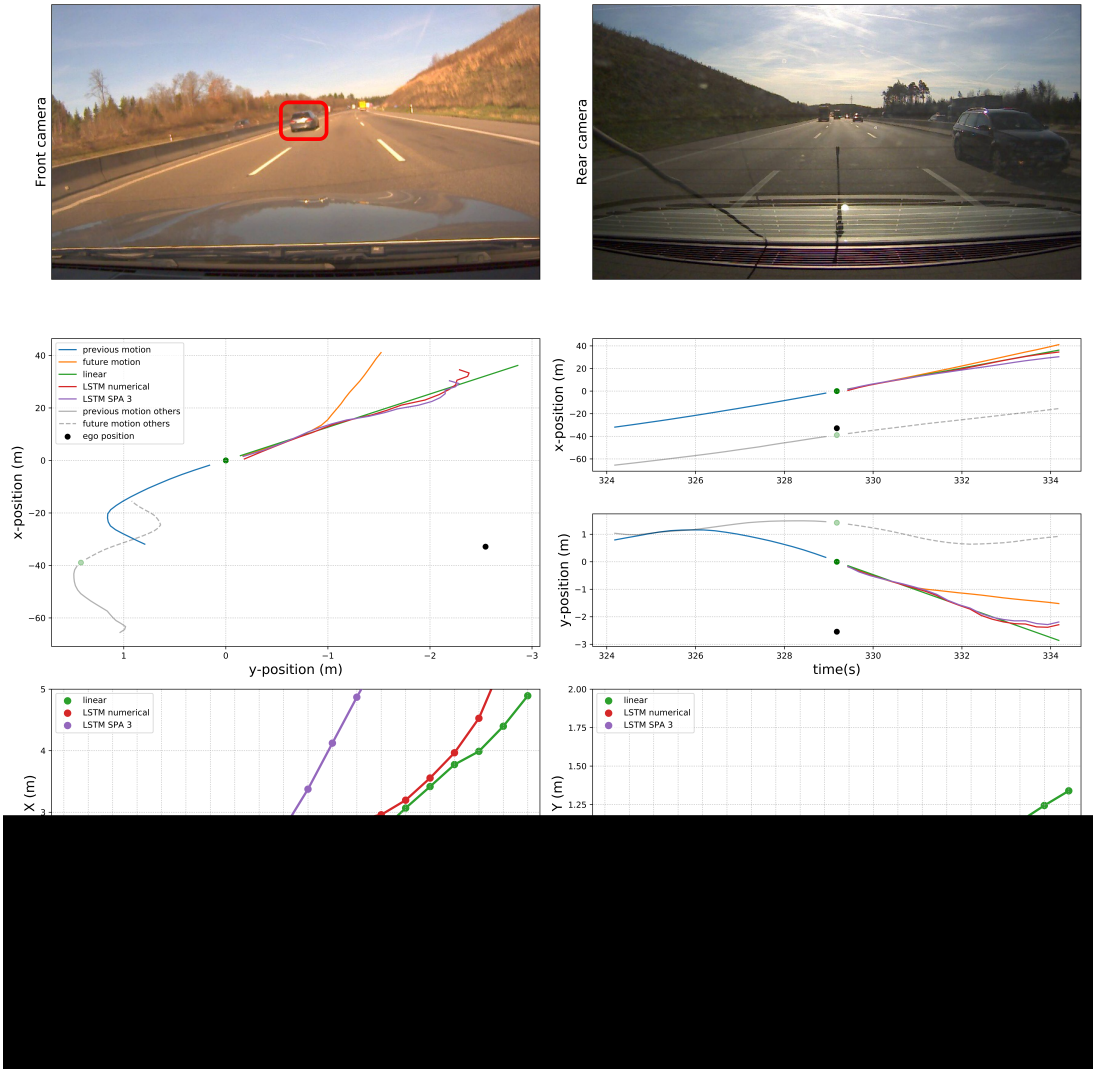


Figure 7.3: One example from the *On-board* data set depicting a particular driving situation as well as the predictions made by each individual offline model used as input for our mixture-of-experts model. The upper row shows images of the ego-vehicle's on-board cameras. The middle row shows the trajectory data where the dots indicate the position of the vehicles and color-code the vehicle type (red=motorcycle, green=car, blue=truck, black=ego-vehicle), the solid blue and orange line show past and future motion of the target vehicle, the other colored lines visualize the predictions of the offline models whereas gray lines depict the other vehicles' motion. Finally, the lower row shows the absolute error between each offline models' prediction and the actual positions of the target vehicle.

vehicle's future motion. The lower row finally shows the error between each model's prediction and the target vehicle's actual future motion for longitudinal (x) and lateral (y) direction separately.

Input to the mixture model

Independent of what type of prediction model shown in chapter 6 we use as input for the online learning system, the output of each individual predictor is anticipated positions in x- and y-direction at 20 equidistant time steps t_i for $i = 1,\ldots,20$ for 5 s into the future, i.e. $t_1 = 0.25, t_2 = 0.5, \ldots, t_{20} = 5.0$. Figure 7.3 shows one data sample from the *On-board* data set depicting the input data to the offline models (previous motion) as well as each model's individual predictions, which are the input to the mixture model.

In other words, the colored lines in Fig. 7.3 depicting the predictions of the linear, LSTM SPA 3 and LSTM numerical models form the input of the mixture-of-experts online learning model. Note that these anticipated positions are each individual model's guesses about the actual position values used as ground truth in chapter 6. We use these prediction values unaltered and without further preprocessing as input for our mixture models. Furthermore, we only present vehicles from the test sets V_1 and V_2 to our mixture models to avoid presenting vehicles to the system the individual predictors have already been trained on.

Contextual information used in the mixture model

In contrast to the anticipated positions predicted by each offline model we use as input to our mixture-of-experts model as described in the previous section, here we present the information used to describe the current driving situation, i.e., the contextual information encoded in the context population of our model. While the context information used for the context-sensitive model variants could be described in many different ways, we use the following three pieces of information:

- $\mathbf{z}_1$: the current distance from the target to the ego-vehicle if available (note that there is no ego-vehicle the NGSIM data set. See section 6.1.2)
- $\mathbf{z}_2$: the current distance from the target to the nearest other car (including the ego-vehicle if available)
- $\mathbf{z}_3$: the number of cars currently visible within a certain distance to the target vehicle

We use the results from chapter 6 as hint, since these pieces of information showed significant alterations depending on which prediction model performs best on the corresponding samples. To translate these context values to a common order of magnitude, we use the training data to normalize these values and use their z-scores.

7.2.2 Comparing timing-agnostic context-free and context-sensitive mixture models

In this section, we evaluate a simplified version of our mixture-of-experts online learning model, which ignores the fact that the actual vehicle motion and thus the error signal for the weight updates is not available at prediction time. Instead, we assume that Equation (7.6) can be applied directly at prediction time to update the weights without having to wait for the actual data to become available. Thereby, we get an impression of the benefits that can be expected, if any, from using context information over the context-free variant before employing the more sophisticated, timing-sensitive model. Furthermore, a model having immediate access to the future error signal serves as an upper bound for the performance to be expected from models that have to deal with temporally delayed error signals (see Section 7.2.3).

Experimental setup

Working with prerecorded data allows us to easily evaluate this simplified model before switching to the more complex version employing temporal spreading of the error signal (cf. 7.1.3). The context-free version updates its weights solely based on the prediction error, i.e., the error between the anticipation values predicted by the model and the actual motion of the target vehicle. This context-free approach is equivalent to the context-sensitive model if its context is kept constant. The context-sensitive model variant as described in section 7.1.1, which we investigate here, contains 3000 neurons in the population encoding the driving context. To limit the simulation time, we exposed only a subset of 70 vehicles from the *On-board* test set V_1 and 92 vehicles from *NGSIM* the test set V_2 to the models.

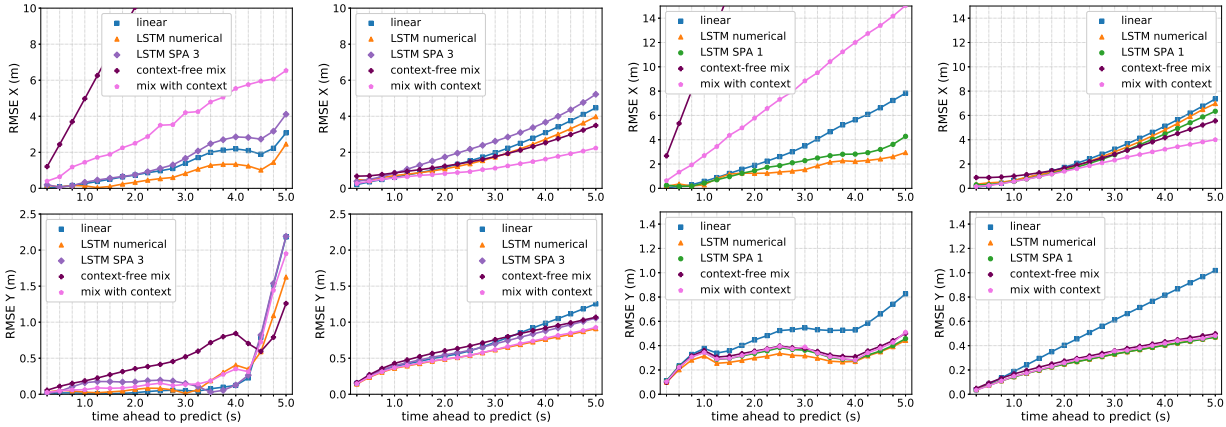


Figure 7.4: Visualization of the RMSE of the timing-agnostic variants of our mixture-of-experts model, both context-free and context-sensitive, on both data sets. (a) shows the performance at the start of the training process on the *On-board* data set. (c) shows the performance at the start of the training process on the *NGSIM* data set. Similarly, (b) shows the models' performance on the first 70 vehicles of the *on-board* data set, whereas Fig. (d) show the models' performance on the first 92 vehicles of the *NGSIM* data set.

Results

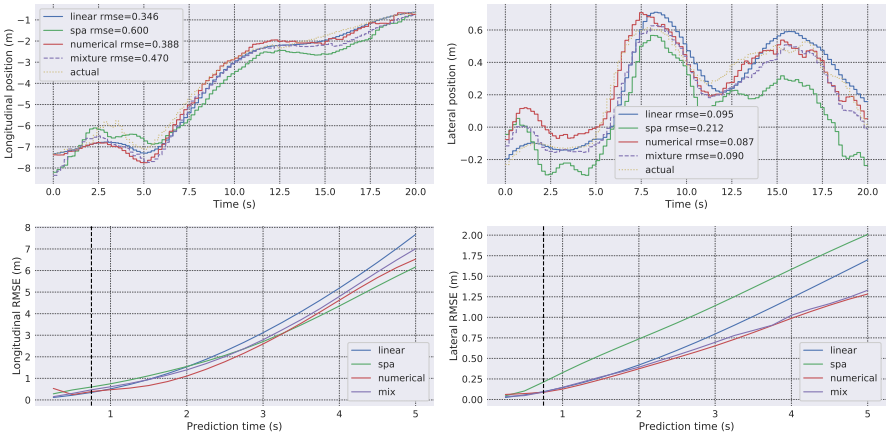
Figure 7.4 shows the results of the simplified, timing-agnostic variants of the context-free and context-sensitive mixture-of-experts online learning models on both data sets. Figure 7.4a and 7.4b show the models' performance on the *On-board* data set at the start of training (Fig. 7.4a) and for the complete set of 70 evaluation vehicles from V_1 (Fig. 7.4b), whereas Fig. 7.4c and Fig. 7.4d similarly depict the models' performance on the *NGSIM* data set at the start of training and for all 92 evaluation vehicles respectively. We observe in Fig. 7.4a and Fig. 7.4c that both, the context-free and context-sensitive mixture models perform poorly at the start of training (which makes sense due to the randomly initialized weights), but improve significantly and consistently on both data sets (Fig. 7.4b and Fig. 7.4d) the more data they receive. The context-free version yields mild improvements over all individual predictors in x-direction without improving over the best individual model in y-direction. The context-sensitive variant outperforms all other models (including the context-free version) in x-direction while being on par with the best individual predictors in y-direction.

7.2.3 Evaluation of the context-sensitive model variant with temporal spreading

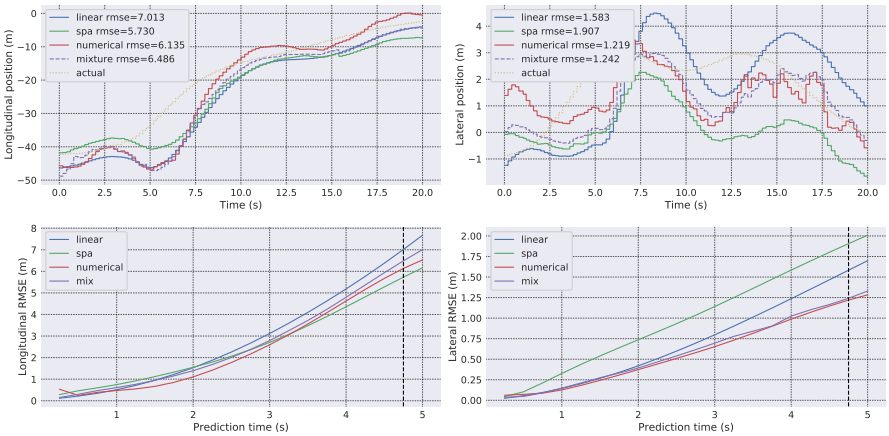
Having seen that the simplified mixture model using contextual information clearly outperformed the context-free version, we now proceed to the evaluation of the context-sensitive mixture-of-experts model variant having to deal with temporally delayed error signals employing our temporal spreading approach. We describe the experimental setup, analyze several adjustable parameters of the model and finally evaluate the performance of the model given the chosen parameter setup.

Experimental setup

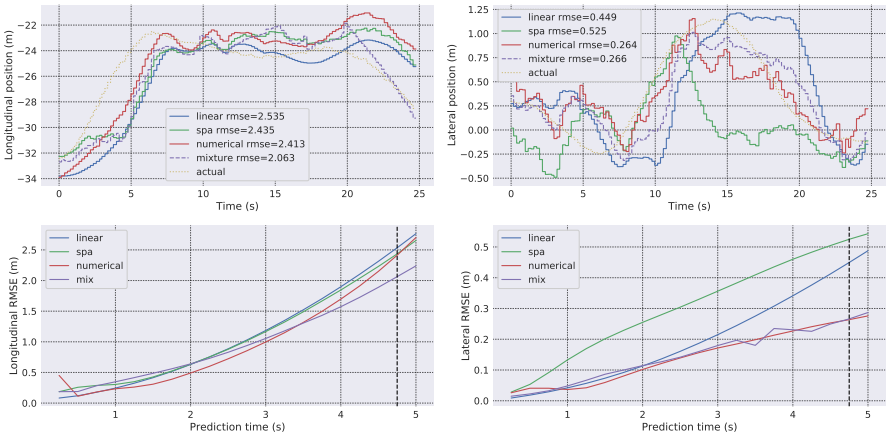
To evaluate feasibility and performance of our mixture-of-experts model having to deal with temporally delayed error signals, we conducted our experiments for both, the parameter analysis as well as the actual performance evaluation, in the following way: after randomly initializing the weights of the mixture model, we present the data of 5 randomly chosen vehicles to the model to make sure the weights depart reasonably from their initial values. After this ramp-up phase, during which we conduct no evaluation,



(a) Predictions for the first training vehicle 0.75 s into the future



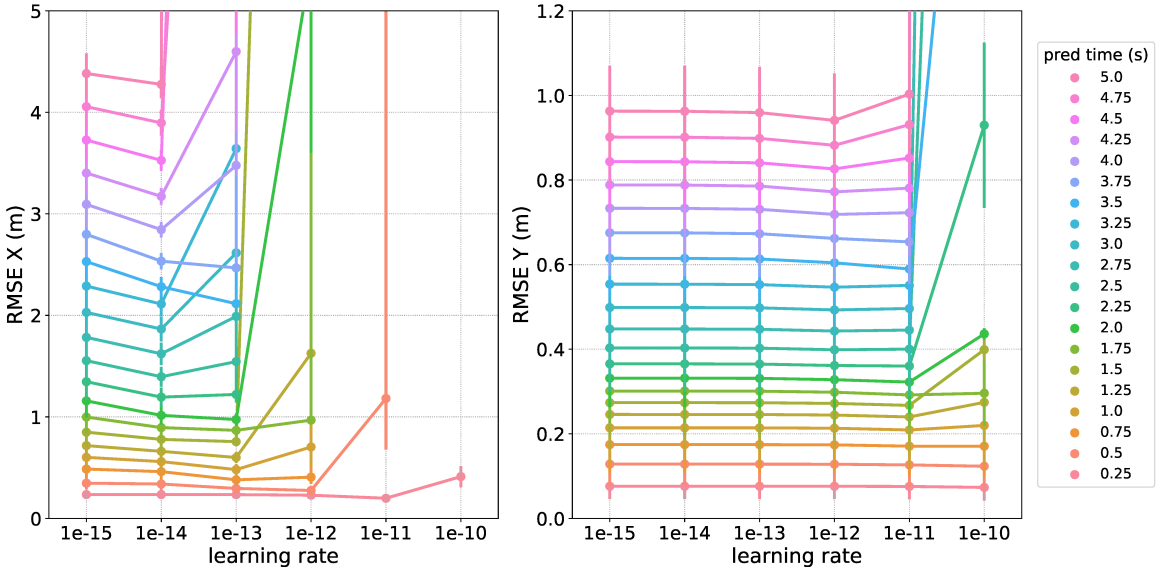
(b) Predictions for the first training vehicle 4.75 s into the future



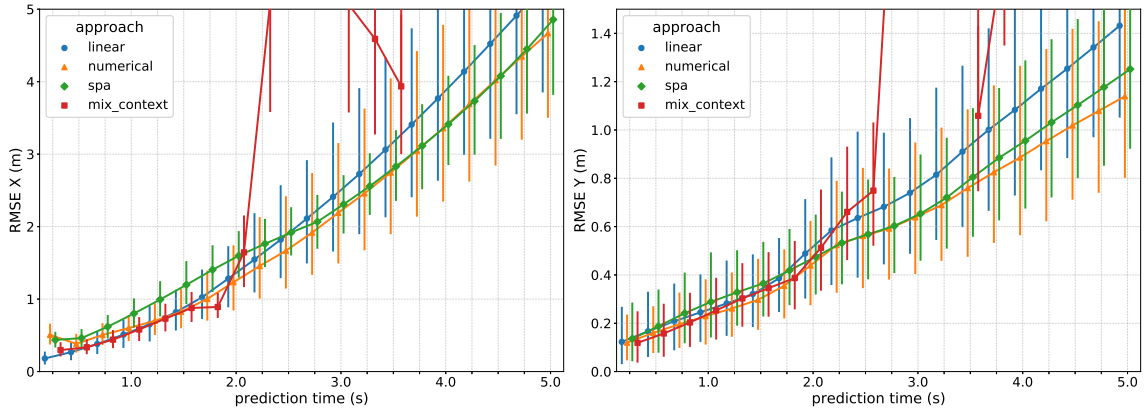
(c) Predictions for the fifth training vehicle 4.75 s into the future

Figure 7.5: Visualization of the mixture-of-experts model's performance on vehicles it is presented during the ramp up phase. The upper row in each plot shows the predictions of all input predictors, the mixture model as well as the actual motion of the target vehicle for one particular prediction time step. The lower row illustrates the RMSE of the models on all prediction time steps with that step shown in the upper row highlighted by a dotted vertical line.

the model is presented with 30 more randomly chosen vehicles to be evaluated on. During the presentation of these 30 test vehicles, the mixture model continues to adapt its weights depending on the current



(a) RMSE performance for varying learning rates for a fixed number of 50 neurons in the context population

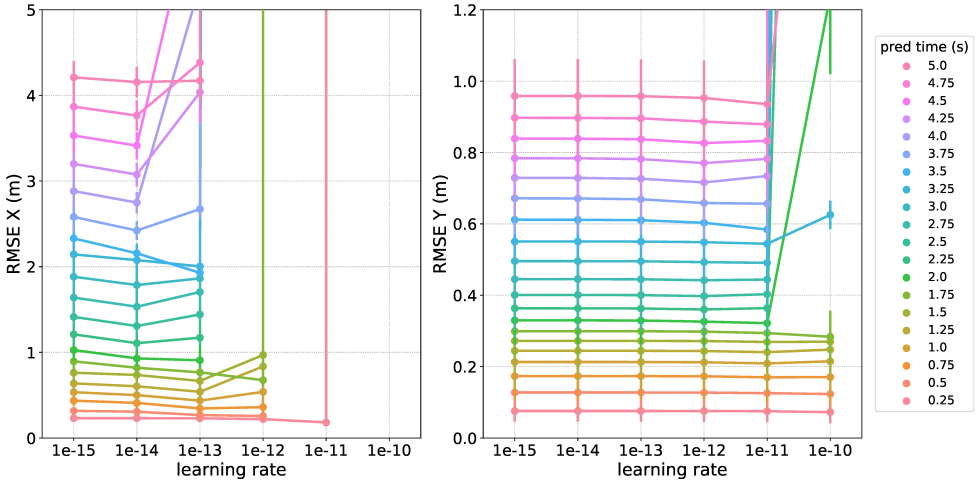


(b) Results of an experiment with too low of a learning rate for the mixture models resulting in high errors

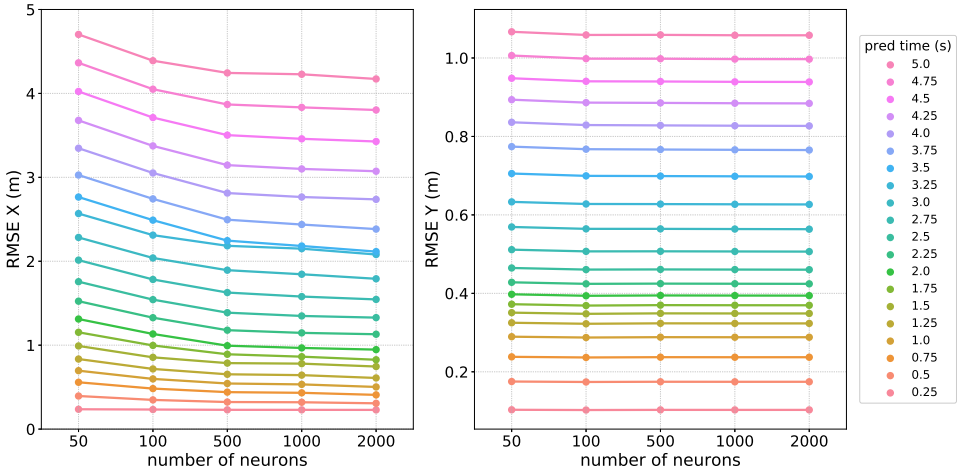
Figure 7.6: Visualization of the RMSE performance of our context-sensitive mixture-of-experts model using temporal spreading on 30 test vehicles after being trained on 5 vehicles for (a) different learning rates for all prediction horizons and (b) the result of one evaluation run with too low of a learning rate.

error and context.

Figure 7.5 shows the performance of the mixture model with 50 neurons in the context population for selected vehicles during the 5-vehicle ramp up phase. The upper row in each plot shows the predictions of all input predictors, the mixture model as well as the actual motion of the target vehicle for one particular prediction time step. For instance, the upper row in Fig. 7.5a illustrates the predictions of all models for 0.75 s into the future over the complete time the target vehicle is visible. The lower row in each plot illustrates the RMSE of the models for all prediction time steps with the particular time step shown in the upper row highlighted by a dotted vertical line. Figures 7.5a and 7.5b show the prediction time steps 0.75 s and 4.75 s into the future respectively of the first vehicle during the ramp-up phase while Fig. 7.5c shows the prediction time step 4.75 s into the future for the last vehicle of the ramp-up phase. We observe that the evenly distributed weights are a reasonable initialization for further adaptation by our model, since already for the first vehicle presented to the model achieves a decent prediction accuracy. Reaching the fifth training vehicle, the mixture model already achieves improvements for prediction time-steps further into the future in x-direction and equal to the best performing input predictor in y-direction.



(a) RMSE performance for varying learning rates for a fixed number of 100 neurons in the context population



(b) RMSE performance for varying number of neurons in the context population for a fixed learning rate of $\kappa = 10^{-15}$

Figure 7.7: Visualization of the RMSE performance of our context-sensitive mixture-of-experts model using temporal spreading on 30 test vehicles after being trained on 5 vehicles for certain hyper-parameter variations.

Parameter analysis

To find the best possible variant of our mixture-of-experts model, we first analyzed the adjustable parameters of the model. Figure 7.6 shows a visualization of the RMSE performance of our context-sensitive mixture-of-experts model using temporal spreading on 30 test vehicles after being trained on 5 vehicles. Figure 7.6a visualizes the RMSE performance of our mixture-model with 50 neurons in the context population for different values of the learning rate κ with scaling factors $\nu_t = 1$ for all prediction time steps t. In this evaluation, a learning rate of 10^{-14} in x-direction and 10^{-12} show the best performance. However, Fig. 7.6b shows the results of a model variant with 2000 neurons in the context population employing these learning rates for a different set of vehicles. We observe that the model performs well for earlier prediction horizons but fails to make accurate predictions further into the future with large errors and even complete failure due to overfitting.

Therefore, we focus on scaling factors ν_t decreasing linearly from 1 to 0.001 over the 20 prediction times with 1 corresponding to the earliest prediction step to ensure low enough learning rates prediction time steps further into the future. Figure 7.7 shows a visualization of the RMSE performance of our

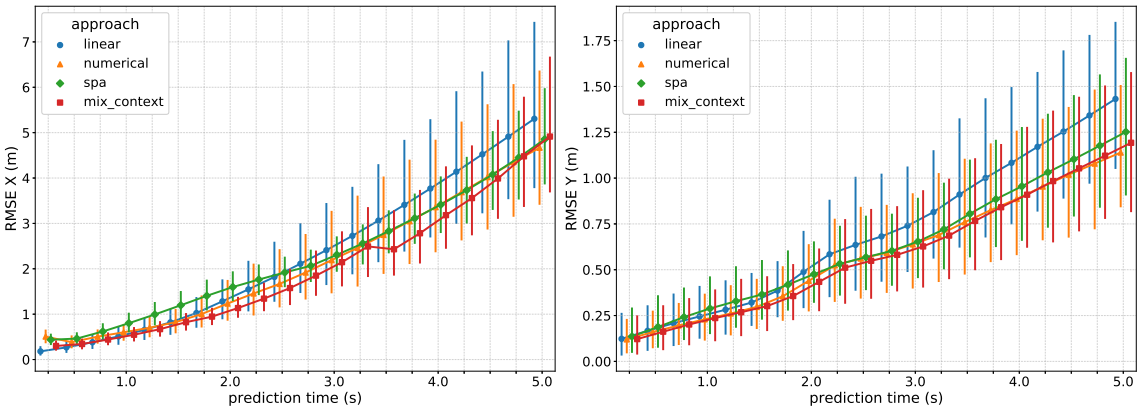


Figure 7.8: Visualization of the RMSE performance of all individual expert predictors as well as our context-sensitive and temporal spreading mixture-of-experts model with 2000 neurons in the context population, evaluated on 30 vehicles after the mixture model has been trained on 5 vehicles in advance to make the weights depart reasonably from their random initialization.

mixture-model with 100 neurons in the context population with for different values of the learning rate κ. Figure 7.7a shows an analysis similar to Fig. 7.6a but using the linearly decreasing scaling factors. We observe that the scaling the learning rates down stabilizes the model compared to the even scaling factors. As already shown in Fig. 7.6b, we found that a rather low learning rate κ is necessary as higher learning rates tend to let the model overfit the current vehicle, which leads to large errors and even complete failure when switching from one vehicle to the next. We observe that issue in particular in x-direction when already a learning rate of $\kappa = 10^{-13}$ leads to large errors and even failure. Given the results shown in Fig. 7.6b, we use a learning rate of $\kappa = 10^{-15}$. As the evaluation shown in Fig. 7.7 included only 30 test vehicles, we chose this rather conservative learning rate to make sure our model does not overfit when using a different set of vehicles.

Furthermore, we investigated the influence of the number of neurons in the context ensemble on the model's performance employing the aforementioned values for the learning rate κ and scaling factors ν_t. Figure 7.7b shows the RMSE of the model for varying numbers of neurons in the context population. In y-direction, the influence of the number of neurons is nearly invisible, whereas in x-direction increasing the number of neurons results in lower RMSE values. Thus, we focus our further investigations on models with 2000 neurons in the context population.

Results of the mixture-model using temporal spreading

Figure 7.8 shows the RMSE performance of our context-sensitive mixture-of-experts model employing temporal spreading with 2000 neurons on the 30 test vehicles in comparison to the individual input predictors after the ramp-up phase of 5 vehicles. We observe that our mixture-of-experts model improves over the individual models on average over all 30 test vehicles in both directions without clearly outperforming them.

To further investigate these results, we evaluate the model's performance on individual vehicles. Figure 7.9 shows the RMSE performance of our mixture-of-experts model in comparison to the individual input predictors on a selection of 8 individual test vehicles. For most of the examples, we observe results similar to the overall, mean performance shown in Fig. 7.8 with the mixture model improving over all individual predictors in x-direction and being at least comparable to the best individual predictor in y-direction (cf. Fig. 7.9a, 7.9c, 7.9d, 7.9e). However, there are also vehicles such as the one shown in Fig. 7.9b, where we observe improvements of the mixture model over the individual predictors in y-direction with no improvements shown in x-direction. For the vehicle shown in Fig. 7.9f, the mixture

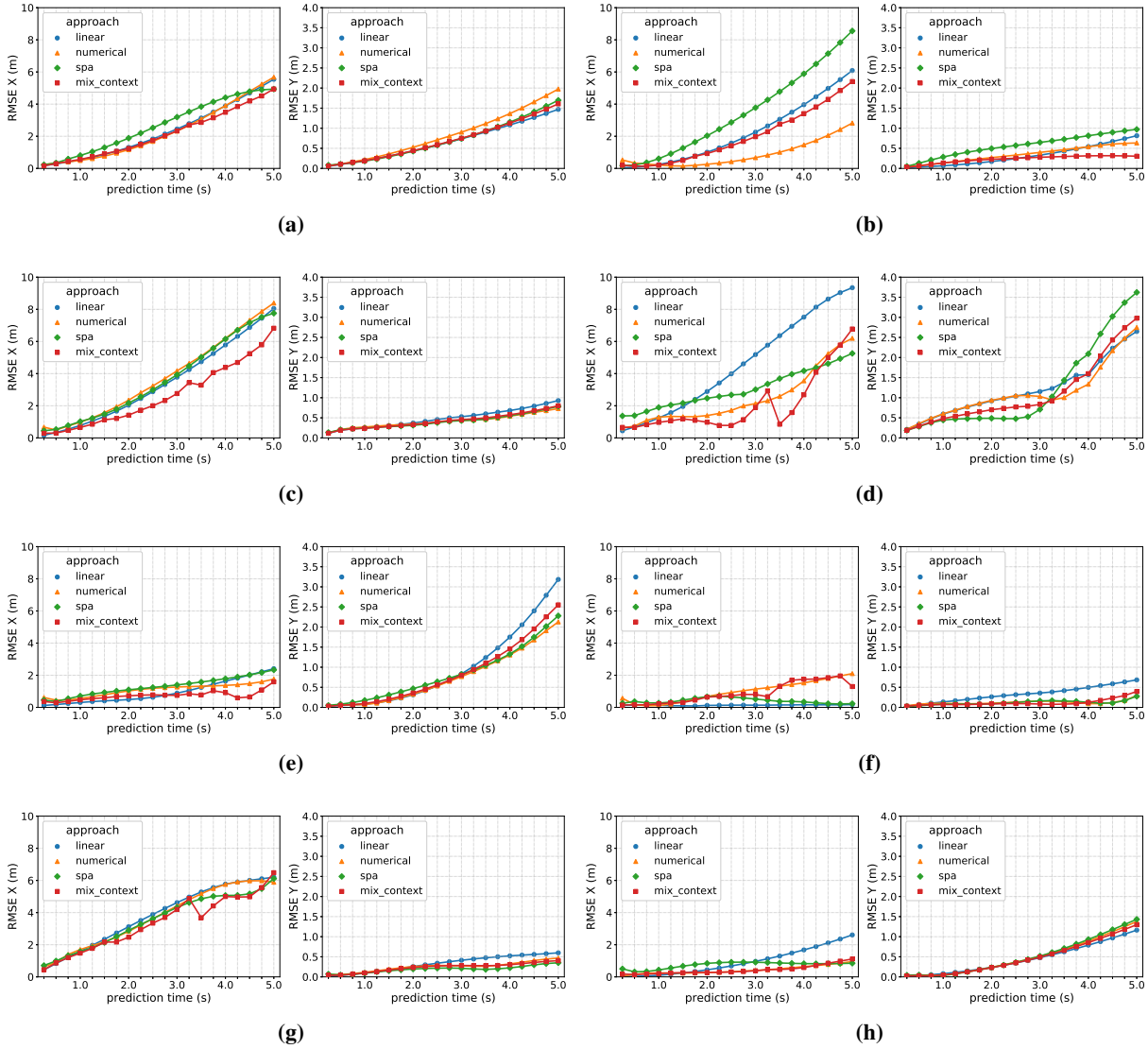


Figure 7.9: Visualization of the RMSE of our mixture-of-experts model on 6 different test vehicles in comparison to the input predictors' performance.

model does not yield any improvements over the input models in either direction. For that particular vehicle however, we also observe for both directions that one of the offline models achieves a remarkably low RMSE performance, namely the linear predictor in x-direction and the LSTM models in y-direction, leaving little room for improvement. However, it would be desirable that our online learning model detects such situations and at least learns to approximate the best available individual offline model. Figure 7.8 shows the RMSE performance of our context-sensitive and temporal spreading mixture-of-experts model with 2000 neurons on the 30 test vehicles in comparison to the individual input predictors.

7.3 Summary

In this chapter, we have extended our work on vehicle trajectory prediction presented in chapter 6. We have introduced a novel mixture-of-experts online learning model implemented in a spiking neuron substrate employing a simple delta learning rule to learn to weight different predictors, which have been previously trained offline, at run time. We presented two variants of this model: one adapting its neural weights simply based on the error between the model's predictions and the target vehicle's actual motion

and a more advanced version anticipating future motion using contextual information in addition to the error signal. An evaluation of a simplified model variant having access to the error signal, which in a realistic scenario lies in the future, at prediction time showed that the model using contextual information clearly outperforms the context-free version. Additionally, this simplified model served as an upper bound for the improvement that can be expected from the model having to deal with temporally delayed error signals. To tackle this issue of delayed error signals, we proposed an approach to spread the error signal of earlier prediction times to later time steps. Our evaluation of this advanced model on real world driving data showed that our approach is able to achieve improvements over the individual offline models already after being presented with only a few example vehicles. However, the performance gain is not as significant as expected from the experiments with the simplified model variant. Therefore, there is room for improvement for future work by, for instance, evaluating how much the model at hand can be further improved by increasing the number of neurons in the context population. Furthermore, we only investigated one option of contextual information to be used for the mixture model to make predictions, which is in line with the findings of chapter 6. However, other information such as the vector representation of the driving scene itself or another more detailed description could be used as context for the mixture model. Additionally, we only trained our mixture model on one vehicle at a time. To allow actual deployment in a driving vehicle, we aim to investigate an advanced version of our approach, which spawns multiple model instantiations to be trained on several vehicles in parallel using shared weights. Finally, we have not investigated what efficiency benefits could be achieved by deploying our model on specialized neuromorphic hardware.

8 Closed-loop neuromorphic control systems

In this chapter, we present a first step towards a neuromorphic control architecture, that can be used to implement generic control algorithms in the language of SNNs (Spiking Neural Networks). This offers the advantage that the overall task can be divided into several sub-networks. We develop two sample instantiations of neurally-inspired control algorithms, namely for mobile robot manipulation and vehicle trajectory control based on reinforcement learning. Here, we partly depart from the application of automated driving using real-world data, which is an intentional choice. As mentioned earlier, control of an automated vehicle is an extremely safety-critical task. Furthermore, current vehicle hardware architectures are designed neither to include neuromorphic hardware nor to efficiently process SNNs. Therefore, we demonstrate the general feasibility of this neuromorphic control architecture on the task of bringing order in a sequence of unordered visual stimuli detected using a neuromorphic vision sensor, the DVS (section 8.1). The second demonstration scenario is closer to automotive context, presenting a system employing reinforcement learning for short-term vehicle trajectory planning evaluated in the simulated environment of TORCS (The Open Racing Car Simulator) (Wymann et al., 2014). Both applications demonstrate certain aspects of the control architecture: the mobile manipulation tasks shows how the principles of neural engineering can be used to manually program non-trivial tasks in a spiking neural substrate. This can be useful, to either bootstrap learning to improve task performance without having to start the process from an initial blank state or to complement learning networks with manually designed networks involving human expert knowledge. The second task presents such a combination in an instantiation of the neuromorphic control architecture, with a trajectory selection module employing reinforcement learning while all other modules are manually designed for simplicity. This also allows to train and validate learning systems solving certain sub-tasks decoupled from the rest of the system.

8.1 Sensorimotor adaptation for mobile robotic manipulation

In this section, we propose a novel neural controller for a mobile manipulator platform that can adapt its control policy for grasping different objects within a visual scene. Although the structure of the task is invariant, the scene changes. Our algorithm is not dedicated to solving a single scene, rather is able to robustly accommodate changes in it without any prior knowledge about the changes. Such behavior is a fundamental aspect in sensorimotor control. We propose a system which uses a spiking neural substrate for representation and computation, that allows the system to approximate sensorimotor correlations for both basic and complex motion and grasping scenarios. Constructing such a system entirely with simulated neurons gives us two unique advantages. First, the resulting system can be run on energy-efficient neuromorphic hardware. Second, we can use the network that we design here as the starting point for learning from experience (as opposed to traditional neural network solutions, which learn from a blank state). Such a processing substrate supports learning and allows the system to adapt during operation to unforeseen changes in the task. However, in order to generate an initial functioning neural network model that could be used to bootstrap learning, we need a way to program such a network using something similar to traditional engineering programming methods. The proposed system describes a unified design approach that links low-level sensorimotor data representation with high-level reasoning using a generic computational substrate.

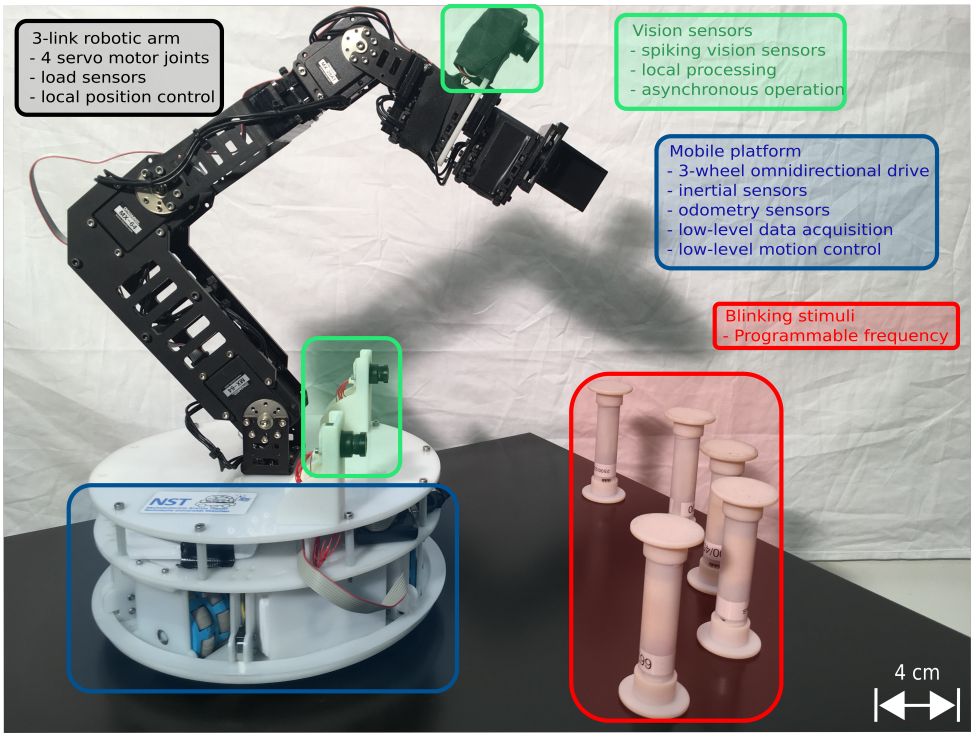


Figure 8.1: Robotic platform

8.1.1 Neuromorphic system architecture

Hardware setup

The mobile manipulator used in this project is comprised of a custom developed omni-directional mobile platform, depicted in Figure 8.1, with embedded low-level motor control and elementary sensory acquisition. The on-board ARM7 micro-controller receives desired motion commands and continuously adapts three PID (Proportional-Integral-Derivative) motor control signals to achieve desired velocities. The robot's integrated sensors include wheel encoders for estimating odometry, a 9 degrees of freedom IMU (Inertial Measurement Unit), a bump-sensor ring, which triggers binary contact switches upon contact with objects in the environment, and three silicon retinas providing visual sensor input. Two of these cameras are fixed on the mobile base, while one retina is attached to the end-effector to monitor the workspace of the robotic arm. The silicon retinas are eDVSs (embedded Dynamic Vision Sensors) and provide discrete events as response to temporal contrast. All 128×128 pixels of the DVS operate asynchronously and illumination changes are signaled within a few microseconds after occurrence (without having to wait for a frame to send information). Such information is communicated through spikes, representing a quantized change of log intensity at a particular location. The mobile platform is equipped with a 6-axis robotic arm with a working space between 10 cm and 41 cm. The robotic arm is composed of a set of links connected together by revolute joints and has a lifting weight force up to 800 g. The mobile platform contains an on-board battery of 12 V @ 30 Ah; thereby providing a total of 360 W, which allows autonomous operation for well above 5 h.

Software setup

The overall software architecture is based on a modular design allowing the extension of the current sensorimotor capabilities of the robotic platform by adding more sensors and actuators. The software architecture is comprised of an embedded sensorimotor platform running on-board the robot and a neurocomputing platform suitable to run on various computing backends like CPU, GPU and NPU (Neuro-

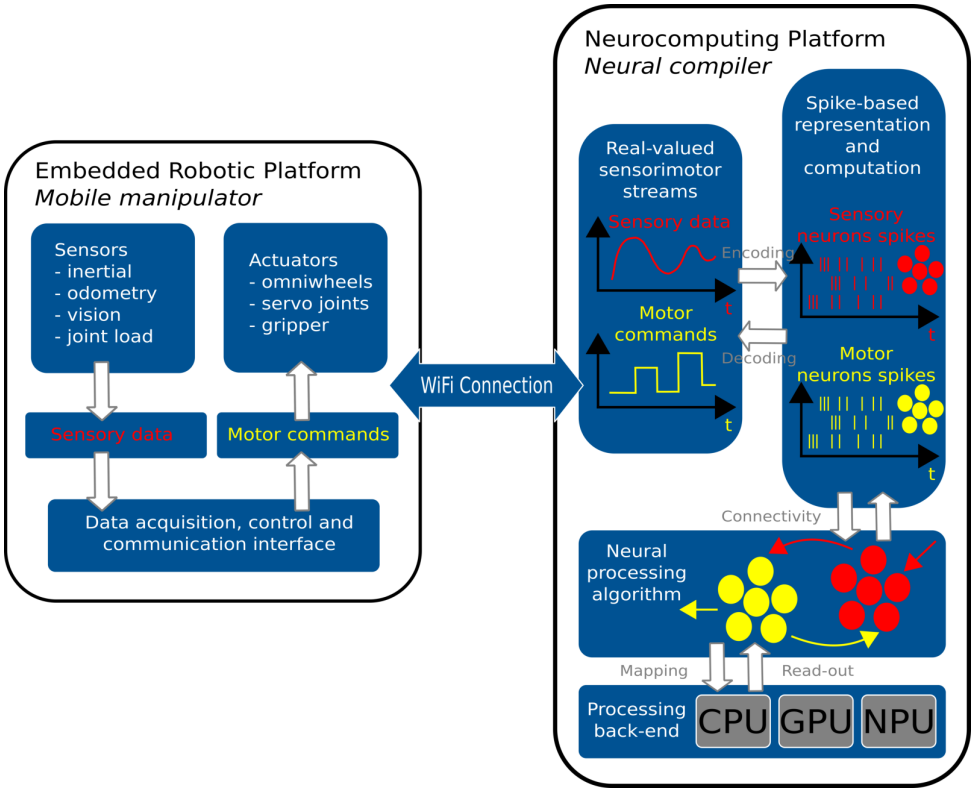


Figure 8.2: Generic architecture: Embedded Robotic Platform and Neurocomputing Platform

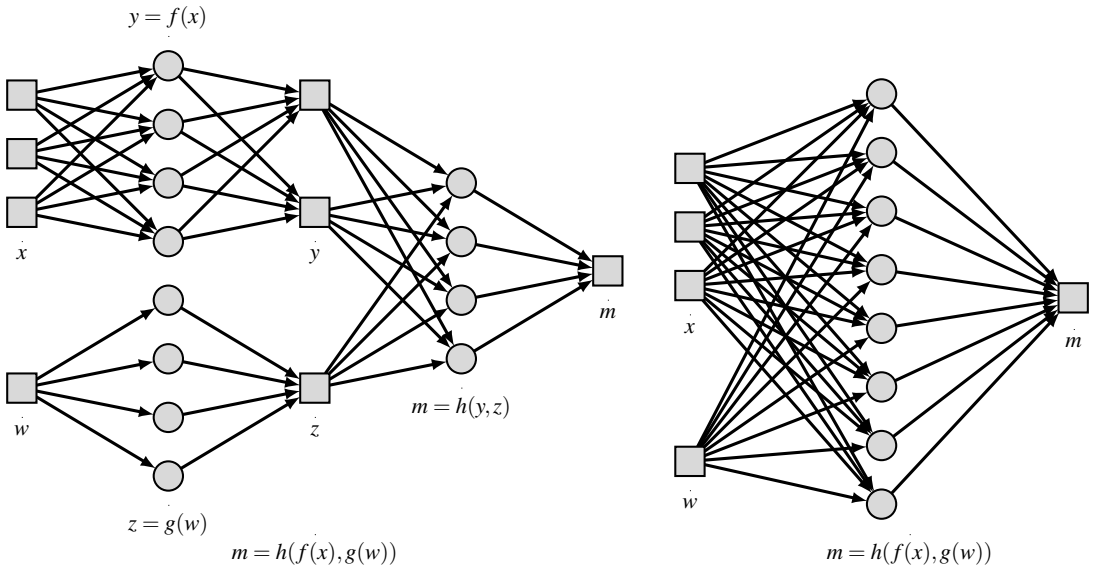
morphic Processing Unit), as shown in Figure 8.2. The embedded platform is responsible for the low-level sensory perception, low-level motor control, and the bi-directional communication (i.e., outgoing data streaming and incoming commands) with the neurocomputing platform. Decoupling low-level sensorimotor control from the high level behavior, the neurocomputing platform offers a generic interface to implement neurocontrol algorithms. This is achieved by separating the representation of the sensorimotor streams, the transformation to be applied on these streams and the actual dynamics of the algorithm. Using such a decoupling and a high-level description of the task, the neurocomputing platform acts as a neural compiler. Such a neural compiler is able to encode real-world sensorimotor streams in spiking activity over populations of neurons. This representation is highly informative and can support efficient computation and learning needed in closed-loop robotic applications, where data uncertainty, noise and unstructured environments yield adaptive behavior. Supporting intrinsically parallelizable processing mechanisms (i.e., neural networks) the neurocomputing platform can accelerate computation by natively mapping the neural controller on parallel computing hardware.

Interface infrastructure

The interface between the embedded platform and the neurocomputing platform is designed to abstract the elementary data acquisition and control. It encapsulates it in spiking neural activity of neural populations that represent sensory data or generate motor commands as provided by Nengo. Such an interface allows the neurocomputing platform to natively operate on either real-valued encodings of the sensory data and motor commands or their spike based representations.

8.1.2 Neural algorithm development

Our overall goal is to develop robotic control systems that are programmable and, at the same time, implemented as neural networks. We want them to be programmable so that we can leverage expert



(a) Combined network. **(b)** Direct network.

Figure 8.3: Schematic visualization of two neural networks computing a complex function $m = h(f(x), g(w))$: (a) shows a combined network computing $m = h(f(x), g(w))$ while the network in (b) computes the function directly.

knowledge about the steps required to perform a task. We want them to be implemented as neural networks partly so that we can make use of energy-efficient neuromorphic hardware, but mostly because neural networks allow for gradual improvement of task performance via learning. However, neural networks on their own generally start with zero knowledge (random connection weights) and can often take a long time to learn, or completely fail to learn complex tasks. What we thus need is a method for taking a complex algorithm (i.e., the program we want to implement) and breaking it down into smaller components. Each of those components can then be implemented in a neural network. These individual neural networks can then be combined into one large neural network that can perform the entire task. This final network can then be implemented in neuromorphic hardware, and it could also be used as the starting point for further learning. Crucially, it should be noted that we are not trying to implement a perfect version of a task. Rather, we want to program a somewhat competent, initial version of a task that could then be further refined using a variety of neural network learning algorithms. In particular, Duan et al. (2016) provide a comprehensive survey of Deep Reinforcement Learning algorithms that are suitable for robotic control learning. Most of these algorithms are based around adjusting the connection weights of a neural network in order to improve performance on a task, based on occasional "positive" and "negative" reward values. Our long-term research goal is to apply these learning systems to the programmed neural networks that we describe in this section.

Network composition In order to build larger systems with complex algorithms, we can combine multiple neural networks together. For example, Figure 8.3a shows a system where we not only have $y = f(x)$, but we also have $z = g(w)$, and our final output is a function of the outputs of both of the other two networks, $m = h(y,z)$. By creating networks to compute these intermediate functions and then combining them together, we can implement complex computations. One crucial question, however, is what advantage do we get by breaking this complex function down into smaller, simpler functions. After all, it would have been possible to just make one single network that directly approximates the desired function. The primary reason not to combine functions together is that of scaling. As algorithms become more complex, it becomes harder and harder for neural networks to approximate them. The traditional solutions are to either increase the number of neurons in the middle layer, or to add more layers. Both

of these approaches make it harder for the learning algorithms to find the connection weights that best approximate the function, and make the network require more neurons, and thus more computational resources are needed. Importantly, one can think of Figure 8.3a as the result of starting with Figure 8.3b, adding in more hidden layers, and adjusting the connectivity. In other words, by building a complex network out of smaller networks, we are imposing structure on the network. We are specifically indicating that y and z are useful intermediate results that the network should find as an intermediate step before computing m. If we, as programmers, are correct in our decisions about this intermediate structure, then we can greatly simplify the process of creating these networks.
Given these tools, we can use Nengo to automatically construct large neural networks for us. In order to do this, we have to take our desired algorithm and break it into small parts, each of which is a function computed on an input vector or a differential equation.

8.1.3 Neural task implementation

Even the simplest behaviors are exploiting relations (i.e., functions) between sensory streams and motor commands. In order to design a neural controller able to adaptively switch control policies, it is natural to extract such functions in a reliable way, given the variability and uncertainty in the sensorimotor streams. The relatively complex and nonlinear interactions among the sensors on the robot and its actuators, makes the derivation of analytical forms of such functions hard. Alleviating the need for such precise and task-dependent modeling, neural networks are able to approximate such functions just from observations of the available sensorimotor streams. Moreover, using learning mechanisms, such systems can ultimately autonomously extract such functions from the data. In general, adaptive behavior is regarded as autonomous when the actions performed by the agent result from the interaction between its internal dynamics and the environment. Following such a perspective, our system is able to incrementally build up complex behaviors by superimposing more simple, basic behaviors.
In the current instantiation of our framework, we want to solve a non-trivial mobile manipulation task. Using our mobile platform, the task is to manipulate objects with LED stimuli blinking at different frequencies to bring them in order. More precisely, the task can be seen as a grasp and sort task, in which the robot will select a certain control policy depending on the current sensory streams (mainly visual input) to find the misplaced stimulus and place it in the correct location. Action selection is performed using a model of a biologically plausible neural circuit, namely the Basal Ganglia. The Basal Ganglia, according to T. C. Stewart et al. (2010), is an action selector that chooses whatever action has the best "salience" or "goodness". Selection is done on the basis of a context dependent utility signal for each possible action. Actions that are inappropriate for the current context may have low utility, and the task of the Basal Ganglia is to select the action that currently has the highest utility value. The Basal Ganglia will choose between the four possible behavior stacks: Grab, Hold And Move Side, Put Down or Finish. We design our control network by defining a set of intermediate-level networks, namely Grab, Hold And Move Side, Put Down and Finish that are composed of various different low-level behaviors, and then we create a high-level network whose outputs activate and deactivate the low-level networks. To accomplish this, each of the low-level networks will have an input a which indicates its level of activation. If a is 0 then the output from that network should be 0, and if a is 1 then it should perform its basic activity. It should be noted that this sort of control system design is strongly reminiscent of the classic subsumption architecture (Brooks, 1986). For each of these behaviors, we used Nengo to take the functions provided, generate input and output training data, use that data to generate individual neural networks.

Perception and motor-control: low-level reflex behaviors

Orient Left/Right using all cameras This behavior uses the x-position location of the target in all three camera views to control the rotation of the robot. If the object is on the left, it turns left, and if it is on the right, it turns right. This should cause the robot to turn to face the object. If it can not see the object, it does nothing. If the object can not be seen by a particular camera, it does not contribute.

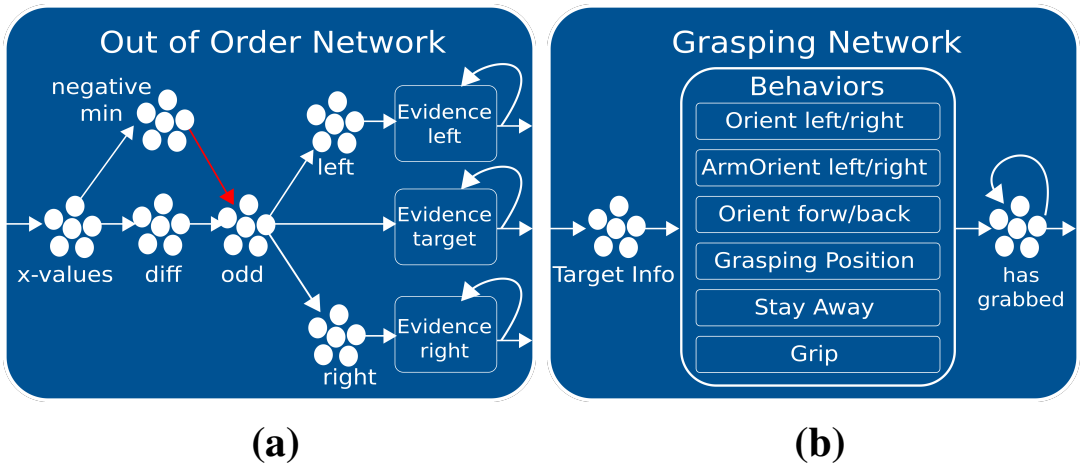


Figure 8.4: Schematic visualization of two sub-networks of our model with sets of white circles indicating neural populations while boxes depict sub-networks. (a) shows the *Out of Order* network, which detects if one frequency does not fit the assumed order as target and keeps this object's information in a memory. (b) illustrates the *Perform Grasping Action* network, which finds an object and grabs it.

Orient Left/Right Using Arm Camera Only This behavior is similar to the previous one, but only uses the arm camera. This is meant to be used when the arm camera has a good view of the target, allowing for a more fine-grained close-up control.

Move Forward/Backward to Grasping Distance Here, we use the binocular disparity to the object to control whether we should move forward or backward. This behavior will output 0 if the object is not mostly in front of the robot (as computed by averaging the x positions in the left and right camera). If it is in front of the robot, then we move forward or backward to achieve the desired disparity (hard-coded to be 0.8).

Move arm to grasping position This behavior simply moves the arm from its resting position to a position suitable for grasping.

Move Backwards This simply moves the robot base backwards.

Close grip This simply closes the gripper.

Move sideways This behavior moves the robot base sideways towards a goal position and rotates the base to keep the goal position in the middle of the field of view of the robot. We make use of the target positions of the neighboring objects (cf. *Out of Order* Network), where the middle between them is the goal position of this behavior.

Move arm to put-down position This behavior simply moves the arm from its holding position to a position suitable for putting down the object again.

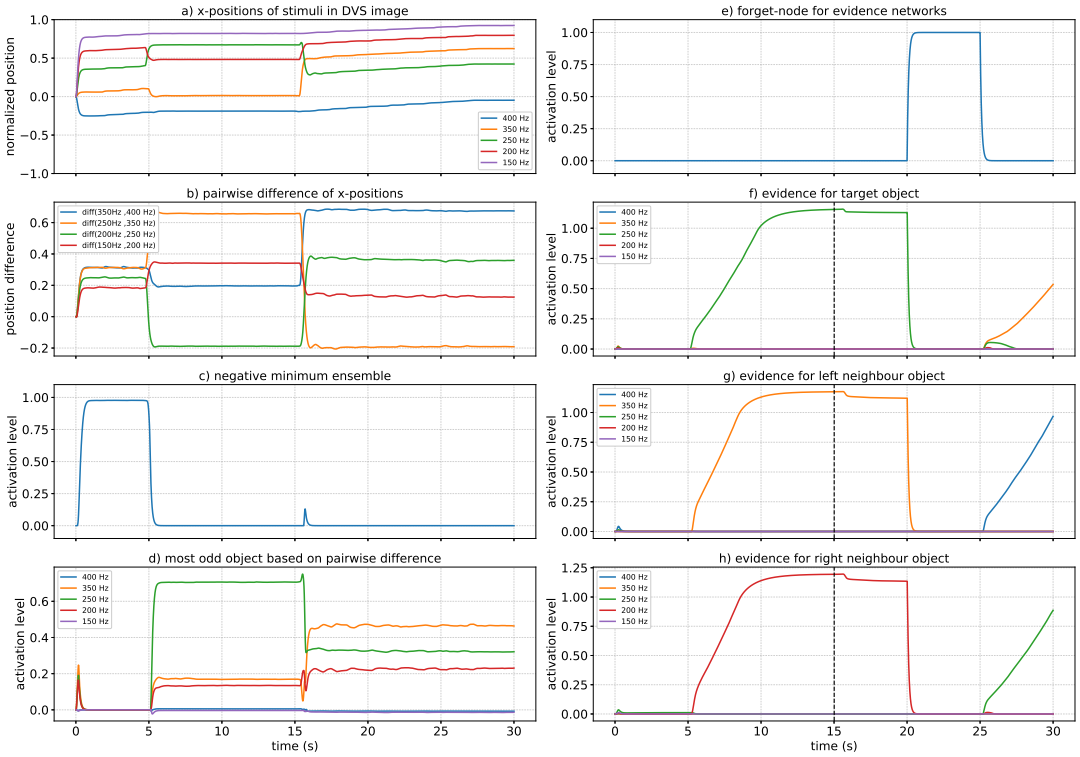
Reasoning: higher-level, cognitive behaviors

Out of Order Network This network computes the object to manipulate and serves as basis for all following behaviors built on top of it. Assuming the objects' blinking frequencies are given in descending

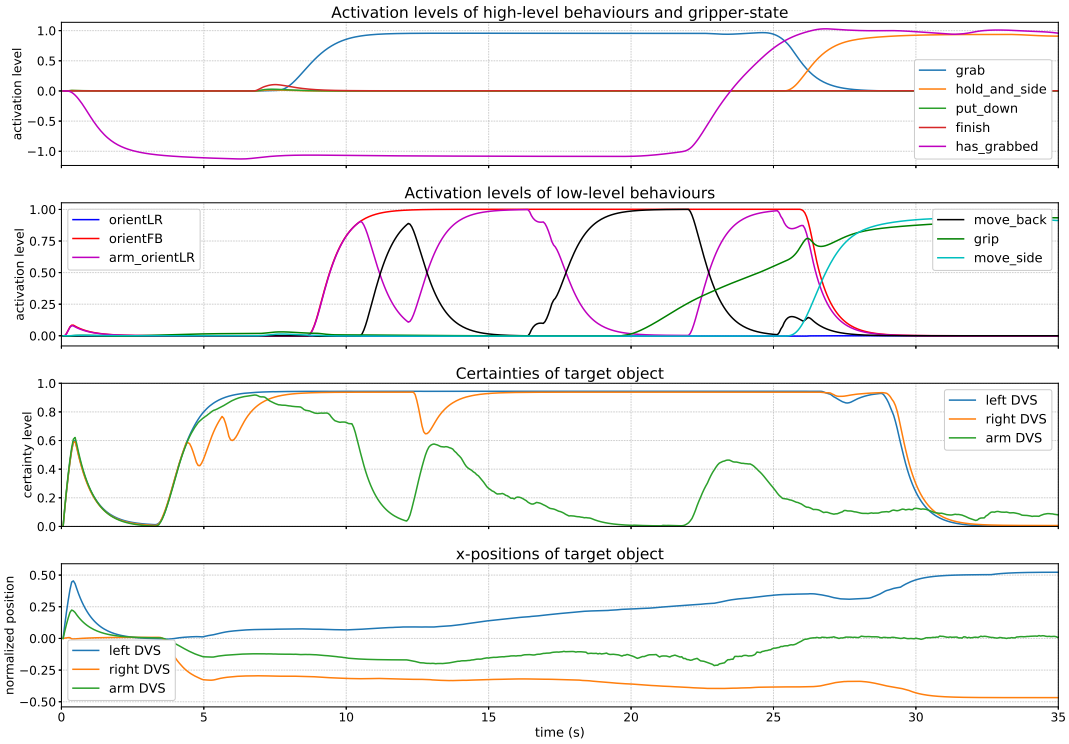
order, this network detects if one frequency does not fit the assumed order, indicates the corresponding object as target and keeps this object's information in a memory. Furthermore, the network detects those frequencies, which should be the neighbors of our detected target if in correct order and keeps their information in a memory as well. Figure 8.4a gives a schematic visualization of the network and its individual components. The x-values ensemble encodes x-positions of stimuli in DVS-image, the diff ensemble encodes pairwise differences between x-positions, the negative-min ensemble indicates if the minimal difference is negative, odd encodes the frequency, which is out of order (inhibited by negative min when all differences are positive), the evidence networks integrate evidence for the target object and the neighbor frequencies if in correct order. Figure 8.5a shows an example of actual DVS input data from the embedded tracking algorithm as well as the decoded output of the network's sub-components activity: during the first 5 s the stimuli are in correct descending order from left to right in the DVS-image (first plot in the left column of Fig. 8.5a), so the minimum pairwise difference is non-negative (second and third plots in the left column of Fig. 8.5a). In the interval 5 s to 15 s, the 250 Hz stimulus is put between the 150 Hz and 200 Hz stimuli, so now the sequence is out of order and the 250 Hz frequency is detected by the odd ensemble (last row in the left column of Fig. 8.5a) while the evidence networks integrate accordingly (second to last row in the right column of Fig. 8.5a). Starting from around 15 s the 250 Hz and 350 Hz stimuli are interchanged and the network's outputs change accordingly (left column of Fig. 8.5a). However, the evidence networks - as desired - still keep the information about the old target until a forget mechanism (first row of the right column in Fig. 8.5a) is triggered in the interval 20 s to 25 s allowing the evidence networks to recover for new input (right column of Fig. 8.5a).

Perform Grasping Action This is a high-level behavior that uses the low-level behaviors. The idea here is to find the object. If the robot is unable to see it in both cameras, then it backs up until it can and moves forward and backward until the robot is at the right distance. If the robot can see the object with the arm camera, then it uses the arm camera for orientation, otherwise it uses all three cameras. Also, if the robot is unable to detect the object with the arm camera, then it backs up. Note that the *MoveBack* behavior and the *Orientation* behaviors both move the robot forward and backward, so when they are both active the robot will end up achieving a position farther away from the object than when just *Orientation* behavior is active (the motor commands are summed). This positions the robot such that it can move forward and grab the object. Finally, if the robot is at the correct distance from the object (as measured by binocular disparity), and it is right in front of it, then it closes the gripper. This combination of actions serves to successfully grab the object. Figure 8.4b gives a schematic visualization of this network. The *TargetInfo* ensemble encodes sensory information (x- and y-position in the image, as well as radius and track-certainty for each DVS) of the target object (coming from the *evidence* sub-network in the out of order network), which serves as input for the low-level behaviors. The *has-grabbed* ensemble keeps the information once the object was grabbed in memory to indicate this task finished successfully. Figure 8.5b shows the activation levels of the high- and low-level behaviors (first two rows of Fig. 8.5b) as well as the sensory information the behaviors make use of, namely tracking certainty and disparity and x-position of the target object in the arm retina (last two rows in Fig. 8.5b).

Hold Object and Move To Goal Position This behavior combines holding an object by keeping the gripper closed while moving to the goal position at the same time. Here, we make use of the stored information about the target object's neighbors to calculate the goal position. Therefore, we use the mean value of the lateral positions of the left neighbor in the left base camera and the right neighbor in the right base camera as an estimation of the middle between the neighbor objects, which is where we want to place our target object. This value is used to control sideways and rotation motion of the base to navigate the robot to the correct position for putting down the target object (see Fig. 8.7a in the interval from 28 s to 37 s). Figure 8.6a gives a schematic visualization of this network. The Left/Right *TargetInfo* ensembles encode sensory information (x- and y-position in the image, as well as radius and track-certainty for each DVS) of the neighbor objects (coming from the left/right evidence networks in



(a) Decoded output of the *Out of Order* network's neural components based on DVS input data from embedded tracking



(b) Decoded output of the *Grab* network's neural components

Figure 8.5: Illustration of the *Out of Order* and *Grab* networks neural populations decoded output.

the out of order network), which serves as input for the low-level behaviors. The *MoveSideways* behavior uses the mean value of the lateral positions of the left and right neighbor in the left and right base camera

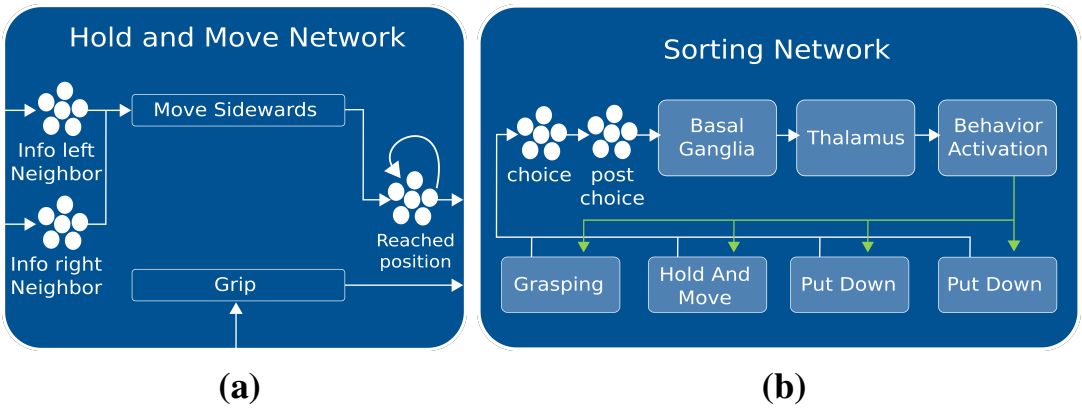


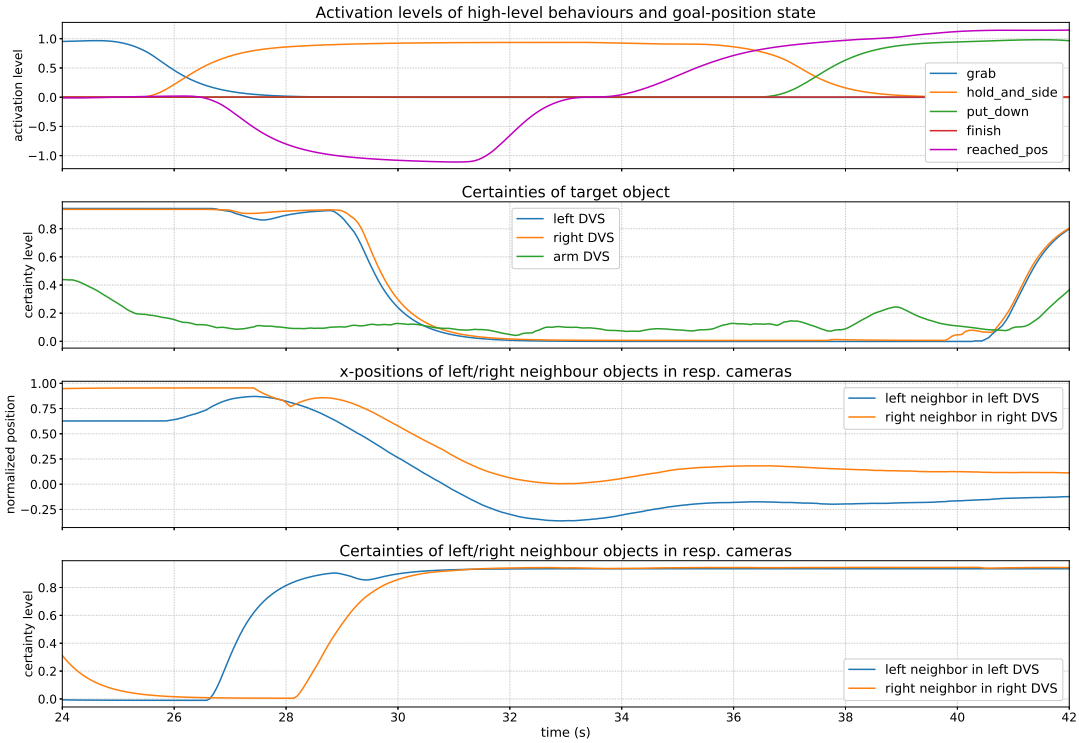
Figure 8.6: Schematic visualization of two networks of our model with sets of white circles indicating neural populations while boxes depict sub-networks. (a) shows the *HoldAndMove* network, while (b) illustrates the complete Sorting network. The boxes in the lower part visualize the subtask-networks, which are activated by the upper network chain for action selection incorporating the Basal Ganglia and Thalamus networks pre-implemented in Nengo.

respectively as an estimation of the middle between the neighbor objects and moves the base to this position, while the *Grip* behavior keeps the gripper closed. The *Reached position* ensemble serves as a memory integrating evidence once the goal position is reached to indicate that this sub-tasks finished successfully. Figure 8.7b shows the activation levels of the high- and low-level behaviors.

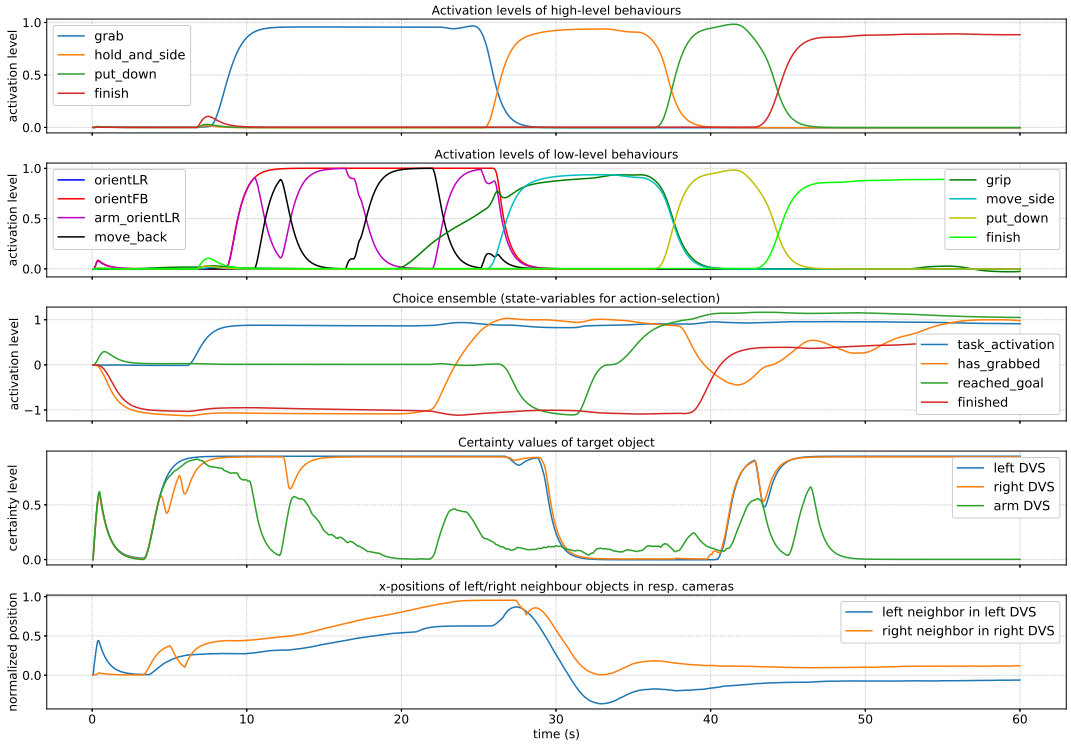
Put Object Down This behavior simply moves the arm from its gripping position to a position suitable for releasing an object, while opening the gripper and moving the base slightly backwards at the same time to ensure smooth and safe placement of the target object.

Finish Task This behavior simply makes the robot base back off from the manipulated objects. After stopping the base - implicitly by deactivating all other behaviors - the arm moves to back resting position automatically, which indicates that the whole sequence of tasks is completed.

Perform Sorting Task This is a high-level behavior combining all the other behaviors described so far to complete the whole task of moving the target object to its correct position in the sequence of frequencies. To choose the appropriate action to take, we used models of the Basal Ganglia and Thalamus proposed by T. C. Stewart et al. (2010), which are available as pre-implemented networks in Nengo. Corresponding to each of the high-level behaviors, we created input values for the Basal Ganglia network to choose from. Initially, *Perform Grasping Action* is enabled. Once the robot picked up the target object and built up sufficient evidence, the Basal Ganglia network activates the behavior to hold the (target) object and navigate the robot to the goal position. As soon as the robot reached its goal position between the neighbor objects and the according network built sufficient evidence, the *PutDown* behavior to put down the target object is activated. As soon as this behavior is completed, the whole sequence is wrapped up by activating the *Finish* behavior. Figure 8.6b gives a schematic visualization of the network. Figure 8.8 illustrates the most important stages of one example run while Figure 8.7b gives a visualization of the decoded output of the network's components: once the *Out of Order* network detected the target object (roughly the first 5 s), the *Grab* behavior is activated to find and grasp the target object (Fig. 8.8 a-c, Fig. 8.7b, t =8 s to 26 s). The different low-level behaviors for orientation and navigation are enabled based on the certainty and disparity of the tracked stimuli. Once the robot grabbed the target object, the *Basal Ganglia* activates the *HoldAndMoveSide* behavior (Fig. 8.8d-e, Fig. 8.7b, t =26 s to 39 s).



(a) Decoded output of the *HoldAndMove* network's neural components



(b) Decoded output of the *Sorting* network's neural components

Figure 8.7: Illustration of the *HoldAndMove* and *Sorting* networks neural populations decoded output.

The main sensory input in this phase is the x-position of the neighbor objects (last row in Fig. 8.7b). A clear indicator for successful pick-up is the decrease of certainty in the base-retinas (forth row in

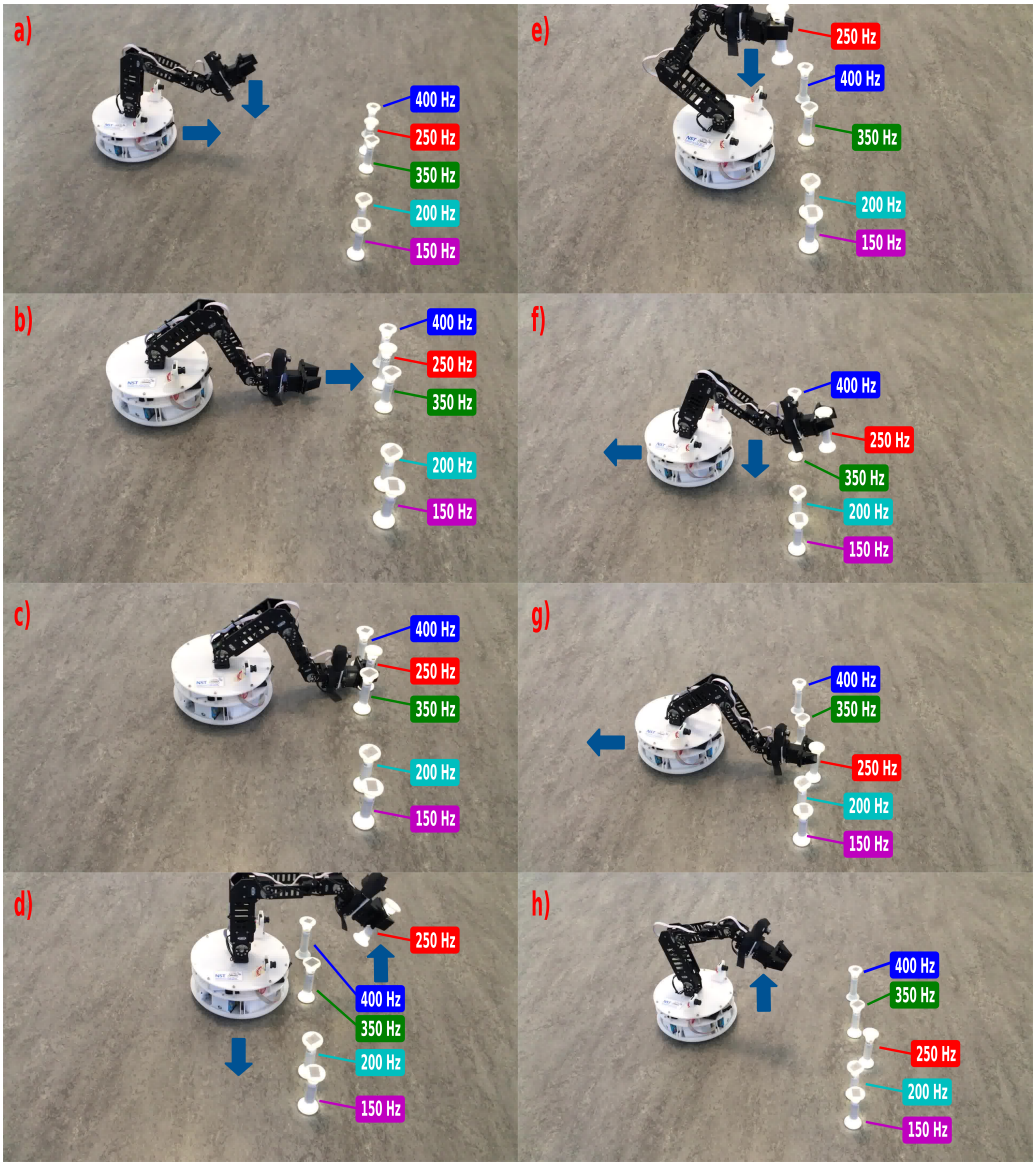


Figure 8.8: Selected stages of an example run of the *Perform Sorting Task*.

Fig. 8.7b). Once the robot reached its goal position between the neighboring objects, the *PutDown* behavior is activated (Fig. 8.8 f-g). Finally the whole task is wrapped up by the Finish behavior and all other behaviors are deactivated (Fig. 8.8 h) to put the robot back into resting position.

8.1.4 Summary

In this section, we have shown a mobile robotic manipulator capable of solving a pick-and-place task, with algorithmic components completely implemented in the framework of Spiking Neural Networks. This makes our approach not only suitable to expand its functionality with supervised and unsupervised learning methods but also highly scalable. The underlying Nengo-based neural compiler supports the use of dedicated neuromorphic hardware systems, which allows to run even large-scale neural networks in real-time. The possibility to introduce learning will eventually enable such systems to adapt their control policies and decision making to unpredictable changes in the environment. Furthermore, the neural implementation is beneficial in terms of allowing to combine networks "hand-programmed" by experts/engineers with learning networks that improve themselves over time with increasing data/experience. Here, our ultimate goal is to design systems that are capable of adapting to

new task during operation time while at the same time using experience from previous tasks and expert knowledge.

Limitations

The current algorithmic implementation has some inherent limitations. For now, the network is only able to detect one object not fitting in the sequence of frequencies. To overcome this limitation, the *Out of Order* network could be enhanced to solve the sorting-problem in a neural fashion. An intermediate workaround could be to repeat the current simple task and thereby solve the sorting problem incrementally. Another obvious limitation of the current implementation is the need for neighbor objects to find the goal position for put-down. This makes it impossible to solve the task for those objects blinking at a maximum or a minimum frequency. As for the first limitation, the *Out of Order* network needs to become more sophisticated to detect the goal position in these edge-cases when a border-object needs to be manipulated. One possibility could be detect one neighbor and to use the maximum of the pairwise differences as distance estimation for the offset.

Outlook

A long-term goal is to use the current implementation as a starting point for self-improving learning systems. As the current implementation is built in the framework of SNNs (Spiking Neural Networks), it naturally supports learning. One direction for future research could be to use the current low-level behaviors for initialization and to let the system learn and improve them incrementally by experience. For instance, one problem in learning robotic systems such as policy search for motor control (Levine et al., 2016), is the acquisition of sufficiently large training data sets. Recording large amounts of training data by repeating one specific task with real robotic systems is time-consuming and often infeasible, while training-data from simulation is usually not realistic enough to capture the complexity of noisy real-world data. In these cases, a system which is able to use expert-knowledge for certain (sub-) tasks as a starting point could significantly speed-up the learning process. We believe that our current approach is a promising first step for further research in this direction.

8.2 Neuromorphic reinforcement learning for vehicle trajectory control

In this section, we propose a neuromorphic system for vehicle control. Our system is implemented entirely in a spiking neuron substrate using the Nengo simulator (Bekolay et al., 2014) and is designed to be both distributed and hierarchical. In a sample instantiation, we train a trajectory selection module using reinforcement learning to investigate the feasibility of a learnable, neuromorphic control system in an automotive context. This approach has been selected in order to be decoupled from the quality of training data in this first investigation. We evaluate our approach in TORCS (The Open Racing Car Simulator), which allows us to generate training data in a safe and controlled simulation environment. Furthermore, TORCS offers an advanced vehicle physics simulation as well as a variety of sensors and actors for interaction (Wymann et al., 2014).

8.2.1 Neuromorphic control architecture

The architecture of our proposed system is visualized in Fig. 8.9. Interactions with the TORCS environment (see Loiacono et al., 2010, for detailed information on the sensor and actuator setup) are channeled over several ROS (Robot Operating System) nodes used for the gateway communication. For evaluation and training purposes, a controller using the global position and orientation of the vehicle is imple-

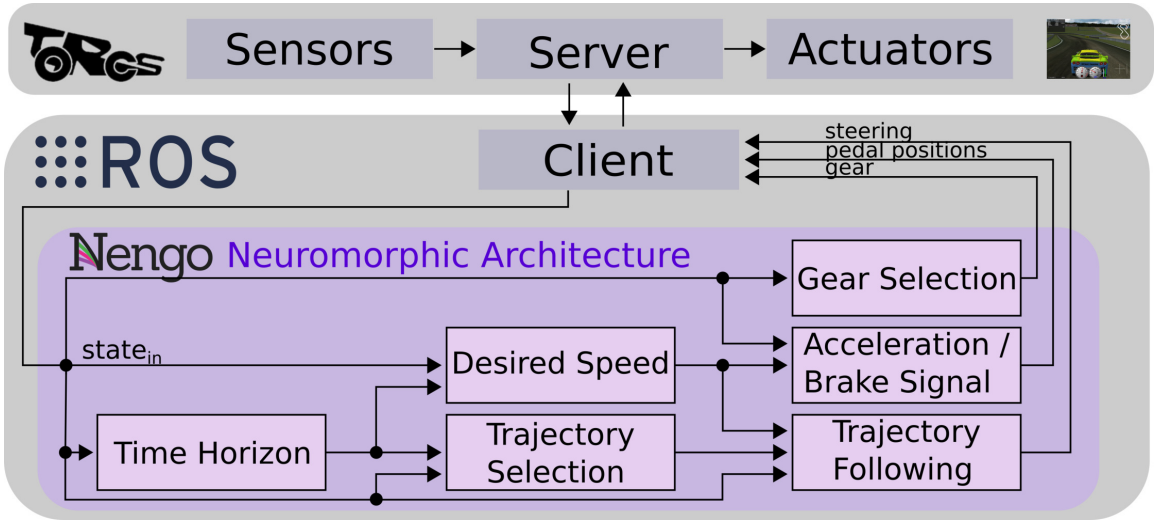


Figure 8.9: Proposed distributed neuromorphic architecture utilizing individual modules for separate control signal calculation.

mented in ROS to determine control signals to follow a given trajectory or the roadways centerline at a fixed speed[1].

The proposed architecture for a learnable and energy-efficient vehicle control system is a holistic neuromorphic approach for determining steering, gas and brake pedal as well as gear signals. It is a distributed system in the sense that these signals are calculated separately in different modules. In addition, it is a hierarchical system as several intermediate values have to be calculated and different modules and subsystems are dependent upon each other. The modules' vertical alignments within Fig. 8.9 indicate the distinction of three core subsystems, each responsible for one of the control signals (*gear selection*, *breaking/acceleration* and *steering*). We envision each of these sub-modules to be learnable either by supervised or reinforcement learning. Some of the depicted modules are heavily dependent on each other and will have to be trained in parallel rather than independently.

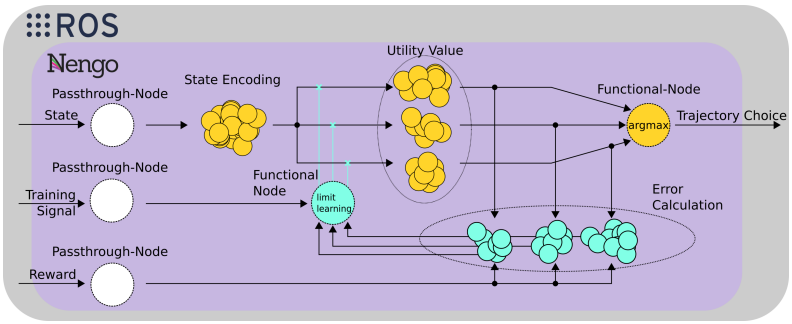
Here, we focus on the *trajectory selection* module to be the only learning module for simplicity. Therefore, gear selection and acceleration are determined from the engine's RPM (Rotations per minute) and a desired velocity set to a fixed value for now. The *trajectory following* module determines the control values for steering from the chosen trajectory. We envision the *time horizon* module to estimate the time window for the next trajectory to be followed. Assuming a perfect control algorithm and the desired speed to be the actual speed, this equates to determining the trajectory's end point's longitudinal position. Accordingly, the *trajectory selection* module is an action selection module for the agent to choose a trajectory type as well as the lateral component of the trajectory's end point from a set of predefined options.

Trajectory selection module

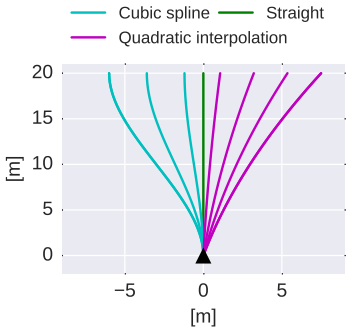
We focus on the *trajectory selection* module isolated from the other modules. Therefore, we use a classical controller to steer the vehicle along the chosen trajectories at a fixed velocity of 34 km/h with the trajectories' end points' longitudinal component set to 20 m, which emulates a simplified version of *trajectory following*.

We implemented a set of 15 trajectories to describe varying degrees of different behaviors (see Fig. 8.10b). We use straight lines to emulate lane following, cubic splines (with their derivative set to 0 at the extrem-

[1]The TORCS-to-ROS interface can be found at https://github.com/fmirus/torcs_ros. The Nengo and ROS implementation of the trajectory selection module and the framework that is built upon it can be found at https://github.com/fmirus/torcs_neural_trajectory_ctrl



(a) Simplified visualization of the SNN for *trajectory selection*. The yellow (A) resp. cyan networks (B) implement action selection resp. weight optimization for three exemplary trajectory choices



(b) Visualization of a set of exemplary trajectories.

Figure 8.10: SNN for *trajectory selection* and exemplary set of trajectories

ities) to model lane change maneuvers and quadratic interpolations between a start and end point to perform a change in orientation. The latter trajectories are intended for driving curves or to correct slight misalignments between the vehicle and the road.

8.2.2 Reinforcement Learning

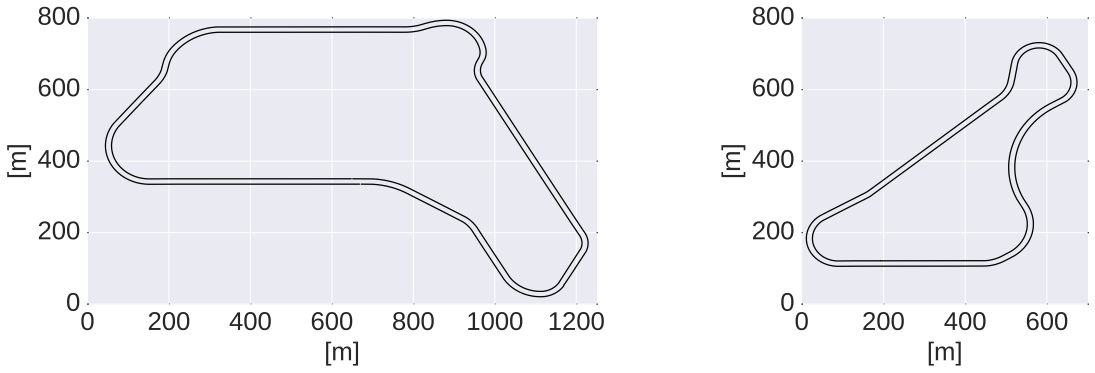
The core of this section is the learning Spiking Neural Network for *trajectory selection*, which is visualized in Fig. 8.10a and consists of two sub-networks. It was built using the Nengo neural simulator (Bekolay et al., 2013), which implements the principles of the NEF (Eliasmith and C. H. Anderson, 2003). The first sub-network A (yellow components in Fig. 8.10a) encodes the current state in a neural population. This population feeds its output to an array of downstream populations, each representing the utility value Q associated to one of the possible actions. Unless an exploration step is enforced, the highest filtered utility value determines the next action to be performed. As we want to adapt the mapping between input (state) and utility values by learning the respective connection weights, their decoders are initialized to a fixed value. The second sub-network B (cyan components in Fig. 8.10a) encodes the reward received from the environment (cf. Equation (8.1)). From this reward, the offset to the current utility values is calculated, which in turn is used to adapt the weights of sub-network A's learning connection.

Learning rule

The associative learning process is implemented using the PES (Prescribed Error Sensitivity) (Bekolay et al., 2013) learning rule, which modifies connection weights based on a (multi-dimensional) error signal $\mathbf{e}$. We calculate the error from the reward function

$$r(t) = |p(t)|\,\beta_1 \cdot |\theta(t)|\,\beta_2 + |p(t) - p(t-1)|\,\beta_3 + |\theta(t) - \theta(t-1)|\,\beta_4, \tag{8.1}$$

where p describes the vehicle's lateral position along the road at successive time steps t and $t-1$ (a value of 1 being centered and 0 being at the track boundary), θ describes the vehicle's orientation with respect to the road's centerline (values of 1 resp. 0 indicate parallel resp. perpendicular alignment) and β_k being scalar weights. Thus, the reward function is designed to blend the two objectives of driving aligned with and centered on the road.
With each ensemble representing a utility value $Q(s,a_j)$ of the state s when taking action a_j, we define the error for dimension j as



(a) Training **(b)** Validation

Figure 8.11: Visualization of the chosen training and validation tracks.

$$e_j = \begin{cases} r(t) - Q(s(t), a_j) & \text{if } a_j \text{ is selected} \\ 0 & \text{else.} \end{cases} \tag{8.2}$$

Here, we have chosen to limit the agent's knowledge to immediate rewards. Therefore, we neglect the correlation between the *trajectory selection* and *time horizon* modules, as it will introduce a temporal component to the discretization. Hence, the training procedure needs to be adjusted once both modules are trained jointly. Naturally, such a limited approach will impair the quality of our results. Our goal, however, is to show general feasibility of our approach which is why we postpone the coupling with other distributed modules to future work.

Training procedure

In order to evaluate the derived policy's performance, we chose one training and one validation track (Fig. 8.11) that show similar characteristics in road width and occurring curvatures. Therefore, they show comparable features while being distinct enough to identify if a straight mapping to the training track were to happen. Whenever the vehicle leaves the track during training, the simulation is reset and a reward of 0 is awarded as the track sensors become unreliable in such a scenario. After every five training runs, three runs until the vehicle goes off-track are driven on the validation course without updating any weights. This process constitutes one training/validation episode.
We use the percentage of completion on the validation track multiplied by the collected average reward in that validation episode as a measure of reliability performance. This measure is additionally normalized by the maximally achievable reward for the utilized parameter set. As the reset leads to a large bias in the experience of states close to the starting line during training, several cyclic points of interests have been defined up-front as substitute starting points before which a non-neuromorphic controller handles the vehicle. This procedure ensures a large exploration of different possible states and accelerates learning. During training, a linearly decaying epsilon exploration is utilized.

Results

Figure 8.12 visualizes the performance of the derived control policy on the validation track after each episode. The average performance tends to increase, indicating an overall improvement of the policy given the state values approximated by the SNN. The neuromorphic controller is able to reliably complete the validation lap after roughly 35 episodes without having performed any training on it. This indicates that the SNN is able to learn a meaningful policy and to generalize beyond the training track given the sensors' uncertainties.

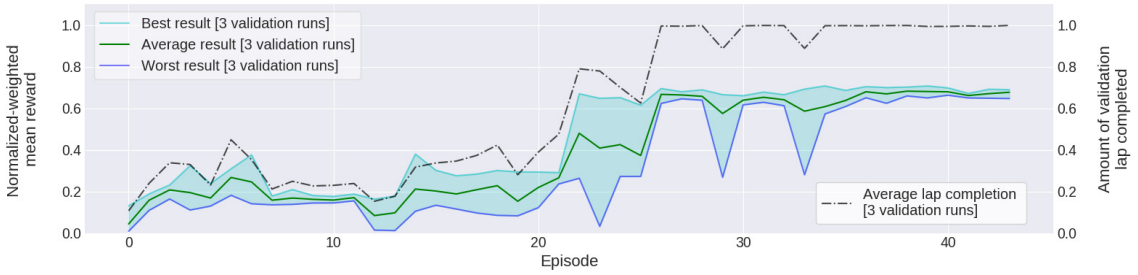


Figure 8.12: Results (mean reward and lap completion) for three validation runs per episode

8.2.3 Summary

Our investigation in this section shows promise to be a first step in the direction of a neuromorphic vehicle control. Although we chose a very limited approach in associative reinforcement learning, our system is able to fulfill the initial requirement to complete laps on the unknown validation track. However, tests on other validation tracks revealed issues with this approach. Successive curves are a problematic situation as the vehicle manages to pass the first curve but oftentimes maneuvers itself into a position where it is impossible to complete the second curve with the available trajectories. This problem can not be identified timely, and therefore solved with the current approach, as there is no temporal component in associative reinforcement learning.

Outlook

We envision to tackle the temporal issue using Q-learning with *dual learning*. Another direction for future work is to couple the *trajectory selection* with the *time horizon* module to enable planning over time while remaining within a discretized domain and delaying the continuous control to the *trajectory following* module. These modules could be trained separately or jointly to evaluate applicability of this more sophisticated approach.

8.3 Summary

In this chapter, we have presented a first step towards a neuromorphic control architecture. Due to the absence of a suitable vehicle demonstrator, lacking integration of neuromorphic prototypes into current vehicle hardware architectures and finally due to safety considerations, we have demonstrated the general feasibility of such a neuromorphic control architecture in two sample instantiation. First, we presented a proof-of-concept implementation of a non-trivial mobile robot manipulation tasks in the substrate of SNNs. We showed that our architecture can be used to induce human expert knowledge into neural task implementations. Such manually designed networks can either be used to complement other, decoupled learning networks or serve as a basis to bootstrap learning to avoid starting the training procedure from a completely blank state. The second proof-of-concept implementation combines manually designed and automatically learned networks in the context of short-term vehicle trajectory planning. We use reinforcement learning in SNNs to select trajectories for a classical controller, which is decoupled from the learning network. Thereby, we demonstrate that neuromorphic principles can be used in the context of vehicle control. However, today's vehicle architectures are not designed yet to integrate neuromorphic hardware, which on the other hand have not reached the technical and commercial maturity to be integrated in such architectures. This lack of integration prohibits the adoption of any algorithm implemented in a spiking neuron substrate into vehicle applications, since the advantages of this algorithmic substrate are closely coupled to energy-efficient deployment on dedicated hardware. That is, one of the unique strengths of SNNs in contrast to more traditional learning approaches is their ability to be effi-

ciently processed on neuromorphic hardware mostly leading to a trade-off between efficiency and higher accuracy delivered by traditional approaches.

9 Discussion

This thesis presented a novel perspective on several automotive tasks with regard to cognitive modeling and/or neuromorphic principles, particularly representation of the environment around the vehicle. After giving an overview of recent research in the field of neuromorphic engineering regarding both hardware and software, cognitive modeling as well as automated driving in chapter 2, we proceeded to the theoretical background in chapter 3, the backbone of this thesis. We established a coherent mathematical formalism to describe the theory and properties behind VSAs, a family of modeling techniques deploying representations based on high-dimensional vectors. In this coherence, such a description is not available in the literature except for treatments of particular instantiations of VSAs such as in T. Plate (1994), Gayler (1998), and Kanerva (2009). We proceeded to describe the theory of the SPA, a special case of the general construct of VSAs. We presented essential features like circular convolution, unitary vectors and convolutive powers, which form the basis of structured representations used in later chapters. Following Eliasmith and C. H. Anderson (2003), we presented a brief introduction to the NEF and showed how its principles can be applied to implement the SPA in a spiking neuron substrate. Finally, we introduced established approaches to generate structured representations to build cognitive models from to be used in later chapters.

In chapter 4, we established our general approach to represent automotive scenes in the representational substrate of the SPA. Following the workflow of Gallant and Okaywe (2013), we first presented different options to generate vocabularies of atomic vectors and secondly, how to generate structured representations from them. We proposed several approaches to encapsulate different structures such as visual similarity, semantic similarity and a combination of both in vector vocabularies. We focused on encoding entities typically occurring in automotive context, namely traffic signs and traffic participants. While manual vocabulary design is feasible and demonstrated in this context of such rather small vocabularies, we also successfully demonstrated approaches to automatically learn such vocabularies based on CNN learning systems for visual similarity and unsupervised word embedding algorithms like word2vec for semantic similarity. Proceeding to structured representations based on atomic vocabularies, we presented several approaches to encapsulate numerical or, more particularly, spatial information in a vector substrate. The main contribution was the introduction of a representation for spatial information using the convolutive power established in chapter 3. We concluded the chapter with an experimental analysis of the amount of information, that can effectively be stored in structured vector representations. We analyzed the capacity of such representations for simple superposition of concepts as well as superpositions of vectors encapsulating spatial information using the convolutive power yielding upper bounds for the total number of concepts or objects being encoded.

Starting with chapter 5, we proceeded to the third stage as proposed in Gallant and Okaywe (2013), which was omitted in chapter 4: the *output computation* stage or, in other words, applying our vector representations to particular tasks. We presented a spiking neuron model learning to classify the current driving context based on a vector representation of the current scene from real-world driving data. This model made use of one of the key strengths of vector representations, namely being able to combine symbol-like manipulation with neural network learning. We demonstrated that our model is able to capture the semantics of the scene to successfully classify the current driving context. Comparing the model's performance with a traditional deep network trained on the same input data, a CNN trained on raw visual input as well as human level-performance, we demonstrated that the representation was successful in abstracting away visual features irrelevant for the task at hand. However, the data set used in this chapter has clear limitations in terms of size and diversity with human bias being imposed through the manual labeling process. We concluded the chapter by analyzing the influence of structured vocabularies

created in chapter 4 on the model's performance. Although imposing more structure into the vector representation led to measurable differences in the models' classification accuracy, most of the structured vocabularies only deteriorated performance with only the visual-semantic vocabulary achieving subtle improvements. We assume that the limited amount of data as well as the rather small vocabulary, with not even all vectors absorbing the similarity structure, are the main reasons for these results.

Proceeding to another essential task in automotive context in chapter 6, we encapsulated spatial positions of several objects into one coherent vector representation of fixed length using the convolutive power. We trained several learning models to predict the future trajectory of one target vehicle based on that vector representation. On the one hand, we employed traditional deep learning models based on LSTM cells to predict the future trajectory based on a sequence of semantic vectors. We compared these models to similar networks making predictions based on other, reference encoding schemes of the input data and a simple linear predictor. Evaluating these systems on two real-world driving data sets, we observed subtle performance nuances without one model clearly outperforming the others while simple linear prediction already offered solid prediction accuracy especially for short term horizons. The main reason for this result is the composition of both data sets with straight driving constituting the majority of the samples compared to more challenging maneuvers such as lane changes. However, we were able to demonstrate that the models using the convolutive power representation tended to perform better in situations where the target vehicle drives closely to other vehicles in crowded situations. Furthermore, this model was also able to predict lane change situations better than all other models when trained on a subset of the training data consisting of these particular situations, whereas the other models did not show a similar adaptation behavior. Hence, we expect the models to improve with a more balanced data set since the data sets used in this work show a strong imbalance towards straight driving compared to lane change maneuvers. Finally, we observed when training an unsupervised learning algorithm for detecting anomalies on the convolutive vector representation, that it tended to label crowded situations with a higher number of vehicles driving closely to the target as outliers. Therefore, we conclude, that the encoding of the scene in semantic vectors is able to capture the essential semantics of the spatial information of the driving scene. Additionally, we compared the LSTM models to simpler single-layer SNNs, which achieved results slightly worse, but comparable to the more complex networks.

In chapter 7, we extended our work on vehicle trajectory prediction by developing a novel, mixture-of-experts online learning model trained at run time to refine the anticipations of several individual prediction models using delta rule learning and spiking neurons. We tackled the issue of delayed error signals introducing potentially long lags into the learning process through a temporal spreading of the error signal. That is, the model delays the error signal of earlier prediction steps to later prediction steps assuming that the general direction of the error is the same for earlier and further prediction horizons. We presented a context-free model variant updating its weights simply based on the prediction error as well as a model incorporating the current driving context. We used the indicators for crowded situations identified in chapter 6 as description for this context. We conducted a thorough analysis and showed, that this context-sensitive model performs significantly better than the context-free version. Furthermore, we showed that the model employing temporal spreading is successful in learning to predict trajectories already after being presented with a small number of vehicles.

Finally in chapter 8, we introduced a neuromorphic control architecture to provide a first step towards linking the higher-level tasks of earlier chapters, which are more focused on knowledge representation, to actual control. We showed, that implementing control algorithms in this spiking neuron substrate has two key advantages: it allows to manually program certain networks inducing human expert knowledge while employing automated learning in other, decoupled models, which can be validated separately. Furthermore, the manually implemented networks could function as initialization of the learning networks to refine task performance by bootstrapping the learning process instead of starting from a completely blank state. We demonstrated these principles on two proof-of-concepts implementations: a non-trivial mobile robot manipulation task as well as a simple reinforcement learning model short-term vehicle trajectory control in a simulated environment.

9.1 Conclusion and outlook

The results achieved in this thesis allow a novel perspective on knowledge representation, modeling and computation in automotive context through distributed representations and SNNs. We introduced a novel kind of representational substrate to encapsulate automotive scenes in high-dimensional vectors that additionally can be implemented in the language of SNNs. After evaluating the properties and limitations of this representation approach, we applied it to two automotive tasks, namely driving context classification and vehicle trajectory prediction. One key feature of vector representations is their distributed nature as well as their ability to encapsulate symbol-like concepts as well as numerical information. However, we learned that there are certain limitations depending on the dimension of the representational vector space to the amount of information the vectors are able to capture. We presented upper bounds for simple superposition representations as well as more complex encodings of spatial structures employing convolutive powers. However, these bound impose specific limitations on the representations themselves: when encapsulating increasingly complex structures through the architecture's algebraic operations, the limits enforce a certain focus on the most relevant features to encode.

Additionally, our analysis in chapter 5 showed, that encoding certain similarity structures within the underlying vector vocabularies themselves does not necessarily improve the performance of downstream learning models employing such representations. Hence, good care has to be taken when deciding for similarities to be encoded in the vector vocabulary since random vocabularies, although not following an inherent structure, already have some desirable properties. Furthermore, it might be beneficial to generate several vocabularies to encode different similarities to encapsulate several aspects of the same concepts. For instance, we as humans have a very wide understanding of perceived entities for example visual appearance, sound, smell, taste or their linguistic meaning: the word *apple* can stand for a fruit with a certain look, taste and smell, while it also could be referring to several electrical devices. This approach of generating different vocabularies for aspects of objects has not been investigated in this thesis. Another idea regarding vocabularies could be to learn the embedding based on the performance of a learning model on the task to be solved. That is, starting out with a randomly chosen vocabulary used by a model solving the desired task and then, based on this model's task performance, employ randomization, unsupervised learning or evolutionary algorithms to adapt the underlying vocabulary to let it converge to one best suitable for solving the particular task at hand. This also circumvents the human bias often explicitly imposed on the structure encoded in such vocabularies through either manually engineering the desired similarity structure or deciding for one to be automatically learned. On the other hand, the evaluation of the driving context classification model showed that vector representations of driving situations are able to capture the relevant semantics of the scene abstracting away potentially unnecessary features. However, current sensor systems do not natively produce such vector representations as output, which could be an alternative as a low-dimensional semantic compression of high-dimensional raw sensory values. Once perception of the outside world would be available directly in such a vector substrate, we could avoid the intermediate step of vectorization by engineering the structured representations of the scene.

We proceeded to investigate models for vehicle trajectory prediction models based on our vector representation encoding spatial information through the convolutive vector power in chapter 6. There were three key findings to point out: first, there is not one model or representational substrate of the input data that outperforms all other models in all driving situations. In contrast, each model and representation showed particular strengths and weaknesses, which we are able to draw a connection to the current driving context. For instance, the models using the convolutive power representation of the current scene tended to perform better in crowded and potentially dangerous situations with multiple vehicles driving close to one another. These findings were further investigated in chapter 7 introducing a novel online learning model employing a mixture-of-experts approach to weight several prediction models based either solely on their prediction error or by incorporating contextual information. This weighting is intended to be learned while the model is running for being able to adapt to unexpected behavior of

particular vehicles and adjust the model accordingly, which is not possible for models trained and validated offline. We demonstrated that it is possible to improve over the individual offline models through this online learning approach, applying temporal spreading of the error signal from earlier to later prediction horizons to avoid lags in the learning process. However, the advanced model having to deal with temporally delayed error signals was not able to achieve results comparable to a simplified model variant, which was virtually given access to the future error at prediction time. This simplified model was investigated to decide between the context-free and context-sensitive version while also serving as an upper bound for the more complex model having to deal with delayed error signals. Thus, there is still room for improvement for this online learning system in terms of parameter tuning. Furthermore, it could be investigated how varying contextual information influence the model's performance. For instance, we could simply use the vector representing the current scene as context information or include other descriptive values into the context. Additionally, other online learning approaches, which are more advanced than the simple delta rule learning rule employed in this thesis, could be investigated.

The second key finding regarding trajectory prediction is that the data set used for training has a significant impact on the learning capabilities of the models. This is not a new finding: deep learning approaches mainly rely on vast amounts of data to adjust their multitude of parameters. However, the issue of composition of data sets is rather rarely investigated. We found for the particular task of vehicle trajectory prediction, that both our data sets consist predominantly of straight driving with rather rarely occurring samples containing a lane change of the target vehicle. We demonstrated significant changes in the models' performance when trained solely on the samples containing a lane change of the target vehicle. This supports the hypothesis that a more balanced data set would most likely improve the learning models' performance on critical data samples that can not be predicted using simple linear regression. However, we expect that the prediction of lane change maneuvers could even be further improved when incorporating additional information such as lane information (Which lane are the vehicles driving? How large is the distance to the lane border?) or dynamical information such as velocity or acceleration. The imbalanced composition of the data, however, was somewhat expected, since both data sets mainly consist of highway driving, which reveals a second issue: although our models learned to reasonably predict vehicle behavior based on the presented training data, the same models would most likely show weaker performance when presented with data consisting of interurban or inner-city driving. Hence, we would either have to train on even larger data sets containing a well-balanced mix of city, interurban and highway driving with all of these sub-categories being more balanced as the current data sets or we could also train separate models for each driving context category and select at run time the best suitable model variant. The latter approach would offer the interesting possibility to combine all models presented in this thesis: a driving context classification model predicting the current context, which in turn could be used as contextual information for an online learning model, that decides for the most suitable prediction model in the current situation.

An additional evaluation of unsupervised anomaly detection supports the third key finding that there is sufficient information encoded in our vector representation to at least classify driving situations as outliers and thus potentially dangerous, since there are more objects present and closer to the target and ego-vehicle than in the "normal" samples. However, the outliers do not contain proportionally more lane change samples than the complete data set. Additionally, the anomaly detection system was trained with an unsupervised learning approach on an unlabeled data set. Hence, it was impossible to qualitatively evaluate the model's performance against known ground truth data.

Finally, in chapter 8, we introduced a novel neuromorphic control architecture to demonstrate at least general feasibility and usability of neuromorphic control principles employing spiking neurons as algorithmic substrate. We presented two proof-of-concept implementations for mobile robot manipulation on a real robotic system and a simplified vehicle trajectory planning system using reinforcement learning in an open source race car simulator. We demonstrated that our neuromorphic control architecture employing SNNs allows to decouple manually designed networks from learning networks, or even combine the two by using a manual implementation as starting point to bootstrap the neural network's learning phase.

However, we did not actually explore this possibility of extending an existing manual task implementation in spiking neurons to further improve task performance through learning. Furthermore, although both proof-of-concept implementations translated sensory stimuli to complex control behaviors, they are not fully coupled or integrated into neuromorphic computing hardware.
In conclusion, higher level cognition and lower level perception/action are not two separate aspects of the same system, but tightly coupled and integrated in biological organisms. While artificial systems already show advanced capabilities regarding perception and action, the principles and functions in humans are less well understood. The first coherent integration of both aspects into a large-scale cognitive model making sense out of complex sensor stream into active motion was presented in Eliasmith et al. (2012). However, the overall question of how biological organism effectively integrate and combine low-level behaviors and higher-level cognitive functions is still an unsolved scientific problem. Furthermore, a complete integration of such a *brain model* in an actual *body* of hardware is yet to be developed. The goal of this thesis was to propose a first step into the direction of cognitive automated vehicles. However, it was originally intended to complement the thesis at hand with another, more hardware oriented thesis, which was unfortunately not completed. Revisiting the original motivation introduced in chapter 1, we achieved promising results regarding neuromorphic principles in an automotive context on both "ends" of the perception-action-cycle. However, a coupled integration into one coherent system as well as the integration into neuromorphic hardware to demonstrate benefits regarding energy-efficiency of our proposed approaches is still missing and an interesting task for future work. There are several promising neuromorphic hardware prototypes such as SpiNNaker, IBM's TrueNorth and Intel's Loihi chip each offering a unique perspective on spike-based computation, which could be evaluated with the models proposed in this work and possible extensions.

Bibliography

Abadi, M., P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng (2016). "TensorFlow: A System for Large-scale Machine Learning". In: *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*. OSDI'16. Savannah, GA, USA: USENIX Association, pp. 265–283. ISBN: 978-1-931971-33-1. URL: http://dl.acm.org/citation.cfm?id=3026877.3026899.

Aeberhard, M., S. Rauch, M. Bahram, G. Tanzmeister, J. Thomas, Y. Pilat, F. Homm, W. Huber, and N. Kaempchen (2015). "Experience, Results and Lessons Learned from Automated Driving on Germany's Highways". In: *IEEE Intelligent Transportation Systems Magazine* 7.1, pp. 42–57. ISSN: 1939-1390. DOI: 10.1109/MITS.2014.2360306.

Aeberhard, M., S. Schlichtharle, N. Kaempchen, and T. Bertram (2012). "Track-to-Track Fusion With Asynchronous Sensors Using Information Matrix Fusion for Surround Environment Perception". In: *IEEE Transactions on Intelligent Transportation Systems* 13.4, pp. 1717–1726. ISSN: 1524-9050. DOI: 10.1109/TITS.2012.2202229.

Agranat, A., C. Neugebauer, and A. Yariv (1990). "A CCD based neural network integrated circuit with 64K analog programmable synapses". In: *1990 IJCNN International Joint Conference on Neural Networks*. IEEE. IEEE, pp. 551–555. DOI: 10.1109/ijcnn.1990.137623.

Ahmad, S. and J. Hawkins (2015-03-25). "Properties of Sparse Distributed Representations and their Application to Hierarchical Temporal Memory". In: *arXiv e-prints*, arXiv:1503.07469, arXiv:1503.07469. arXiv: http://arxiv.org/abs/1503.07469v1 [q-bio.NC].

Ahmad, S., A. Lavin, S. Purdy, and Z. Agha (2017). "Unsupervised real-time anomaly detection for streaming data". In: *Neurocomputing* 262. Online Real-Time Learning Strategies for Data Streams, pp. 134–147. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2017.04.070. URL: http://www.sciencedirect.com/science/article/pii/S0925231217309864.

Ahmad, S. and L. Scheinkman (2019-03-27). "How Can We Be So Dense? The Benefits of Using Highly Sparse Representations". In: *arXiv e-prints*, arXiv:1903.11257, arXiv:1903.11257. arXiv: http://arxiv.org/abs/1903.11257v2 [cs.LG].

Akopyan, F., J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta, G.-J. Nam, B. Taba, M. Beakes, B. Brezzo, J. B. Kuang, R. Manohar, W. P. Risk, B. Jackson, and D. S. Modha (2015). "TrueNorth: Design and Tool Flow of a 65 mW 1 Million Neuron Programmable Neurosynaptic Chip". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 34.10, pp. 1537–1557. ISSN: 0278-0070. DOI: 10.1109/TCAD.2015.2474396.

Alahi, A., K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei, and S. Savarese (2016). "Social LSTM: Human Trajectory Prediction in Crowded Spaces". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 961–971. DOI: 10.1109/CVPR.2016.110.

Altche, F. and A. de La Fortelle (2017). "An LSTM network for highway trajectory prediction". In: *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, pp. 353–359. DOI: 10.1109/itsc.2017.8317913.

Alvarez, J. M. Á. and A. M. Lopez (2011). "Road Detection Based on Illuminant Invariance". In: *IEEE Transactions on Intelligent Transportation Systems* 12.1, pp. 184–193. ISSN: 1524-9050. DOI: 10.1109/TITS.2010.2076349.

Amir, A., P. Datta, W. P. Risk, A. S. Cassidy, J. A. Kusnitz, S. K. Esser, A. Andreopoulos, T. M. Wong, M. Flickner, R. Alvarez-Icaza, E. McQuinn, B. Shaw, N. Pass, and D. S. Modha (2013). "Cognitive computing programming paradigm: A Corelet Language for composing networks of neurosynaptic cores". In: *The 2013 International Joint Conference on Neural Networks, IJCNN 2013, Dallas, TX, USA, August 4-9, 2013*, pp. 1–10. DOI: 10.1109/IJCNN.2013.6707078.

Anderson, J. R. (1983). "A spreading activation theory of memory". In: *Journal of Verbal Learning and Verbal Behavior* 22.3, pp. 261–295. ISSN: 0022-5371. DOI: 10.1016/s0022-5371(83)90201-3.

— (1996). "ACT: A simple theory of complex cognition". In: *American Psychologist* 51.4, pp. 355–365. DOI: 10.1037/0003-066x.51.4.355.

Applied Brain Research Inc. (2018). *The Nengo neural simulator*. URL: https://www.nengo.ai/ (visited on 2018-04-05).

Arthur, J. V., P. A. Merolla, F. Akopyan, R. Alvarez, A. Cassidy, S. Chandra, S. K. Esser, N. Imam, W. Risk, D. B. D. Rubin, R. Manohar, and D. S. Modha (2012). "Building block of a programmable neuromorphic substrate: A digital neurosynaptic core". In: *The 2012 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8. DOI: 10.1109/IJCNN.2012.6252637.

Axenie, C. and J. Conradt (2015). "Cortically Inspired Sensor Fusion Network for Mobile Robot Egomotion Estimation". In: *Robotics and Autonomous Systems* 71.C, pp. 69–82. ISSN: 0921-8890.

Bacha, A., C. Reinholtz, A. Wicks, M. Fleming, A. Naik, M. Avitabile, and N. Elder (2004). "The DARPA Grand Challenge: overview of the Virginia Tech vehicle and experience". In: *Proceedings. The 7th International IEEE Conference on Intelligent Transportation Systems (IEEE Cat. No.04TH8749)*, pp. 481–486. DOI: 10.1109/ITSC.2004.1398947.

Badrinarayanan, V., A. Kendall, and R. Cipolla (2015). "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation". In: *Computing Research Repository (CoRR)* abs/1511.00561. URL: http://arxiv.org/abs/1511.00561.

Bahram, M., C. Hubmann, A. Lawitzky, M. Aeberhard, and D. Wollherr (2016). "A Combined Model- and Learning-Based Framework for Interaction-Aware Maneuver Prediction". In: *IEEE Transactions on Intelligent Transportation Systems* 17.6, pp. 1538–1550. ISSN: 1524-9050. DOI: 10.1109/TITS.2015.2506642.

Barranco, F., C. Fermuller, and Y. Aloimonos (2014). "Contour Motion Estimation for Asynchronous Event-Driven Cameras". In: *Proceedings of the IEEE* 102.10, pp. 1537–1556. ISSN: 0018-9219. DOI: 10.1109/JPROC.2014.2347207.

Bekolay, T., J. Bergstra, E. Hunsberger, T. DeWolf, T. C. Stewart, D. Rasmussen, X. Choo, A. R. Voelker, and C. Eliasmith (2014). "Nengo: A Python tool for building large-scale functional brain models". In: *Frontiers in Neuroinformatics* 7.48. ISSN: 1662-5196. DOI: 10.3389/fninf.2013.00048.

Bekolay, T., C. Kolbeck, and C. Eliasmith (2013). "Simultaneous unsupervised and supervised learning of cognitive functions in biologically plausible spiking neural networks". In: *35th Annual Conference of the Cognitive Science Society*. Cognitive Science Society, pp. 169–174.

Benjamin, B. V., P. Gao, E. McQuinn, S. Choudhary, A. R. Chandrasekaran, J.-M. Bussat, R. Alvarez-Icaza, J. V. Arthur, P. A. Merolla, and K. Boahen (2014). "Neurogrid: A Mixed-Analog-Digital Multichip System for Large-Scale Neural Simulations". In: *Proceedings of the IEEE* 102.5, pp. 699–716. ISSN: 0018-9219. DOI: 10.1109/jproc.2014.2313565.

Benosman, R., C. Clercq, X. Lagorce, S. H. Ieng, and C. Bartolozzi (2014). "Event-Based Visual Flow". In: *IEEE Transactions on Neural Networks and Learning Systems* 25.2, pp. 407–417. ISSN: 2162-237X. DOI: 10.1109/TNNLS.2013.2273537.

Bertozzi, M., A. Broggi, and A. Fascioli (2000). "Vision-based intelligent vehicles: State of the art and perspectives". In: *Robotics and Autonomous Systems* 32.1, pp. 1–16. ISSN: 0921-8890. DOI: 10.1016/S0921-8890(99)00125-6. URL: http://www.sciencedirect.com/science/article/pii/S0921889099001256.

Beyeler, M., F. Mirus, and A. Verl (2014). "Vision-based robust road lane detection in urban environments". In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4920–4925. DOI: 10.1109/ICRA.2014.6907580.

Bi, G.-Q. and M.-M. Poo (2001). "Synaptic Modification by Correlated Activity: Hebb's Postulate Revisited". In: *Annual Review of Neuroscience* 24.1, pp. 139–166. DOI: 10.1146/annurev.neuro.24.1.139.

Bittel, S., V. Kaiser, M. Teichmann, and M. Thoma (2015). "Pixel-wise Segmentation of Street with Neural Networks". In: *Computing Research Repository (CoRR)* abs/1511.00513. URL: http://arxiv.org/abs/1511.00513.

Blouw, P., E. Solodkin, P. Thagard, and C. Eliasmith (2016). "Concepts as Semantic Pointers: A Framework and Computational Model". In: *Cognitive Science* 40.5, pp. 1128–1162. ISSN: 1551-6709. DOI: 10.1111/cogs.12265.

Boerlin, M. and S. Denève (2011). "Spike-Based Population Coding and Working Memory". In: *PLoS computational biology* 7.2. Ed. by K. J. Friston, pp. 1–18. DOI: 10.1371/journal.pcbi.1001080.

Bohte, S. M., J. N. Kok, and H. LaPoutre (2002). "Error-backpropagation in temporally encoded networks of spiking neurons". In: *Neurocomputing* 48, pp. 17–37.

Bohte, S. M. (2004). "The evidence for neural information processing with precise spike-times: A survey". In: *Natural Computing* 3.2, pp. 195–206. ISSN: 1572-9796. DOI: 10.1023/B:NACO.0000027755.02868.60.

Bojarski, M., D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba (2016). "End to End Learning for Self-Driving Cars". In: *Computing Research Repository (CoRR)* abs/1604.07316. URL: http://arxiv.org/abs/1604.07316.

Bonnin, S., F. Kummert, and J. Schmüdderich (2012). "A Generic Concept of a System for Predicting Driving Behaviors". In: *2012 15th International IEEE Conference on Intelligent Transportation Systems*, pp. 1803–1808. DOI: 10.1109/ITSC.2012.6338695.

Bracewell, R. (2000). *The Fourier Transform and Its Applications*. Electrical engineering series. McGraw Hill. ISBN: 9780073039381. URL: https://books.google.de/books?id=ZNQQAQAAIAAJ.

Brandli, C., R. Berner, M. Yang, S.-C. Liu, and T. Delbruck (2014). "A 240×180 130 dB 3 μs Latency Global Shutter Spatiotemporal Vision Sensor". In: *IEEE Journal of Solid-State Circuits* 49.10, pp. 2333–2341. ISSN: 0018-9200. DOI: 10.1109/JSSC.2014.2342715.

Brette, R., M. Rudolph, T. Carnevale, M. Hines, D. Beeman, J. Bower, M. Diesmann, A. Morrison, P. Goodman, F. Harris, M. Zirpe, T. Natschläger, D. Pecevski, B. Ermentrout, M. Djurfeldt, A. Lansner, O. Rochel, T. Vieville, E. Muller, A. Davison, S. El Boustani, and A. Destexhe (2007). "Simulation of networks of spiking neurons: A review of tools and strategies". In: *Journal of Computational Neuroscience* 23.3, pp. 349–398.

Brooks, R. (1986). "A robust layered control system for a mobile robot". In: *IEEE Journal on Robotics and Automation* 2.1, pp. 14–23. ISSN: 0882-4967. DOI: 10.1109/JRA.1986.1087032.

Buehler, M., K. Iagnemma, and S. Singh (2009). *The DARPA Urban Challenge: Autonomous Vehicles in City Traffic*. 1st ed. Springer Publishing Company, Incorporated.

Calimera, A., E. Macii, and M. Poncino (2013). "The Human Brain Project and neuromorphic computing". In: *Functional Neurology* 28.3, pp. 191–196.

Carlson, K. D., M. Beyeler, N. Dutt, and J. L. Krichmar (2014). "GPGPU accelerated simulation and parameter tuning for neuromorphic applications". In: *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, pp. 570–577. DOI: 10.1109/aspdac.2014.6742952.

Carnevale, N. T. and M. L. Hines (2009). *The NEURON Book*. 1st ed. New York, NY, USA: Cambridge University Press. ISBN: 9780521115636.

Cassidy, A. S., P. Merolla, J. V. Arthur, S. K. Esser, B. Jackson, R. Alvarez-Icaza, P. Datta, J. Sawada, T. M. Wong, V. Feldman, A. Amir, D. B. D. Rubin, F. Akopyan, E. McQuinn, W. P. Risk, and D. S. Modha (2013). "Cognitive computing building block: A versatile and efficient digital neuron model for neurosynaptic cores". In: *The 2013 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–10.

Cauwenberghs, G. (1998). "Neuromorphic Learning VLSI Systems: A Survey". In: *The Springer International Series in Engineering and Computer Science*. Ed. by T. S. Lande. Boston, MA: Springer US, pp. 381–408. ISBN: 978-0-585-28001-1. DOI: 10.1007/978-0-585-28001-1_17.

Censi, A. and D. Scaramuzza (2014). "Low-latency event-based visual odometry". In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. DOI: 10.1109/icra.2014.6906931. URL: http://purl.org/censi/2013/dvsd.

Censi, A., J. Strubel, C. Brandli, T. Delbrück, and D. Scaramuzza (2013). "Low-latency localization by Active LED Markers tracking using a Dynamic Vision Sensor". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Tokyo,Japan, pp. 891–898. DOI: 10.1109/IROS.2013.6696456.

Chen, G., H. Cao, M. Aafaque, J. Chen, C. Ye, F. Röhrbein, J. Conradt, K. Chen, Z. Bing, X. Liu, G. Hinz, W. Stechele, and A. Knoll (2018). "Neuromorphic Vision Based Multivehicle Detection and Tracking for Intelligent Transportation System". In: *Journal of Advanced Transportation* 2018, p. 13. DOI: 10.1155/2018/4815383.

Chen, J., S. Sathe, C. Aggarwal, and D. Turaga (2017). "Outlier Detection with Autoencoder Ensembles". In: *Proceedings of the 2017 SIAM International Conference on Data Mining*. Society for Industrial and Applied Mathematics, pp. 90–98. DOI: 10.1137/1.9781611974973.11. eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9781611974973.11.

Chollet, F. (2015). *Keras*. URL: https://github.com/keras-team/keras (visited on 2018-04-11).

Choudhary, S., S. Sloan, S. Fok, A. Neckar, E. Trautmann, P. Gao, T. Stewart, C. Eliasmith, and K. Boahen (2012). "Silicon Neurons That Compute". In: *Artificial Neural Networks and Machine Learning – ICANN 2012*. Ed. by A. E. P. Villa, W. Duch, P. Érdi, F. Masulli, and G. Palm. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 121–128. ISBN: 978-3-642-33269-2. DOI: 10.1007/978-3-642-33269-2_16.

Ciresan, D. C., U. Meier, J. Masci, and J. Schmidhuber (2012a). "Multi-column deep neural network for traffic sign classification". In: *Neural Networks* 32, pp. 333–338. DOI: 10.1016/j.neunet.2012.02.023.

Ciresan, D. C., U. Meier, and J. Schmidhuber (2012b). "Multi-column deep neural networks for image classification". In: *2012 IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, USA, June 16-21, 2012*, pp. 3642–3649. DOI: 10.1109/CVPR.2012.6248110.

Colyar, J. and J. Halkias (2018-12-10). *US Highway 101 Dataset*. URL: https://www.fhwa.dot.gov/publications/research/operations/07030/index.cfm (visited on 2018-04-03).

Conradt, J., M. Cook, R. Berner, P. Lichtsteiner, R. J. Douglas, and T. Delbrück (2009). "A Pencil Balancing Robot using a Pair of AER Dynamic Vision Sensors." In: *ISCAS*. IEEE, pp. 781–784. DOI: 10.1109/ISCAS.2009.5117867.

Conradt, J., F. Galluppi, and T. C. Stewart (2015). "Trainable sensorimotor mapping in a neuromorphic robot". In: *Robotics and Autonomous Systems* 71. Emerging Spatial Competences: From Machine Perception to Sensorimotor Intelligence, pp. 60–68. ISSN: 0921-8890. DOI: 10.1016/j.robot.2014.11.004. URL: http://www.sciencedirect.com/science/article/pii/S0921889014002462.

Cordts, M., M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele (2016). "The Cityscapes Dataset for Semantic Urban Scene Understanding". In: *Computing Research Repository (CoRR)* abs/1604.01685. URL: http://arxiv.org/abs/1604.01685.

Cortical.io (2019). *Cortical.io webpage*. URL: https://www.cortical.io/ (visited on 2019-04-03).

Crawford, E., M. Gingerich, and C. Eliasmith (2016). "Biologically Plausible, Human-Scale Knowledge Representation". In: *Cognitive Science* 40.4, pp. 782–821. ISSN: 1551-6709. DOI: 10.1111/cogs.12261.

Cui, Y., S. Ahmad, and J. Hawkins (2017). "The HTM Spatial Pooler - A Neocortical Algorithm for Online Sparse Distributed Coding". In: *Frontiers in Computational Neuroscience* 11, p. 111. ISSN: 1662-5188. DOI: 10.3389/fncom.2017.00111.

D'Agostino, C., A. Saidi, G. Scouarnec, and L. Chen (2013). "Learning-based driving events classification". In: *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*. IEEE, pp. 1778–1783. DOI: 10.1109/ITSC.2013.6728486.

Darius, R. (2018). "Learning a visual-semantic vector vocabulary for automotive environment modelling". MA thesis. Technical University of Munich.

Darms, M., P. Rybski, and C. Urmson (2008). "Classification and tracking of dynamic objects with multiple sensors for autonomous driving in urban environments". In: *Intelligent Vehicles Symposium, 2008 IEEE*, pp. 1197–1202. DOI: 10.1109/IVS.2008.4621259.

DARPA (2017). *DARPA SYNAPSE webpage*. URL: https://www.darpa.mil/program/systems-of-neuromorphic-adaptive-plastic-scalable-electronics (visited on 2017-12-20).

Davies, M., N. Srinivasa, T. H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C. K. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y. H. Weng, A. Wild, Y. Yang, and H. Wang (2018). "Loihi: A Neuromorphic Manycore Processor with On-Chip Learning". In: *IEEE Micro* 38.1, pp. 82–99. ISSN: 0272-1732.

Davies, S., C. Patterson, F. Galluppi, A. D. Rast, D. R. Lester, and S. Furber (2010). "Interfacing Real-Time Spiking I/O with the SpiNNaker Neuromimetic Architecture". In: *Australien Journal of Intelligent Information Processing Systems* 11.1.

Davis, R., R. Davis, and P. Szolovits (1993-03-15). "What is Knowledge Representation?" In: *AI Magazine* 14.1, pp. 17–33. DOI: 10.1609/aimag.v14i1.1029. URL: https://www.aaai.org/ojs/index.php/aimagazine/article/view/1029.

Davison, A. P., D. Brüderle, J. Eppler, J. Kremkow, E. Muller, D. Pecevski, L. Perrinet, and P. Yger (2008). "PyNN: A Common Interface for Neuronal Network Simulators". In: *Frontiers in neuroinformatics* 2, p. 11. ISSN: 1662-5196. DOI: 10.3389/neuro.11.011.2008.

Deisenroth, M. P., G. Neumann, and J. Peters (2013). "A Survey on Policy Search for Real-Time Robotics". In: *Foundations and Trends® in Robotics* 2.1-2, pp. 1–142. ISSN: 1935-8253. DOI: 10.1561/2300000021.

Delbruck, T. and M. Lang (2013). "Robotic Goalie with 3ms Reaction Time at 4% CPU Load Using Event-Based Dynamic Vision Sensor". In: *Frontiers in Neuroscience* 7.223. ISSN: 1662-453X. DOI: 10.3389/fnins.2013.00223.

Deng, J., W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei (2009). "ImageNet: A Large-Scale Hierarchical Image Database". In: *CVPR09*.

Denk, C., F. Llobet-Blandino, F. Galluppi, L. A. Plana, S. Furber, and J. Conradt (2013). "Real-Time Interface Board for Closed-Loop Robotic Tasks on the SpiNNaker Neural Computing System". In: *Artificial Neural Networks and Machine Learning - ICANN 2013 - 23rd International Conference on Artificial Neural Networks, Sofia, Bulgaria, September 10-13, 2013. Proceedings*, pp. 467–474. DOI: 10.1007/978-3-642-40728-4_59.

Deo, N. and M. M. Trivedi (2018a-05-15). "Convolutional Social Pooling for Vehicle Trajectory Prediction". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, 2018, pp. 1468-1476* abs/1805.06771. arXiv: http://arxiv.org/abs/1805.06771v1 [cs.CV]. URL: http://arxiv.org/abs/1805.06771.

Deo, N. and M. M. Trivedi (2018b). "Multi-Modal Trajectory Prediction of Surrounding Vehicles with Maneuver based LSTMs". In: *2018 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, pp. 1179–1184. DOI: 10.1109/ivs.2018.8500493.

Dethier, J., P. Nuyujukian, C. Eliasmith, T. C. Stewart, S. A. Elasaad, K. V. Shenoy, and K. A. Boahen (2011). "A Brain-Machine Interface Operating with a Real-Time Spiking Neural Network Control Algorithm". In: *Advances in Neural Information Processing Systems 24*. Ed. by J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger. Curran Associates, Inc., pp. 2213–2221. URL: http://papers.nips.cc/paper/4276-a-brain-machine-interface-operating-with-a-real-time-spiking-neural-network-control-algorithm.pdf.

DeWolf, T., T. C. Stewart, J.-J. Slotine, and C. Eliasmith (2016). "A spiking neural model of adaptive arm control". In: *Proceedings of the Royal Society of London B: Biological Sciences* 283.1843. ISSN: 0962-8452. DOI: 10.1098/rspb.2016.2134. eprint: http://rspb.royalsocietypublishing.org/content/283/1843/20162134.full.pdf. URL: http://rspb.royalsocietypublishing.org/content/283/1843/20162134.

Dickmanns, E. D., B. Mysliwetz, and T. Christians (1990). "An integrated spatio-temporal approach to automatic visual guidance of autonomous vehicles". In: *IEEE Transactions on Systems, Man, and Cybernetics* 20.6, pp. 1273–1284. ISSN: 0018-9472. DOI: 10.1109/21.61200.

Diehl, P. U. and M. Cook (2014). "Efficient implementation of STDP rules on SpiNNaker neuromorphic hardware". In: *2014 International Joint Conference on Neural Networks (IJCNN)*, pp. 4288–4295. DOI: 10.1109/IJCNN.2014.6889876.

Diehl, P. U., D. Neil, J. Binas, M. Cook, S.-C. Liu, and M. Pfeiffer (2015). "Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing". In: *2015 International Joint Conference on Neural Networks (IJCNN)*. IEEE, pp. 1–8. DOI: 10.1109/ijcnn.2015.7280696.

Drazen, D., P. Lichtsteiner, P. Hafliger, T. Delbruck, and A. Jensen (2011). "Toward real-time particle tracking using an event-based dynamic vision sensor". In: *Experiments in Fluids* 51.5, pp. 1465–1469.

Duan, Y., X. Chen, R. Houthooft, J. Schulman, and P. Abbeel (2016). "Benchmarking Deep Reinforcement Learning for Continuous Control". In: *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pp. 1329–1338. URL: http://jmlr.org/proceedings/papers/v48/duan16.html.

École Polytechnique Fédérale de Lausanne' (2017). *Blue Brain project webpage*. URL: http://bluebrain.epfl.ch (visited on 2017-12-20).

Edwards, C. (2015). "Growing Pains for Deep Learning". In: *Communications of the ACM* 58.7, pp. 14–16. ISSN: 0001-0782. DOI: 10.1145/2771283.

Elfring, J., R. Appeldoorn, S. van den Dries, and M. Kwakkernaat (2016). "Effective World Modeling: Multisensor Data Fusion Methodology for Automated Driving". In: *Sensors* 16.10, p. 1668. ISSN: 1424-8220. DOI: 10.3390/s16101668. URL: http://www.mdpi.com/1424-8220/16/10/1668.

Eliasmith, C. (2013). *How to build a brain: A neural architecture for biological cognition*. New York, NY: Oxford University Press.

Eliasmith, C. and C. H. Anderson (2003). *Neural Engineering : Computation, Representation, and Dynamics in Neurobiological Systems*. Computational neuroscience. Cambridge, Mass. MIT Press. ISBN: 0-262-05071-4.

Eliasmith, C., T. C. Stewart, X. Choo, T. Bekolay, T. DeWolf, Y. Tang, and D. Rasmussen (2012). "A Large-Scale Model of the Functioning Brain". In: *Science* 338.6111, pp. 1202–1205. ISSN: 0036-8075. DOI: 10.1126/science.1225266. eprint: http://science.sciencemag.org/content/338/6111/1202.full.pdf. URL: http://science.sciencemag.org/content/338/6111/1202.

Engstrom, J. and T. Victor (2001). "Real-time recognition of large-scale driving patterns". In: *ITSC 2001. 2001 IEEE Intelligent Transportation Systems. Proceedings (Cat. No.01TH8585)*, pp. 1018–1023. DOI: 10.1109/ITSC.2001.948801.

Esser, S. K., A. Andreopoulos, R. Appuswamy, P. Datta, D. Barch, A. Amir, J. V. Arthur, A. Cassidy, M. Flickner, P. Merolla, S. Chandra, N. Basilico, S. Carpin, T. Zimmerman, F. Zee, R. Alvarez-Icaza, J. A. Kusnitz, T. M. Wong, W. P. Risk, E. McQuinn, T. K. Nayak, R. Singh, and D. S. Modha (2013). "Cognitive computing systems: Algorithms and applications for networks of neurosynaptic cores". In: *The 2013 International Joint Conference on Neural Networks, IJCNN 2013, Dallas, TX, USA, August 4-9, 2013*, pp. 1–10. DOI: 10.1109/IJCNN.2013.6706746.

Farahini, N. (2016). "SiLago: Enabling System Level Automation Methodology to Design Custom High-Performance Computing Platforms : Toward Next Generation Hardware Synthesis Methodologies". PhD thesis. Stockholm: Royal Institute of Technology (KTH), School of Information and Communication Technology.

Feng, D., C. Haase-Schütz, H. Hertlein, F. Duffhauß, C. Gläser, and W. Wiesbeck (2019). "Deep Multimodal Object Detection and Semantic Segmentation for Autonomous Driving: Datasets, Methods, and Challenges". en. In: DOI: 10.13140/RG.2.2.10246.01606.

Fodor, J. A. (1975). *The Language of Thought*. Language and thought series. Harvard University Press. ISBN: 9780674510302.

Fong, R. and A. Vedaldi (2018). "Net2Vec: Quantifying and Explaining how Concepts are Encoded by Filters in Deep Neural Networks". In: *ArXiv e-prints*. arXiv: 1801.03454.

Freund, Y. and R. E. Schapire (1997). "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting". In: *Journal of Computer and System Sciences* 55.1, pp. 119–139. ISSN: 0022-0000. DOI: 10.1006/jcss.1997.1504.

Furber, S. B., F. Galluppi, S. Temple, and L. A. Plana (2014). "The SpiNNaker Project". In: *Proceedings of the IEEE* 102.5, pp. 652–665. ISSN: 0018-9219. DOI: 10.1109/jproc.2014.2304638.

Furgale, P., et al., U. Schwesinger, M. Rufli, C. Pradalier, R. Siegwart, K. Köser, C. Häne, L. Heng, G. H. Lee, F. Fraundorfer, and M. Pollefeys (2013). "Toward automated driving in cities using close-to-market sensors: An overview of the V-Charge Project". In: *Proceedings of 2013 IEEE Intelligent Vehicles Symposium (IV)*. Piscataway, NJ: IEEE, pp. 809–816.

Fuster, J. M. (2004). "Upper processing stages of the perception-action cycle". In: *Trends in Cognitive Sciences* 8.4, pp. 143–145. ISSN: 1364-6613. DOI: 10.1016/j.tics.2004.02.004. URL: http://www.sciencedirect.com/science/article/pii/S1364661304000476.

Gallant, S. I. and T. W. Okaywe (2013). "Representing Objects, Relations, and Sequences". In: *Neural Computation* 25.8, pp. 2038–2078. ISSN: 0899-7667. DOI: 10.1162/NECO_a_00467.

Gallego, G., T. Delbrück, G. Orchard, C. Bartolozzi, B. Taba, A. Censi, S. Leutenegger, A. J. Davison, J. Conradt, K. Daniilidis, and D. Scaramuzza (2019-04-17). "Event-based Vision: A Survey". In: *CoRR* abs/1904.08405. arXiv: http://arxiv.org/abs/1904.08405v1 [cs.CV]. URL: http://arxiv.org/abs/1904.08405.

Gallego, G., C. Forster, E. Mueggler, and D. Scaramuzza (2015). "Event-based Camera Pose Tracking using a Generative Event Model". In: *Computing Research Repository (CoRR)*. URL: http://arxiv.org/abs/1510.01972.

Galluppi, F., S. Davies, A. Rast, T. Sharp, L. A. Plana, and S. Furber (2012). "A Hierachical Configuration System for a Massively Parallel Neural Hardware Platform". In: *Proceedings of the 9th Conference on Computing Frontiers*. CF '12. Cagliari, Italy: ACM, pp. 183–192. ISBN: 978-1-4503-1215-8. DOI: 10.1145/2212908.2212934.

Galluppi, F., C. Denk, M. C. Meiner, T. C. Stewart, L. A. Plana, C. Eliasmith, S. B. Furber, and J. Conradt (2014). "Event-based neural computing on an autonomous mobile platform". In: *2014 IEEE International Conference on Robotics and Automation, ICRA 2014, Hong Kong, China, May 31 - June 7, 2014*, pp. 2862–2867. DOI: 10.1109/ICRA.2014.6907270.

Gayler, R. (1998). "Multiplicative Binding, Representation Operators and Analogy". In: *Advances in analogy research: Integration of theory and data from the cognitive, computational, and neural sciences*. Ed. by
bibinitperiod B. N. K. D. Gentner K. J. Holyoak. Sofia, Bulgaria: New Bulgarian University., pp. 1–4. URL: http://cogprints.org/502.

— (2003). "Vector Symbolic Architectures answer Jackendoff's challenges for cognitive neuroscience". In: *ICCS/ASCS International Conference on Cognitive Science*. Ed. by P. Slezak. University of New South Wales. CogPrints, pp. 133–138.

Geiger, A., P. Lenz, C. Stiller, and R. Urtasun (2013). "Vision meets robotics: The KITTI dataset". In: *The International Journal of Robotics Research* 32.11, pp. 1231–1237. DOI: 10.1177/0278364913491297. eprint: https://doi.org/10.1177/0278364913491297.

Georgopoulos, A., J. Lurito, M. Petrides, A. Schwartz, and J. Massey (1989). "Mental rotation of the neuronal population vector". In: *Science* 243.4888, pp. 234–236. ISSN: 0036-8075. DOI: 10.1126/science.2911737. eprint: http://science.sciencemag.org/content/243/4888/234.full.pdf. URL: http://science.sciencemag.org/content/243/4888/234.

Gerstner, W. and W. Kistler (2002). *Spiking Neuron Models: An Introduction*. New York, NY, USA: Cambridge University Press. ISBN: 0521890799.

Gerstner, W., W. Kistler, R. Naud, and L. Paninski (2014). *Neuronal Dynamics - From single neurons to networks and models of cognition*. Cambridge University Press. URL: http://neuronaldynamics.epfl.ch/online/index.html.

Gewaltig, M.-O. and M. Diesmann (2007). "NEST (NEural Simulation Tool)". In: *Scholarpedia* 2.4, p. 1430. DOI: 10.4249/scholarpedia.1430. URL: http://www.scholarpedia.org/article/NEST_(NEural_Simulation_Tool).

Goldberg, Y. and O. Levy (2014). "word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method". In: *Computing Research Repository (CoRR)*. URL: http://arxiv.org/abs/1402.3722.

Gomes, H. M., J. P. Barddal, F. Enembreck, and A. Bifet (2017). "A Survey on Ensemble Learning for Data Stream Classification". In: *ACM Computing Surveys* 50.2, pp. 1–36. DOI: 10.1145/3054925.

Goodman, D. F. M. and R. Brette (2009). "The Brian simulator". In: *Frontiers in Neuroscience* 3.26, pp. 192–197. ISSN: 1662-453X. DOI: 10.3389/neuro.01.026.2009.

Gordon, N. J., D. J. Salmond, and A. F. M. Smith (1993). "Novel approach to nonlinear/non-Gaussian Bayesian state estimation". In: *IEE Proceedings F Radar and Signal Processing* 140.2, pp. 107–113. ISSN: 0956-375X. DOI: 10.1049/ip-f-2.1993.0015.

Graf, R., H. Deusch, F. Seeliger, M. Fritzsche, and K. Dietmayer (2014). "A Learning Concept for Behavior Prediction at Intersections". In: *2014 IEEE Intelligent Vehicles Symposium Proceedings*, pp. 939–945. DOI: 10.1109/IVS.2014.6856415.

Guizzo, E. and E. Ackerman (2015). "The hard lessons of DARPA's robotics challenge [News]". In: *IEEE Spectrum* 52.8, pp. 11–13. ISSN: 0018-9235. DOI: 10.1109/MSPEC.2015.7164385.

Gupta, N. and M. Stopfer (2014). "A temporal channel for information in sparse sensory coding". In: *Current biology* 24.19, pp. 2247–2256. ISSN: 1879-0445. URL: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC4189991/.

Hallac, D., S. Bhooshan, M. Chen, K. Abida, R. Sosic, and J. Leskovec (2018). "Drive2Vec: Multiscale State-Space Embedding of Vehicular Sensor Data". In: *21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, pp. 3233–3238. DOI: 10.1109/ITSC.2018.8569550.

Handjaras, G., E. Ricciardi, A. Leo, A. Lenci, L. Cecchetti, M. Cosottini, G. Marotta, and P. Pietrini (2016). "How concepts are encoded in the human brain: A modality independent, category-based cortical organization of semantic knowledge". In: *NeuroImage* 135, pp. 232–242. ISSN: 1053-8119.

DOI: 10.1016/j.neuroimage.2016.04.063. URL: http://www.sciencedirect.com/science/article/pii/S1053811916301021.

Hauptmann, W., F. Graf, and K. Heesche (1996). "Driving environment recognition for adaptive automotive systems". In: *Proceedings of IEEE 5th International Fuzzy Systems*. Vol. 1, pp. 387–393. DOI: 10.1109/FUZZY.1996.551772.

He, Z. (2017). *Research based on high-fidelity NGSIM vehicle trajectory datasets: A review*. Tech. rep. DOI: 10.13140/RG.2.2.11429.60643.

Hebb, D. O. (1949). *The Organization of Behavior*. John Wiley.

Held, D., J. Levinson, and S. Thrun (2012). "A probabilistic framework for car detection in images using context and scale". In: *IEEE International Conference on Robotics and Automation, ICRA 2012, 14-18 May, 2012, St. Paul, Minnesota, USA*, pp. 1628–1634. DOI: 10.1109/ICRA.2012.6224722.

Hermann, A. and J. Desel (2008). "Driving situation analysis in automotive environment". In: *IEEE International Conference on Vehicular Electronics and Safety (ICVES)*, pp. 216–221. DOI: 10.1109/ICVES.2008.4640860.

Hochreiter, S. and J. Schmidhuber (1997). "Long Short-Term Memory". In: *Neural Computation* 9.8, pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735. eprint: https://doi.org/10.1162/neco.1997.9.8.1735.

Hodgkin, A. and A. Huxley (1952). "A quantitative description of membrane current and its application to conduction and excitation in nerve". In: *Journal of Physiology* 117, pp. 500–544.

Hohm, A., F. Lotz, O. Fochler, S. Lüke, and H. Winner (2014). "Automated Driving in Real Traffic: from Current Technical Approaches towards Architectural Perspectives". In: *SAE Technical Papers* 1. DOI: 10.4271/2014-01-0159.

Hoi, S. C. H., D. Sahoo, J. Lu, and P. Zhao (2018-02-08). "Online Learning: A Comprehensive Survey". In: *CoRR* abs/1802.02871. arXiv: 1802.02871 [cs.LG]. URL: http://arxiv.org/abs/1802.02871.

Holler, Tam, Castro, and Benson (1989). "An Electrically Trainable Artificial Neural Network (ETANN) with 10240 "Floating Gate" Synapses". In: *International Joint Conference on Neural Networks*. Ed. by N. Morgan. Piscataway, NJ, USA: IEEE, pp. 50–55. ISBN: 0-8186-2029-3. DOI: 10.1109/ijcnn.1989.118698. URL: https://ieeexplore.ieee.org/document/118698/.

Hsu, F.-H. (2002). *Behind Deep Blue: Building the Computer That Defeated the World Chess Champion*. Princeton, NJ, USA: Princeton University Press. ISBN: 0691090653.

Huang, X., X. Cheng, Q. Geng, B. Cao, D. Zhou, P. Wang, Y. Lin, and R. Yang (2018). "The ApolloScape Dataset for Autonomous Driving". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pp. 954–960.

Human Brain Project (2018). *Human Brain Project webpage*. URL: https://www.humanbrainproject.eu (visited on 2018-04-05).

Hunsberger, E. and C. Eliasmith (2015). "Spiking Deep Networks with LIF Neurons". In: *Computing Research Repository (CoRR)* abs/1510.08829. URL: http://arxiv.org/abs/1510.08829.

— (2016). "Training Spiking Deep Networks for Neuromorphic Hardware". In: *Computing Research Repository (CoRR)* abs/1611.05141. arXiv: 1611.05141. URL: http://arxiv.org/abs/1611.05141.

Huval, B., T. Wang, S. Tandon, J. Kiske, W. Song, J. Pazhayampallil, M. Andriluka, P. Rajpurkar, T. Migimatsu, R. Cheng-Yue, F. Mujica, A. Coates, and A. Y. Ng (2015). "An Empirical Evaluation of Deep Learning on Highway Driving". In: *Computing Research Repository (CoRR)* abs/1504.01716. URL: http://arxiv.org/abs/1504.01716.

Indiveri, G. (1997). "Neuromorphic Systems For Industrial Applications". In: *Proc.International Body Engineering Conference & Exposition*. Stuttgart, Germany. URL: http://ncs.ethz.ch/pubs/pdf/Indiveri97.pdf.

Izhikevich, E. M. (2003). "Simple model of spiking neurons". In: *IEEE Transactions on Neural Networks* 14.6, pp. 1569–1572. ISSN: 1045-9227. DOI: 10.1109/TNN.2003.820440. URL: http://www.izhikevich.org/publications/spikes.pdf.

— (2004). "Which Model to Use for Cortical Spiking Neurons?" In: *IEEE Transactions on Neural Networks* 15.5, pp. 1063–1070. ISSN: 1045-9227. DOI: 10.1109/TNN.2004.832719. URL: http://www.izhikevich.org/publications/whichmod.pdf.

Jackendoff, R. (2002). *Foundations of Language: Brain, Meaning, Grammar, Evolution*. Oxford scholarship online. Oxford University Press. ISBN: 9780198270126. URL: https://books.google.de/books?id=gtGliq-q2aMC.

Janai, J., F. Güney, A. Behl, and A. Geiger (2017). "Computer Vision for Autonomous Vehicles: Problems, Datasets and State-of-the-Art". In: *ArXiv e-prints*. arXiv: 1704.05519 [cs.CV].

Kalman, R. E. (1960). "A New Approach to Linear Filtering and Prediction Problems". In: *Journal of Basic Engineering* 82.1, pp. 35–45. ISSN: 0098-2202. DOI: 10.1115/1.3662552.

Kanerva, P. (1988). *Sparse Distributed Memory*. Cambridge, MA, USA: MIT Press. ISBN: 0262111322.

— (2000). "Large Patterns Make Great Symbols: An Example of Learning from Example". In: *Hybrid Neural Systems*. Ed. by S. Wermter and R. Sun. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 194–203. ISBN: 978-3-540-46417-4. DOI: 10.1007/10719871_13.

— (2009). "Hyperdimensional Computing: An Introduction to Computing in Distributed Representation with High-Dimensional Random Vectors". In: *Cognitive Computation* 1.2, pp. 139–159. DOI: 10.1007/s12559-009-9009-8.

Karlsruhe Institute of Technology (2018). *Annieway Project*. URL: http://www.mrt.kit.edu/annieway/ (visited on 2018-02-22).

Karpathy, A. and L. Fei-Fei (2017). "Deep Visual-Semantic Alignments for Generating Image Descriptions". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.4, pp. 664–676. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2016.2598339.

Kieras, D. E. and D. E. Meyer (1997). "An Overview of the EPIC Architecture for Cognition and Performance With Application to Human-Computer Interaction". In: *Human-Computer Interaction* 12.4, pp. 391–438. ISSN: 0737-0024. DOI: 10.1207/s15327051hci1204_4.

Kim, H., A. Handa, R. Benosman, S.-H. Ieng, and A. J. Davison (2014). "Simultaneous Mosaicing and Tracking with an Event Camera". In: *British Machine Vision Conference, BMVC 2014, Nottingham, UK, September 1-5, 2014*.

Kirchhoff-Institute for Physics, Heidelberg University (2018a). *BrainScaleS-project*. URL: https://brainscales.kip.uni-heidelberg.de (visited on 2018-04-05).

— (2018b). *FACETS-project*. URL: https://facets.kip.uni-heidelberg.de (visited on 2018-04-05).

Kleyko, D., E. Osipov, R. W. Gayler, A. I. Khan, and A. G. Dyer (2015). "Imitation of honey bees' concept learning processes using Vector Symbolic Architectures". In: *Biologically Inspired Cognitive Architectures* 14, pp. 57–72. ISSN: 2212-683X. DOI: 10.1016/j.bica.2015.09.002. URL: http://www.sciencedirect.com/science/article/pii/S2212683X15000456.

Koopman, P. and M. Wagner (2016). "Challenges in Autonomous Vehicle Testing and Validation". In: *SAE International Journal of Transportation Safety* 4, pp. 15–24. DOI: 10.4271/2016-01-0128.

Krichmar, J. L., N. Dutt, J. M. Nageswaran, and M. Richert (2011). "Neuromorphic Modeling Abstractions and Simulation of Large-scale Cortical Networks". In: *Proceedings of the International Conference on Computer-Aided Design*. ICCAD '11. San Jose, California: IEEE Press, pp. 334–338. ISBN: 978-1-4577-1398-9. URL: http://dl.acm.org/citation.cfm?id=2132325.2132411.

Lagorce, X., C. Meyer, S. H. Ieng, D. Filliat, and R. Benosman (2015). "Asynchronous Event-Based Multikernel Algorithm for High-Speed Visual Features Tracking". In: *IEEE Transactions on Neural*

Networks and Learning Systems 26.8, pp. 1710–1720. ISSN: 2162-237X. DOI: 10.1109/TNNLS.2014.2352401.

Laird, J. E., A. Newell, and P. S. Rosenbloom (1987). "SOAR: An architecture for general intelligence". In: *Artificial Intelligence* 33.1, pp. 1–64. ISSN: 0004-3702. DOI: 10.1016/0004-3702(87)90050-6. URL: http://www.sciencedirect.com/science/article/pii/0004370287900506.

LeCun, Y., L. Bottou, Y. Bengio, and P. Haffner (1998). "Gradient-Based Learning Applied to Document Recognition". In: *Proceedings of the IEEE* 86.11, pp. 2278–2324.

LeCun, Y., Y. Bengio, and G. Hinton (2015). "Deep learning". In: *Nature* 521.7553, pp. 436–444. ISSN: 0028-0836.

Lefèvre, S., D. Vasquez, and C. Laugier (2014-07-23). "A survey on motion prediction and risk assessment for intelligent vehicles". In: *ROBOMECH Journal* 1.1, p. 1. ISSN: 2197-4225. DOI: 10.1186/s40648-014-0001-z.

Levine, S., C. Finn, T. Darrell, and P. Abbeel (2016). "End-to-End Training of Deep Visuomotor Policies". In: *Journal of Machine Learning Research* 17.1, pp. 1334–1373. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=2946645.2946684.

Levy, O., Y. Goldberg, and I. Dagan (2015). "Improving Distributional Similarity with Lessons Learned from Word Embeddings". In: *Transactions of the Association for Computational Linguistics* 3, pp. 211–225. ISSN: 2307-387X. URL: https://transacl.org/ojs/index.php/tacl/article/view/570.

Levy, S. D. and R. Gayler (2008). "Vector Symbolic Architectures: A New Building Material for Artificial General Intelligence". In: *Proceedings of the 2008 Conference on Artificial General Intelligence 2008: Proceedings of the First AGI Conference*. Amsterdam, The Netherlands, The Netherlands: IOS Press, pp. 414–418. ISBN: 978-1-58603-833-5. URL: http://dl.acm.org/citation.cfm?id=1566174.1566215.

Li, H., F. Sun, L. Liu, and L. Wang (2015). "A novel traffic sign detection method via color segmentation and robust shape matching". In: *Neurocomputing* 169. Learning for Visual Semantic Understanding in Big DataESANN 2014Industrial Data Processing and AnalysisSelected papers from the 22nd European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN 2014)Selected papers from the 11th World Congress on Intelligent Control and Automation (WCICA2014), pp. 77–88. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2014.12.111. URL: http://www.sciencedirect.com/science/article/pii/S0925231215006712.

Li, X., L. Li, F. Flohr, J. Wang, H. Xiong, M. Bernhard, S. Pan, D. M. Gavrila, and K. Li (2017). "A Unified Framework for Concurrent Pedestrian and Cyclist Detection". In: *IEEE Transactions on Intelligent Transportation Systems* 18.2, pp. 269–281. ISSN: 1524-9050. DOI: 10.1109/TITS.2016.2567418.

Lichtsteiner, P., C. Posch, and T. Delbruck (2008). "A 128x128 120 dB 15 μs Latency Asynchronous Temporal Contrast Vision Sensor". In: *IEEE Journal of Solid-State Circuits* 43.2, pp. 566–576. ISSN: 0018-9200. DOI: 10.1109/JSSC.2007.914337.

Lin, T.-Y., M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár (2014-05-01). "Microsoft COCO: Common Objects in Context". In: *arXiv e-prints*, arXiv:1405.0312, arXiv:1405.0312. arXiv: http://arxiv.org/abs/1405.0312v3[cs.CV].

Liu, S.-C., J. Kramer, G. Indiveri, T. Delbruck, and R. Douglas (2002). *Analog VLSI:Circuits and Principles*. MIT Press.

Liu, S.-C. and T. Delbruck (2010). "Neuromorphic sensory systems". In: *Current Opinion in Neurobiology* 20.3, pp. 288–295. ISSN: 0959-4388. DOI: 10.1016/j.conb.2010.03.007.

Liu, S.-C., A. van Schaik, B. A. Minch, and T. Delbruck (2014). "Asynchronous Binaural Spatial Audition Sensor With 2 x 64 x 4 Channel Output". In: *IEEE Transactions on Biomedical Circuits and Systems* 8.4, pp. 453–464. ISSN: 1932-4545. DOI: 10.1109/tbcas.2013.2281834.

Loeb, G. E. and J. A. Fishel (2014). "Bayesian Action&Perception: Representing the World in the Brain". In: *Frontiers in Neuroscience* 8, p. 341. ISSN: 1662-453X. DOI: 10.3389/fnins.2014.00341.

Loiacono, D., P. L. Lanzi, J. Togelius, E. Onieva, D. A. Pelta, M. V. Butz, T. D. Lonneker, L. Cardamone, D. Perez, Y. Saez, M. Preuss, and J. Quadflieg (2010). "The 2009 Simulated Car Racing Championship". In: *IEEE Transactions on Computational Intelligence and AI in Games* 2.2, pp. 131–147. ISSN: 1943-068X. DOI: 10.1109/TCIAIG.2010.2050590.

Long, J., E. Shelhamer, and T. Darrell (2015). "Fully convolutional networks for semantic segmentation". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, pp. 3431–3440. DOI: 10.1109/CVPR.2015.7298965.

Losing, V., B. Hammer, and H. Wersing (2017). "Personalized maneuver prediction at intersections". In: *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, pp. 1–6. DOI: 10.1109/ITSC.2017.8317760.

— (2018). "Incremental on-line learning: A review and comparison of state of the art algorithms". In: *Neurocomputing* 275, pp. 1261–1274. DOI: 10.1016/j.neucom.2017.06.084.

Lundgren, M., E. Stenborg, L. Svensson, and L. Hammarstrand (2014). "Vehicle self-localization using off-the-shelf sensors and a detailed map". In: *2014 IEEE Intelligent Vehicles Symposium Proceedings*. IEEE, pp. 522–528. DOI: 10.1109/IVS.2014.6856524.

Maass, W. (1997). "Networks of Spiking Neurons: The Third Generation of Neural Network Models". In: *Neural Networks* 14.4, pp. 1659–1671. ISSN: 0893-6080. DOI: 10.1016/S0893-6080(97)00011-7. URL: http://www.sciencedirect.com/science/article/pii/S0893608097000117.

Mahowald, M. (1992). "VLSI analogs of neuronal visual processing: a synthesis of form and function". PhD thesis. California Institute of Technology.

Marr, B., B. Degnan, P. Hasler, and D. Anderson (2013). "Scaling Energy Per Operation via an Asynchronous Pipeline". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 21.1, pp. 147–151. ISSN: 1063-8210. DOI: 10.1109/TVLSI.2011.2178126.

Masquelier, T., R. Guyonneau, and S. J. Thorpe (2007). "Spike Timing Dependent Plasticity Finds the Start of Repeating Patterns in Continuous Spike Trains". In: *PLoS ONE* 3.1. Ed. by O. Sporns, e1377. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0001377. URL: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2147052/.

Maye, J., R. Triebel, L. Spinello, and R. Siegwart (2011). "Bayesian On-line Learning of Driving Behaviors". In: *Proc. of The International Conference in Robotics and Automation (ICRA)*.

McCulloch, W. S. and W. Pitts (1943). "A logical calculus of the ideas immanent in nervous activity". In: *The Bulletin of Mathematical Biophysics* 5.4, pp. 115–133. DOI: 10.1007/bf02478259.

Mead, C. (1989). *Analog VLSI and Neural Systems*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc. ISBN: 0-201-05992-4.

— (1990). "Neuromorphic electronic systems". In: *Proceedings of the IEEE*. Vol. 78. 10, pp. 1629–1636.

Merolla, P., J. Arthur, F. Akopyan, N. Imam, R. Manohar, and D. S. Modha (2011). "A digital neurosynaptic core using embedded crossbar memory with 45pJ per spike in 45nm". In: *2011 IEEE Custom Integrated Circuits Conference (CICC)*, pp. 1–4. DOI: 10.1109/CICC.2011.6055294.

Merolla, P., J. Arthur, R. Alvarez, J. M. Bussat, and K. Boahen (2014). "A Multicast Tree Router for Multichip Neuromorphic Systems". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 61.3, pp. 820–833. ISSN: 1549-8328. DOI: 10.1109/TCSI.2013.2284184.

Mikolov, T., K. Chen, G. Corrado, and J. Dean (2013a). "Efficient Estimation of Word Representations in Vector Space". In: *Computing Research Repository (CoRR)* abs/1301.3781. URL: http://arxiv.org/abs/1301.3781.

Mikolov, T., I. Sutskever, K. Chen, G. S. Corrado, and J. Dean (2013b). "Distributed Representations of Words and Phrases and their Compositionality". In: *Advances in Neural Information Processing Systems 26*. Ed. by C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger. Curran Associates, Inc., pp. 3111–3119. URL: http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf.

Mikolov, T., S. W.-t. Yih, and G. Zweig (2013c). "Linguistic Regularities in Continuous Space Word Representations". In: *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT-2013)*. Association for Computational Linguistics. URL: https://www.microsoft.com/en-us/research/publication/linguistic-regularities-in-continuous-space-word-representations.

Miller, G. A. (1956). "The magical number seven, plus or minus two: some limits on our capacity for processing information". In: *Psychological Review* 63.2, pp. 81–97. DOI: 10.1037/h0043158.

Minsky, M. and S. Papert (1969). *Perceptrons*. Cambridge, MA: MIT Press.

Minsky, M. (1986). *The Society of Mind*. New York, NY, USA: Simon and Schuster, Inc. ISBN: 0-671-60740-5.

Mirus, F., C. Axenie, T. C. Stewart, and J. Conradt (2018a). "Neuromorphic sensorimotor adaptation for robotic mobile manipulation: From sensing to behaviour". In: *Cognitive Systems Research* 50, pp. 52–66. ISSN: 1389-0417. DOI: 10.1016/j.cogsys.2018.03.006. URL: http://www.sciencedirect.com/science/article/pii/S1389041717300955.

Mirus, F., P. Blouw, T. C. Stewart, and J. Conradt (2019a-10). "An Investigation of Vehicle Behavior Prediction Using a Vector Power Representation to Encode Spatial Positions of Multiple Objects and Neural Networks". In: *Frontiers in Neurorobotics* 13, p. 84. ISSN: 1662-5218. DOI: 10.3389/fnbot.2019.00084. URL: https://www.frontiersin.org/article/10.3389/fnbot.2019.00084.

— (2019b). "Predicting vehicle behaviour using LSTMs and a vector power representation for spatial positions". In: *27th European Symposium on Artificial Neural Networks, ESANN 2019, Bruges, Belgium*, pp. 113–118.

Mirus, F., T. C. Stewart, and J. Conradt (2018b). "Towards cognitive automotive environment modelling: reasoning based on vector representations". In: *26th European Symposium on Artificial Neural Networks, ESANN 2018, Bruges, Belgium*, pp. 55–60.

— (2020a-07-19). "Analyzing the Capacity of Distributed Vector Representations to Encode Spatial Information". In: *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, pp. 1–7. DOI: 10.1109/IJCNN48605.2020.9207137.

— (2020b-10-02). "Detection of abnormal driving situations using distributed representations and unsupervised learning". In: *28th European Symposium on Artificial Neural Networks, ESANN 2020, Bruges, Belgium*.

— (2020c-07-19). "The Importance of Balanced Data Sets: Analyzing a Vehicle Trajectory Prediction Model based on Neural Networks and Distributed Representations". In: *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, pp. 1–8. DOI: 10.1109/IJCNN48605.2020.9206627.

Mirus, F., T. C. Stewart, C. Eliasmith, and J. Conradt (2019c). "A Mixture-of-Experts Model for Vehicle Prediction Using an Online Learning Approach". In: *Artificial Neural Networks and Machine Learning – ICANN 2019: Image Processing*. Ed. by I. V. Tetko, V. Kůrková, P. Karpov, and F. Theis. Vol. 11729. Lecture Notes in Computer Science. Springer International Publishing, pp. 456–471. ISBN: 978-3-030-30508-6. DOI: 10.1007/978-3-030-30508-6_37.

Mirus, F., B. Zorn, and J. Conradt (2019d). "Short-term trajectory planning using reinforcement learning within a neuromorphic control architecture". In: *27th European Symposium on Artificial Neural Networks, ESANN 2019, Bruges, Belgium*, pp. 649–654.

Mohan, R. (2014). "Deep Deconvolutional Networks for Scene Parsing". In: *ArXiv e-prints*. arXiv: 1411.4101 [stat.ML].

Moravec, H. (1988). *Mind Children: The Future of Robot and Human Intelligence*. Cambridge, MA, USA: Harvard University Press. ISBN: 0-674-57616-0.

Mueggler, E., B. Huber, and D. Scaramuzza (2014). "Event-based, 6-DOF pose tracking for high-speed maneuvers". In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, Chicago, IL, USA, September 14-18, 2014*, pp. 2761–2768. DOI: 10.1109/IROS.2014.6942940.

Mundy, A., J. Knight, T. C. Stewart, and S. Furber (2015). "An efficient SpiNNaker implementation of the Neural Engineering Framework". In: *2015 International Joint Conference on Neural Networks (IJCNN)*. IEEE, pp. 1–8. DOI: 10.1109/ijcnn.2015.7280390.

Navaridas, J., M. Luján, J. Miguel-Alonso, L. A. Plana, and S. Furber (2009). "Understanding the Interconnection Network of SpiNNaker". In: *Proceedings of the 23rd International Conference on Supercomputing*. ICS '09. Yorktown Heights, NY, USA: ACM, pp. 286–295. ISBN: 978-1-60558-498-0. DOI: 10.1145/1542275.1542317.

Neckar, A., S. Fok, B. V. Benjamin, T. C. Stewart, N. N. Oza, A. R. Voelker, C. Eliasmith, R. Manohar, and K. Boahen (2019). "Braindrop: A Mixed-Signal Neuromorphic Architecture With a Dynamical Systems-Based Programming Model". In: *Proceedings of the IEEE* 107.1 (1), pp. 144–164. DOI: 10.1109/jproc.2018.2881432. URL: https://ieeexplore.ieee.org/document/8591981.

Nere, A., U. Olcese, D. Balduzzi, and G. Tononi (2012). "A Neuromorphic Architecture for Object Recognition and Motion Anticipation Using Burst-STDP". In: *PLoS ONE* 7.5. Ed. by T. Wennekers, pp. 1–17. DOI: 10.1371/journal.pone.0036958.

Neubert, P., S. Schubert, and P. Protzel (2016). "Learning Vector Symbolic Architectures for Reactive Robot Behaviours". In: *Proc. of International Conference on Intelligent Robots and Systems (IROS), Workshop on Machine Learning Methods for High-Level Cognitive Capabilities in Robotics*.

Neumann, J. von (1993). "First draft of a report on the EDVAC". In: *IEEE Annals of the History of Computing* 15.4, pp. 27–75. ISSN: 1058-6180. DOI: 10.1109/85.238389.

Norton, A., W. Ober, L. Baraniecki, E. McCann, J. Scholtz, D. Shane, A. Skinner, R. Watson, and H. Yanco (2017). "Analysis of human-robot interaction at the DARPA Robotics Challenge Finals". In: *The International Journal of Robotics Research* 36.5-7, pp. 483–513. DOI: 10.1177/0278364916688254. eprint: https://doi.org/10.1177/0278364916688254.

Numenta (2019). *Numenta webpage*. URL: https://numenta.com/ (visited on 2019-04-03).

O'Connor, P., D. Neil, S.-C. Liu, T. Delbruck, and M. Pfeiffer (2013). "Real-Time Classification and Sensor Fusion with a Spiking Deep Belief Network". In: *Frontiers in Neuroscience* 7.178. ISSN: 1662-453X. DOI: 10.3389/fnins.2013.00178.

Olshausen, B. A. and D. J. Field (1996). "Emergence of Simple-Cell Receptive Field Properties by Learning a Sparse Code for Natural Images". In: *Nature* 381, pp. 607–609.

Orchard, G., A. Jayawant, G. Cohen, and N. V. Thakor (2015). "Converting Static Image Datasets to Spiking Neuromorphic Datasets Using Saccades". In: *Frontiers in Neuroscience* 9.00437. DOI: 10.3389/fnins.2015.00437. URL: http://www.frontiersin.org/Journal/Abstract.aspx?s=755&name=neuromorphic_engineering&ART_DOI=10.3389/fnins.2015.00437.

Painkras, E., L. A. Plana, J. Garside, S. Temple, F. Galluppi, C. Patterson, D. R. Lester, A. D. Brown, and S. B. Furber (2013). "SpiNNaker: A 1-W 18-Core System-on-Chip for Massively-Parallel Neural Network Simulation". In: *IEEE Journal of Solid-State Circuits* 48.8, pp. 1943–1953. ISSN: 0018-9200.

Patterson, E. K., S. Gurbuz, Z. Tufekci, and J. N. Gowdy (2002). "CUAVE: A new audio-visual database for multimodal human-computer interface research". In: *IEEE International Conference on Acous-*

tics Speech and Signal Processing. Vol. 2. IEEE, pp. II-2017-II–2020. DOI: 10.1109/icassp.2002.5745028.

Paugam-Moisy, H. and S. Bohte (2009). "Computing with Spiking Neuron Networks". en. In: *Handbook of Natural Computing*. Ed. by J. K. G. Rozenberg T. Back. Springer Berlin Heidelberg, pp. 335–376. DOI: 10.1007/978-3-540-92910-9_10. URL: http://liris.cnrs.fr/publis/?id=4305.

Pennington, J., R. Socher, and C. D. Manning (2014). "GloVe: Global Vectors for Word Representation". In: *Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543. URL: http://www.aclweb.org/anthology/D14-1162.

Petrovskaya, A. and S. Thrun (2009a). "Efficient Techniques for Dynamic Vehicle Detection". In: *Experimental Robotics*. Ed. by O. Khatib, V. Kumar, and G. J. Pappas. Vol. 54. Berlin, Heidelberg: Springer Berlin Heidelberg. Chap. 10, pp. 79–91. DOI: 10.1007/978-3-642-00196-3_10. URL: http://dx.doi.org/10.1007/978-3-642-00196-3%5C_10.

— (2009b). "Model based vehicle detection and tracking for autonomous urban driving". In: *Autonomous Robots* 26.2, pp. 123–139. ISSN: 1573-7527. DOI: 10.1007/s10514-009-9115-1.

Piatkowska, E., A. N. Belbachir, S. Schraml, and M. Gelautz (2012). "Spatiotemporal multiple persons tracking using Dynamic Vision Sensor". In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 35–40. DOI: 10.1109/CVPRW.2012.6238892.

Plate, T. (1991). "Holographic Reduced Representations: Convolution Algebra for Compositional Distributed Representations". In: *International Joint Conference on Artificial Intelligence*. Morgan Kaufmann, pp. 30–35.

— (1994). "Distributed Representations and Nested Compositional Structure". PhD thesis. University of Toronto.

Plate, T. A. (1994). "Estimating analogical similarity by dot-products of Holographic Reduced Representations". In: *Advances in Neural Information Processing Systems 6*. Ed. by J. D. Cowan, G. Tesauro, and J. Alspector. Morgan-Kaufmann, pp. 1109–1116. URL: http://papers.nips.cc/paper/740-estimating-analogical-similarity-by-dot-products-of-holographic-reduced-representations.pdf.

Ponulak, F. and A. Kasinski (2011). "Introduction to spiking neural networks: Information processing, learning and applications." In: *Acta neurobiologiae experimentalis* 71.4, pp. 409–433. ISSN: 1689-0035. URL: http://view.ncbi.nlm.nih.gov/pubmed/22237491.

Preissl, R., T. M. Wong, P. Datta, M. Flickner, R. Singh, S. K. Esser, W. P. Risk, H. D. Simon, and D. S. Modha (2012). "Compass: A Scalable Simulator for an Architecture for Cognitive Computing". In: *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. SC '12. Salt Lake City, Utah: IEEE Computer Society Press, 54:1–54:11. ISBN: 978-1-4673-0804-5.

Qiao, N., H. Mostafa, F. Corradi, M. Osswald, F. Stefanini, D. Sumislawska, and G. Indiveri (2015). "A Re-configurable On-line Learning Spiking Neuromorphic Processor comprising 256 neurons and 128K synapses". In: *Frontiers in Neuroscience* 9.141. ISSN: 1662-453X. DOI: 10.3389/fnins.2015.00141.

Rasmussen, D. and C. Eliasmith (2011). "A Neural Model of Rule Generation in Inductive Reasoning". In: *Topics in Cognitive Science* 3.1, pp. 140–153. DOI: 10.1111/j.1756-8765.2010.01127.x.

Reverter Valeiras, D., G. Orchard, S. H. Ieng, and R. B. Benosman (2016). "Neuromorphic Event-Based 3D Pose Estimation". In: *Frontiers in Neuroscience* 9.522. ISSN: 1662-453X. DOI: 10.3389/fnins.2015.00522.

Rojas, R. (1996). *Neural Networks: A Systematic Introduction*. New York, NY, USA: Springer-Verlag New York, Inc. ISBN: 3-540-60505-3.

Rosenblatt, F. (1958). "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain". In: *Psychological Review* 65.6, pp. 386–408.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams (1986). "Learning representations by back-propagating errors". In: *Nature* 323.6088, pp. 533–536. DOI: 10.1038/323533a0. URL: https://www.nature.com/articles/323533a0.

Rumelhart, D. E. and J. L. McClelland, eds. (1986). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*. Cambridge, MA, USA: MIT Press. ISBN: 0-262-68053-X.

SAE (2016). *J3016, Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*. Tech. rep. SAE International. DOI: 10.4271/J3016_201609. URL: https://www.sae.org/standards/content/j3016_201609/.

Samsonovich, A. V. (2012). "On a roadmap for the BICA Challenge". In: *Biologically Inspired Cognitive Architectures* 1, pp. 100–107. ISSN: 2212-683X. DOI: 10.1016/j.bica.2012.05.002. URL: http://www.sciencedirect.com/science/article/pii/S2212683X12000126.

Saner, D., O. Wang, S. Heinzle, Y. Pritch, A. Smolic, A. Sorkine-Hornung, and M. Gross (2014). "High-Speed Object Tracking Using an Asynchronous Temporal Contrast Sensor". In: *VMV 2014: Vision, Modeling & Visualization, Darmstadt, Germany, 2014. Proceedings*. European Association for Computer Graphics.

Schemmel, J., D. Brüderle, A. Grübl, M. Hock, K. Meier, and S. Millner (2010). "A wafer-scale neuromorphic hardware system for large-scale neural modeling". In: *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, pp. 1947–1950. DOI: 10.1109/ISCAS.2010.5536970.

Schemmel, J., J. Fieres, and K. Meier (2008). "Wafer-scale integration of analog neural networks". In: *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, pp. 431–438.

Schmidhuber, J. (2015). "Deep Learning in Neural Networks: An Overview". In: *Neural Networks* 61. Published online 2014; based on TR arXiv:1404.7828 [cs.NE], pp. 85–117. DOI: 10.1016/j.neunet.2014.09.003.

Schmitt, S., J. Klaehn, G. Bellec, A. Grübl, M. Guettler, A. Hartel, S. Hartmann, D. H. de Oliveira, K. Husmann, V. Karasenko, M. Kleider, C. Koke, C. Mauch, E. Müller, P. Müller, J. Partzsch, M. A. Petrovici, S. Schiefer, S. Scholze, B. Vogginger, R. A. Legenstein, W. Maass, C. Mayr, J. Schemmel, and K. Meier (2017). "Neuromorphic Hardware In The Loop: Training a Deep Spiking Network on the BrainScaleS Wafer-Scale System". In: *Computing Research Repository (CoRR)* abs/1703.01909. URL: http://arxiv.org/abs/1703.01909.

Schöner, G. (2008). "Dynamical Systems Approaches to Cognition". In: *The Cambridge Handbook of Computational Psychology*. Ed. by R. Sun. Cambridge Handbooks in Psychology. Cambridge University Press, pp. 101–126. DOI: 10.1017/CBO9780511816772.007.

Schraml, S., A. N. Belbachir, N. Milosevic, and P. Schon (2010). "Dynamic stereo vision system for real-time tracking". In: *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*. IEEE, pp. 1409–1412. DOI: 10.1109/ISCAS.2010.5537289.

Schwartz, T. J. (1990). "A Neural Chips Survey". In: *AI Expert* 5.12, pp. 34–38. ISSN: 0888-3785. URL: http://dl.acm.org/citation.cfm?id=95986.95993.

Sermanet, P. and Y. LeCun (2011). "Traffic sign recognition with multi-scale Convolutional Networks". In: *The 2011 International Joint Conference on Neural Networks*. IEEE, pp. 2809–2813. DOI: 10.1109/IJCNN.2011.6033589.

Serrano-Gotarredona, T. and B. Linares-Barranco (2013). "A 128x128 1.5% Contrast Sensitivity 0.9% FPN 3 μs Latency 4 mW Asynchronous Frame-Free Dynamic Vision Sensor Using Transimpedance Preamplifiers". In: *IEEE Journal of Solid-State Circuits* 48.3, pp. 827–838. ISSN: 0018-9200. DOI: 10.1109/JSSC.2012.2230553.

Serrano-Gotarredona, T., B. Linares-Barranco, F. Galluppi, L. Plana, and S. Furber (2015). "ConvNets experiments on SpiNNaker". In: *Circuits and Systems (ISCAS), 2015 IEEE International Symposium on*, pp. 2405–2408. DOI: 10.1109/ISCAS.2015.7169169.

Silver, D., A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis (2016). "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529.7587, pp. 484–489. ISSN: 0028-0836. DOI: 10.1038/nature16961.

Simonyan, K. and A. Zisserman (2014). "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *Computing Research Repository (CoRR)* abs/1409.1556. arXiv: 1409.1556. URL: http://arxiv.org/abs/1409.1556.

Smolensky, P. (1990). "Tensor Product Variable Binding and the Representation of Symbolic Structures in Connectionist Systems". In: *Artificial Intelligence* 46.1-2, pp. 159–216. ISSN: 0004-3702. DOI: 10.1016/0004-3702(90)90007-m.

Srinivasa, N. and J. M. Cruz-Albrecht (2012). "Neuromorphic Adaptive Plastic Scalable Electronics: Analog Learning Systems". In: *IEEE Pulse* 3.1, pp. 51–56. ISSN: 2154-2287. DOI: 10.1109/MPUL.2011.2175639.

Stallkamp, J., M. Schlipsing, J. Salmen, and C. Igel (2012). "Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition". In: *Neural Networks* 32.0, pp. 323–332. ISSN: 0893-6080. DOI: 10.1016/j.neunet.2012.02.016. URL: http://www.sciencedirect.com/science/article/pii/S0893608012000457.

Stanley, G. B. (2013). "Reading and writing the neural code". In: *Nature Neuroscience* 16.3, pp. 259–263. ISSN: 1097-6256. DOI: 10.1038/nn.3330.

Stefanini, F., S. Sheik, E. Neftci, and G. Indiveri (2014). "PyNCS: a microkernel for high-level definition and configuration of neuromorphic electronic systems". In: *Frontiers in Neuroinformatics* 8.73. DOI: 10.3389/fninf.2014.00073. URL: http://ncs.ethz.ch/pubs/pdf/Stefanini_etal14.pdf.

Stewart, T., T. Bekolay, and C. Eliasmith (2012). "Learning to Select Actions with Spiking Neurons in the Basal Ganglia". In: *Frontiers in Neuroscience* 6, p. 2. ISSN: 1662-453X. DOI: 10.3389/fnins.2012.00002.

Stewart, T. C., X. Choo, and C. Eliasmith (2010). "Dynamic Behaviour of a Spiking Model of Action Selection in the Basal Ganglia". In: *10th International Conference on Cognitive Modeling*.

Stewart, T. C., A. Kleinhans, A. Mundy, and J. Conradt (2016). "Serendipitous Offline Learning in a Neuromorphic Robot". In: *Frontiers in Neurorobotics* 10.1. ISSN: 1662-5218. DOI: 10.3389/fnbot.2016.00001.

Stewart, T. C., Y. Tang, and C. Eliasmith (2011). "A Biologically Realistic Cleanup Memory: Autoassociation in Spiking Neurons". In: *Cognitive Systems Research* 12, pp. 84–92. DOI: 10.1016/j.cogsys.2010.06.006.

Stewart, T. C., B. Tripp, and C. Eliasmith (2009). "Python scripting in the nengo simulator". In: *Frontiers in neuroinformatics* 3, p. 7. ISSN: 1662-5196. DOI: 10.3389/neuro.11.007.2009.

Steyer, S., G. Tanzmeister, and D. Wollherr (2018). "Grid-Based Environment Estimation Using Evidential Mapping and Particle Tracking". In: *IEEE Transactions on Intelligent Vehicles* 3.3, pp. 384–396. ISSN: 2379-8904. DOI: 10.1109/TIV.2018.2843130.

Stromatias, E., D. Neil, F. Galluppi, M. Pfeiffer, S.-C. Liu, and S. Furber (2015a). "Scalable energy-efficient, low-latency implementations of trained spiking Deep Belief Networks on SpiNNaker". In: *2015 International Joint Conference on Neural Networks (IJCNN)*. IEEE, pp. 1–8. DOI: 10.1109/ijcnn.2015.7280625.

Stromatias, E., D. Neil, M. Pfeiffer, F. Galluppi, S. B. Furber, and S.-C. Liu (2015b). "Robustness of spiking Deep Belief Networks to noise and reduced bit precision of neuro-inspired hardware platforms". In: *Frontiers in Neuroscience* 9.222. ISSN: 1662-453X. DOI: 10.3389/fnins.2015.00222.

Taieb, S. B. and R. Hyndman (2014). "Boosting multi-step autoregressive forecasts". In: *Proceedings of the 31st International Conference on Machine Learning*. Ed. by E. P. Xing and T. Jebara. Vol. 32. Proceedings of Machine Learning Research 1. Bejing, China: PMLR, pp. 109–117. URL: http://proceedings.mlr.press/v32/taieb14.html.

Tan, C., S. Lallee, and G. Orchard (2015). "Benchmarking Neuromorphic Vision: Lessons Learnt from Computer Vision". In: *Frontiers in Neuroscience* 9.374. ISSN: 1662-453X. DOI: 10.3389/fnins.2015.00374.

Tanzmeister, G., J. Thomas, D. Wollherr, and M. Buss (2014). "Grid-based mapping and tracking in dynamic environments using a uniform evidential environment representation". In: *2014 IEEE International Conference on Robotics and Automation, ICRA 2014, Hong Kong, China, May 31 - June 7, 2014*, pp. 6090–6095. DOI: 10.1109/ICRA.2014.6907756.

Thagard, P. (2012). "Cognitive Architectures". In: *The Cambridge Handbook of Cognitive Science*. Ed. by K. Frankish and W. Ramsey. Cambridge University Press, pp. 50–70.

Thorpe, C., M. H. Hebert, T. Kanade, and S. A. Shafer (1988). "Vision and navigation for the Carnegie-Mellon Navlab". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 10.3, pp. 362–373. ISSN: 0162-8828. DOI: 10.1109/34.3900.

Thrun, S., W. Burgard, and D. Fox (2005). *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents series)*. Intelligent robotics and autonomous agents. The MIT Press. ISBN: 0262201623.

Thrun, S., M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L.-E. Jendrossek, C. Koelen, C. Markey, C. Rummel, J. van Niekerk, E. Jensen, P. Alessandrini, G. Bradski, B. Davies, S. Ettinger, A. Kaehler, A. Nefian, and P. Mahoney (2006). *Stanley: The Robot That Won the DARPA Grand Challenge: Research Articles*. Ed. by K. Iagnemma and M. Buehler. Chichester, UK. DOI: 10.1002/rob.v23:9.

Tian, X. and Y. Bar-Shalom (2010). "On algorithms for asynchronous Track-to-Track Fusion". In: *2010 13th International Conference on Information Fusion*. IEEE, pp. 1–8. DOI: 10.1109/ICIF.2010.5711956.

Treisman, A. (1999). "Solutions to the Binding Problem". In: *Neuron* 24.1, pp. 105–125. ISSN: 0896-6273. DOI: 10.1016/s0896-6273(00)80826-0.

Urmson, C., J. Anhalt, D. Bagnell, C. Baker, R. Bittner, M. N. Clark, J. Dolan, D. Duggins, T. Galatali, C. Geyer, M. Gittleman, S. Harbaugh, M. Hebert, T. M. Howard, S. Kolski, A. Kelly, M. Likhachev, M. McNaughton, N. Miller, K. Peterson, B. Pilnick, R. Rajkumar, P. Rybski, B. Salesky, Y.-W. Seo, S. Singh, J. Snider, A. Stentz, W. " Whittaker, Z. Wolkowicki, J. Ziglar, H. Bae, T. Brown, D. Demitrish, B. Litkouhi, J. Nickolaou, V. Sadekar, W. Zhang, J. Struble, M. Taylor, M. Darms, and D. Ferguson (2008). "Autonomous driving in urban environments: Boss and the Urban Challenge". In: *Journal of Field Robotics Special Issue on the 2007 DARPA Urban Challenge, Part I* 25.8. Ed. by S. S. Martin Buehler Karl Lagnemma, pp. 425–466. DOI: 10.1002/rob.20255.

Vapnik, V. N. (1995). *The Nature of Statistical Learning Theory*. New York, NY, USA: Springer-Verlag New York, Inc. ISBN: 0-387-94559-8.

Voelker, A. R., J. Gosmann, and T. C. Stewart (2017). *Efficiently sampling vectors and coordinates from the n-sphere and n-ball*. en. Tech. rep. Waterloo, ON: Centre for Theoretical Neuroscience. DOI: 10.13140/rg.2.2.15829.01767/1. URL: https://www.researchgate.net/publication/312056739_Efficiently_sampling_vectors_and_coordinates_from_the_n-sphere_and_n-ball.

Wahle, M., D. Widdows, J. R. Herskovic, E. V. Bernstam, and T. Cohen (2012). "Deterministic Binary Vectors for Efficient Automated Indexing of MEDLINE/PubMed Abstracts". In: *AMIA Annual Symposium Proceedings* 2012.PMC3540485, pp. 940–949. ISSN: 1942-597X. URL: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3540485/.

Walter, F., F. Röhrbein, and A. Knoll (2015). "Neuromorphic implementations of neurobiological learning algorithms for spiking neural networks". In: *Neural Networks* 72. Neurobiologically Inspired

Robotics: Enhanced Autonomy through Neuromorphic Cognition, pp. 152–167. ISSN: 0893-6080. DOI: 10.1016/j.neunet.2015.07.004.

Wang, Y. and D. Liu (2003). "On information and knowledge representation in the brain". In: *The Second IEEE International Conference on Cognitive Informatics, 2003. Proceedings.* IEEE Comput. Soc, pp. 26–31. DOI: 10.1109/coginf.2003.1225947.

Waymo LLC (2018). *Waymo webpage*. URL: https://waymo.com/ (visited on 2018-02-08).

Webber, F. E. D. S. (2016). *Semantic Folding*. URL: http://www.cortical.io/static/downloads/semantic-folding-theory-white-paper.pdf (visited on 2019-04-03).

Weikersdorfer, D., D. B. Adrian, D. Cremers, and J. Conradt (2014). "Event-based 3D SLAM with a depth-augmented Dynamic Vision Sensor". In: *2014 IEEE International Conference on Robotics and Automation, ICRA 2014, Hong Kong, China, May 31 - June 7, 2014*, pp. 359–364. DOI: 10.1109/ICRA.2014.6906882.

Weikersdorfer, D. and J. Conradt (2012). "Event-based particle filtering for robot self-localization". In: *2012 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. IEEE, pp. 866–870. DOI: 10.1109/robio.2012.6491077.

Weikersdorfer, D., R. Hoffmann, and J. Conradt (2013). "Simultaneous Localization and Mapping for Event-Based Vision Systems". In: *Computer Vision Systems - 9th International Conference, ICVS 2013, St. Petersburg, Russia, July 16-18, 2013. Proceedings*, pp. 133–142. DOI: 10.1007/978-3-642-39402-7_14.

Werbos, P. J. (1974). "Beyond regression: new tools for prediction and analysis in the behavioral sciences". PhD thesis. Harvard University.

Wymann, B., E. Espié, C. Guionneau, C. Dimitrakakis, R. Coulom, and A. Sumner (2014). *TORCS, The Open Racing Car Simulator*. URL: http://www.torcs.org.

Yamazaki, S., C. Miyajima, E. Yurtsever, K. Takeda, M. Mori, K. Hitomi, and M. Egawa (2016). "Integrating driving behavior and traffic context through signal symbolization". In: *2016 IEEE Intelligent Vehicles Symposium (IV)*, pp. 642–647. DOI: 10.1109/IVS.2016.7535455.