Fakultät für Informatik
Technische Universität München

Using Deep Neural Networks for Scene Understanding and Behaviour Prediction in Autonomous Driving

Oliver Scheel

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:
Prof. Dr. Nils Thürey

Prüfende der Dissertation:
1. Priv.-Doz. Dr. Federico Tombari
2. Assistant Prof. Alexandre Alahi, Ph.D.
3. Prof. Dr. Matthias Nießner

Die Dissertation wurde am 11.05.2020 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 12.09.2020 angenommen.

Abstract

Autonomous driving inherently is a sequential problem: Vehicles encounter complex, dynamically changing traffic scenes, and have to anticipate future behaviour of other agents while planning safe trajectories for themselves. To solve this ambitious task, data-driven end-to-end methods are experimented with, as well as more traditional, stack-based approaches. In these, the problem is handled in a layered fashion, and separate, modular components solve specific subtasks. Typical layers are perception, fusion, prediction, planning and control. This thesis addresses the fields prediction and planning, exploiting their sequential nature.

In recent years, recurrent neural networks have become widely used in a variety of fields, achieving promising results, and often outperforming other methods. Exemplary applications include natural language processing, machine translation and trajectory prediction. Here, we apply such techniques for solving pending challenges in the area of autonomous driving.

We begin by analyzing the problem of predicting discrete driving manoeuvres. For this, we develop a recurrent model which processes full scenes as inputs and yields accurate predictions. In particular, compared to previous works, we employ additional features, such as dynamic and static environment information, in order to improve performance. As ambiguity is present in many scenes, we continue by exploring ways of modelling such ambiguity and uncertainty. Therefore, we propose a framework for extending existing recurrent models to predict multiple hypotheses.

We then turn our focus towards planning problems: To circumvent known issues of data-driven methods for solving such tasks, we introduce a novel supporting layer. The aim of this layer is to analyze complex traffic situations and generate recommendations for other planning algorithms, supporting them in decision making w.r.t. feasible manoeuvres. We further examine interesting connections between the two related tasks of predicting and planning manoeuvres.

Finally, we address the issue of shifting domains, since autonomous vehicles need to be able to navigate in various environmental conditions and all over the world. Therefore, we propose a framework to assist deployment of our previously learned prediction models in new domains.

Zusammenfassung

Autonomes Fahren ist von Natur aus ein sequentielles Problem: Fahrzeuge durchlaufen komplexe, dynamische Fahrsituationen und müssen zukünftiges Verhalten anderer Agenten antizipieren, während sie sichere Trajektorien für sich selber planen. Um dieses anspruchsvolle Problem zu lösen, wird mit datengetriebenen Ende-zu-Ende Methoden sowie traditionelleren, Stack-basierten Ansätzen experimentiert. In diesen wird das Problem auf einen schichten-basierten Ansatz heruntergebrochen und eigenständige, modulare Komponenten lösen spezifische Unteraufgaben. Typische Schichten sind Perzeption, Sensorfusion, Prädiktion, Planung und Kontrolle. Diese Dissertation behandelt die Felder Prädiktion und Planung unter Ausnutzung ihrer sequentiellen Natur.

In den letzten Jahren wurden rekurrente neuronale Netzwerke in einer Vielfalt von Anwendungsgebieten eingesetzt und haben in diesen vielversprechende Ergebnisse geliefert. Beispielanwendungen umfassen Computerlinguistik, maschinelle Übersetzung und Trajektorienprädiktion. Hier wenden wir solche Netzwerke auf bestehende Herausforderungen im Bereich des autonomen Fahrens an.

Wir beginnen mit der Erkundung des Problems der Prädiktion von diskreten Manövern. Dafür entwickeln wir ein rekurrentes neuronales Netzwerk, welches komplette Fahrszenarien als Eingabe verarbeitet und genaue Prädiktionen liefert. Im Vergleich zu vorherigen Arbeiten benutzen wir zusätzliche Eingabefeatures, wie dynamische und statische Umgebungsinformationen, und verbessern damit die Prädiktionsleistung. Da Mehrdeutigkeit in vielen solcher Szenarien vorhanden ist, untersuchen wir im Weiteren Möglichkeiten, solche Mehrdeutigkeiten und Unsicherheiten zu modellieren. Dafür führen wir ein Framework ein, mit welchem existierende rekurrente Modelle für die Prädiktion von multiplen Hypothesen erweitert werden.

Danach konzentrieren wir uns auf Planungsprobleme: Um bekannte Nachteile datengetriebener Methoden für solche Aufgaben zu umgehen, führen wir eine neue, unterstützende Schicht ein. Ziel dieser Schicht ist die Analyse von komplexen Verkehrssituationen, die Erzeugung von Empfehlungen für andere Planungsalgorithmen und die Unterstützung dieser in der Entscheidungsfindung von geeigneten Manövern.

Schließlich behandeln wir das Thema Domänentransfer, da autonome Fahrzeuge in verschiedensten Umgebungen einsetzbar sein müssen. Dafür schlagen wir ein Framework zur Übertragung der vorher eingeführten Prädiktionsmodelle in neue Domänen vor.

Acknowledgements

This thesis resulted from a cooperation of BMW Group and Technical University of Munich. I would like to thank both organizations for this possibility: TU Munich for the honour of allowing me to enroll in their PhD programme, and BMW Group for letting me join their autonomous driving department. It has been an incredible time, and fulfilled everything I dared dreaming for, and more.

This would not have been possible without my supervisor PD Dr. Ing. Habil. Federico Tombari, who taught me everything I know today, and to whom I will forever be grateful. The same holds for my supervisor from BMW's side, Dr. Loren Schwarz, who was always there for me with great support and mentorship. I would also like to thank Prof. Dr. Nassir Navab for his incredibly warm welcome into his research group and allowing access to the chair's facilities.

Throughout this PhD, I have had the immense pleasure of working with many great scientists and colleagues: I would like to thank Prof. Dr. Luigi Di Stefano for his great feedback and lengthy discussions about including ambiguity in computer vision and machine learning applications. The same holds for Alessandro Berlati, whose great work started this collaboration. I was blessed to have the opportunity of learning from many incredible people, who helped shape this PhD into what it is now, and always happily offered time for questions and discussions of any kind. In particular, I wish to thank Naveen Shankar Nagaraja, Mira Slavcheva, Christian Rupprecht, Iro Laina and Nikolas Brasch.

Further, I would like to thank my amazing colleagues from BMW Group and the Chair for Computer Aided Medical Procedures at the Technical University of Munich. Next to all the scientific development and knowledge exchange, people are what makes a job special. Unfortunately, the list of great colleagues is way too long to name them all, but special thanks to Alexander and Lukas Frickenstein, Jakob Mayr, Dmitrij Schitz, Luca Puccetti, Thomas Barowski, Egon Ye, Chen Ee Heng, Artem Savkin, Markus Herb, Johanna Wald, Fabian Manhardt, Helisa Dhamo, Maria Tirindelli, Mai Bui and numerous others.

Although I look back with joy, times have not always been easy, and I would like to thank my friends for bearing with me and supporting me and this thesis, particularly Anja Leckel and Jordi Chervitz.

Last, but definitely not least, I want to thank my parents, Brigitte and Werner. They were always there for me, and helped me through this with their continuous support and encouragement.

Contents

List of Figures

List of Tables

Abbreviations

ACC	**A**utomatic **C**ruise **C**ontrol
ADE	**A**verage **D**isplacement **E**rror
AGI	**A**rtificial **G**eneral **I**ntelligence
BPTT	**B**ackpropagation **T**hrough **T**ime
CNN	**C**onvolutional **N**eural **N**etworks
CVAE	**C**onditional **V**ariational **A**utoencoder
DBN	**D**ynamic **B**ayesian **N**network
ECU	**E**lectronic **C**ontrol **U**nit
ELBO	**E**vidence **L**ower **B**ound
FDE	**F**inal **D**isplacement **E**rror
GAN	**G**enerative **A**dversarial **N**etwork
GPU	**G**raphics **P**rocessing **U**nit
GRU	**G**ated **R**ecurrent **U**nit
HD	**H**igh **D**efinition
IDM	**I**ntelligent **D**river **M**odel
IL	**I**mitation **L**earning
LSTM	**L**ong **S**hort-**term** **M**emory
MCL	**M**ultiple **C**hoice **L**earning
MDN	**M**ixture **D**ensity **N**etwork
MHP	**M**ultiple **H**ypothesis **P**rediction
MLP	**M**ulti-layer **P**erceptron
NLP	**N**atural **L**anguage **P**rocessing
PTB	**P**enn **T**ree **B**ank Dataset
RF	**R**andom **F**orests
RL	**R**einforcement **L**earning
RNN	**R**ecurrent **N**eural **N**etwork
SDD	**S**tanford **D**rone **D**ataset
SfM	**S**tructure-**f**rom-**M**otion
STN	**S**patial **T**ransformer **N**etwork
SVM	**S**upport **V**ector **M**achine

TBPTT	**T**runcated **B**ackpropagation **T**hrough **T**ime
VAE	**V**ariational **A**uto**e**ncoder
VI	**V**ariational **B**ayesian **I**nference

Part I

Introduction and Fundamentals

1 Introduction

Automobiles have drastically changed the way billions of people around the globe live: Their invention, and successive cost reduction following innovations in industrial production, for the first time in history provided common people with an easy way of individual mobility. Nowadays, for us it is normal to visit distant places and regularly cover medium distances, either for work or leisure time.

Today, we are close to another revolution: Autonomous vehicles are on the verge of completion, and we might see fully autonomous vehicles on the road within this decade. This will drastically increase safety for all traffic participants, reduce traffic congestion, and again completely reshape our way of transportation: Ride and car sharing is made easier, as owners could simply "send away" their vehicles to autonomously transport other passengers. The need for parking in already congested urban environments would decrease, as vehicles could autonomously find suitable parking possibilities outside these areas, freeing up space for pedestrians and green spaces.

Already in the 80s people were experimenting in the field of autonomous driving, equipping series cars with cameras and (for our terms) rather simple perception and control algorithms, and testing these on highways [7]. But it is only now, that fully autonomous driving is deemed feasible and will likely be available in a not too-distant future. Reasons for this are advances in computer science, software and hardware, especially the ability to effectively train deep neural networks on Graphics Processing Units (GPUs). Due to this, nearly all large automobile manufacturers and many research institutes as well as a multitude of startups currently are heavily focusing on making autonomous driving available for the customer. Although launch schedules have been moved forward in time, it is expected to see fully autonomous vehicles within this decade.

Many experts believe artificial intelligence to be one of the key enablers towards this goal. Techniques from machine learning, especially deep neural networks, have already revolutionized our way of solving problems in many fields and are considered state-of-the-art in these: In the fields image classification, segmentation and

bounding box detection *Convolutional Neural Networks* (CNNs) are best-performing models and have replaced their predecessors. Whereas earlier image classification pipelines consisted of multiple steps, e.g. extracting hand-crafted features and then training a machine learning classifier on these, CNNs solve the whole problem in one step. It is a similar case for sequential problems, such as *Natural Language Processing* (NLP): A conventional pipeline consisting of extracting expert features and modelling language properties is replaced by deep models, such as *Recurrent Neural Networks* (RNNs).

Due to their capability of being able to memorize information, RNNs are especially suited for the understanding of time-series data, for which they excel. This makes them noteworthy candidates for an application in the field of autonomous driving, as here we naturally deal with data varying over time, such as complex traffic scenes and trajectories. However, prior research has typically addressed these using "classical" methods. Therefore, goal of this thesis is exploring the possibility of employing RNNs for the use in different areas of autonomous driving, and thus helping bringing the dream of driverless vehicles one step closer.

1.1 Motivation

For solving the problem of fully autonomous driving, numerous solution candidates exist. From an architectural point of view, these can be classified into *end-to-end*, *mid-to-mid* or more traditional, *stack-based* approaches. End-to-end learning describes the process of mapping raw sensor inputs directly to, e.g., a steering angle. So far more common in the field of robotics as of today is a stack-based architecture. This is a layered architecture, in which each layer serves a specific purpose. Advantages are a better modularity and decomposability of the system, specific functions and layers can be considered and verified separately. For autonomous driving, typically the five building blocks perception, fusion, prediction, planning and control are used (see Figure 1.1).

An autonomous vehicle needs to be able to perceive its environment. For this, various sensors, such as cameras, Radar and LiDAR sensors, are employed. *Perception* describes the handling of these raw sensor inputs. The many different sensors are typically redundant and overlapping, regarding e.g. their field of view. In particular, in addition to employing different sensor types as mentioned before, vehicles are usually equipped with several such sensors, e.g. exhibiting multiple cameras. The layer *Fusion* describes the fusion of these separate sensory inputs into one global understanding of the environment. Since driving scenes are virtually always dynamic, it is important to model them as such, i.e. incorporate at least some basic assumptions about how they unfold in the future. *Prediction* is concerned with these aspects. *Planning* constitutes of planning a path for the vehicle, given all this information, and is hierarchically structured itself: One begins with global / mission planning (finding routes in maps) and moves down in abstraction levels towards actual trajectories and driving paths. The field *control* then translates these into commands

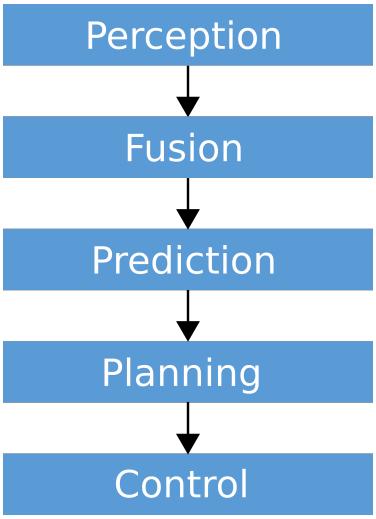


Figure 1.1: Depiction of a typical stack-based architecture used for autonomous vehicles: A perception layer processes inputs of different sensors, such as cameras, Radar and LiDAR sensors. A subsequent fusion layer fuses these information to generate an understanding of traffic scenes. Next, a prediction component turns this static understanding into a dynamic one, predicting how the scene will unfold over time. A planning layer is responsible for planning how the ego vehicle moves through the scene, e.g. calculating trajectories. A subsequent control layers then converts these requirements into actuator commands.

for control units and actuators. In this thesis we concentrate on the fields planning and prediction.

Possible prediction targets are continuous trajectories (e.g. from other traffic participants, such as other vehicles or pedestrians) or discrete manoeuvres, such as lane change events. Research in trajectory prediction is very active, especially regarding pedestrians on public datasets. Knowing such future behaviour is important, as it helps the planning task and reduces risk. Predicting discrete manoeuvres, such as lane changes, is another important task, and e.g. needed for driving assistance functions already existing in current series cars: When steering the vehicle through lane keeping assistant and Automatic Cruise Control (ACC), we would like to know in advance, when other vehicles will merge into our lane, so as to enable smooth and early reactions. Most works in this field employ more traditional, classical methods to model this task, such as Random Forests (RF). We see the potential and advantages of describing this as a sequential problem and applying deep neural networks on this.

Another important aspect of general machine learning applications is the prediction of multiple hypotheses. By default, most models only output a single hypothesis, and underestimate the variance of possible solutions. This is especially true for sequential problems, as here often multiple futures are equally likely, and for the problem of autonomous driving in particular: While driving, agents are faced

with multiple possibilities, and have to account for the numerous possible choices by other agents. Helping improve understanding and prediction performance in these scenarios constitutes another topic of this thesis.

Planning can either be addressed jointly in an end-to-end system, or as a separate tasks. Possible data-driven methods are *Imitation Learning* (IL) and *Reinforcement Learning* (RL). IL tries to mimic expert behaviour to solve a task, while in RL an agent explores vast action-spaces and learns by reward and punishment. Although yielding promising results in many other fields, applications of such techniques in this field are still rare. One reason is the difficult verification process for such black-box models, as well as known limitations of either approach: RL requires a realistic driving simulator, but still exhibits a performance drop when changing to real-world data. IL is know to suffer from compounding errors over time and overfits characteristic expert behaviours. We believe, a combination of both paradigms will be necessary for solving this problem, and further advocate development of new models, which are able to understand complex and dynamic scenes.

Transfer learning describes the process of applying learned knowledge to another domain or even task. In automotive, this is especially important, as different domains arise on many levels: Multimodal sensory inputs are used, whose hardware and software change over time, autonomous vehicles need to navigate through all weather conditions and across diverse countries with country-specific driving styles and rules. Most works in this field focus on image-data, e.g. learn generative models for hallucinating samples from different domains, and pay less attention to intermediate data representations as well as time-series data.

1.2 Objectives

Goal of this thesis is to build up knowledge about applying RNNs for different problems in the field of autonomous driving, and thus help advance this topic. For all of these, we use intermediate feature representations, i.e. no raw sensor data, but instead fused high-level features such as object lists.

We begin with the field of prediction: We would like to know how traffic scenes continue to unfold over time. For this, we propose and analyze different prediction methods to predict future discrete manoeuvres or continuous trajectories of other traffic participants. In the next part, we examine ways of generating multimodal outputs. In particular, we propose a general way of extending recurrent methods to account for ambiguity, and show its applicability for automotive use cases. Moving on to planning, we develop a deep network capable of understanding dynamic traffic scenes. This supporting layer can be queried by other ones in the software stack, such as planning, answering queries whether certain manoeuvres currently are possible and feasible for the ego vehicle, or not. Eventually, due to the numerous scene variations possible when tackling the problem of autonomous driving, we set out to make our previously introduced models robust to domain changes and to develop

methods of transferring knowledge between domains. Summarized, our objectives are:

- early and accurate prediction models;
- extension to multimodal prediction models;
- scene understanding of complex scenes;
- transfer learning for prediction models.

1.3 Contributions

To fulfill the objectives outlined above, we develop several algorithms:

- **Predicting Lane Change Manoeuvres**: We employ RNNs to predict future lane change manoeuvres of other traffic participants. Predictions should be as early as possible, while simultaneously resulting in few false predictions, as these have a strong negative influence on felt driving comfort. In addition we propose usage of an attention mechanism, to model selective focus and allow models to focus on certain important aspects of a scene. This results in better and more interpretable models.
- **Predicting Multiple Futures**: We extend the Multiple Hypothesis Prediction (MHP) framework [8] to recurrent models and show how to incorporate it into different state-of-the-art recurrent architectures, such as encoder-decoder models. We showcase its usefulness for predicting multiple hypotheses with our lane change prediction model and trajectory forecasting models.
- **Situation Assessment for Lane Changes**: We train a bidirectional RNN for scene understanding w.r.t. lane changes. Output in each step is a binary decision indicating whether the scene is suited for a lane change or not. This can be queried by any planning algorithm, whether it is a classical rule-based planner or consists of IL or RL techniques. Due to this, we are invariant to model-specific up- and downsides of learned planning methods, but still make use of the power of deep learning. Incorporating assumptions about rational drivers and physical constraints allows us to explicitly model future scene developments and thus the usage of a bidirectional RNN.
- **Transfer Learning**: We develop a framework for transfer learning, which is explicitly tailored for automotive use cases, e.g. due to its calculation of an explicit transformation between domains. This transformation can be initialized based on domain knowledge, and be used in further safety verification processes.
- **Evaluation Metrics**: We propose several metrics for evaluating lane change prediction models, also applicable to general manoeuvre prediction. Core idea

is their interpretation of lane changes as event-wise continuous sequences, and they allow better quantification of what drivers inside vehicles actually experience, compared to traditional metrics.

1.4 Outline

This section provides a brief overview over the following chapters. Most of the proposed methods are published or under submission for a major conference or journal. We thus indicate the corresponding publications, and also refer the interested reader to the conference websites to consult further supplementary material.

Chapter 2 In this chapter we briefly introduce fundamentals of techniques applied in this thesis. We begin by introducing general machine learning concepts, and then move on to more specific topics, such as RNNs, considering ambiguity and transfer learning. Eventually, we introduce the topic of autonomous driving, defining relevant subproblems and methods for solving them.

Chapter 3 In this chapter we present our recurrent approach for predicting lane change manoeuvres. After defining the problem and introducing used scene representation, which we will reuse throughout the thesis, we showcase the proposed methods and describe findings of numerous experiments. The related publications are:

- O. Scheel, N. S. Nagaraja, L. Schwarz, N. Navab, F. Tombari, "Attention-based Lane Change Prediction", Int. Conf. on Robotics and Automation (ICRA), 2019 [4]

- O. Scheel, N. S. Nagaraja, L. Schwarz, N. Navab, F. Tombari, "Recurrent Models for Planning and Predicting Lane Changes", submitted to Transactions on Robotics (T-RO), 2020 [2]

Chapter 4 Here, we propose a general framework for predicting multiple hypotheses with recurrent models. We examine different recurrent architectures, such as encoder-decoder models, and diverse problems. In particular, we modify the manoeuvre prediction approach from Chapter 3, predict multimodal trajectories and generate text. The corresponding publication is:

- A. Berlati, O. Scheel, L. Di Stefano, F. Tombari, "Ambiguity in Sequential Data: Predicting Uncertain Futures with Recurrent Models", Robotics and Automation Letters, also accepted for publication at Int. Conf. on Robotics and Automation (ICRA), 2020 [5]

Chapter 5 In Chapter 5 we focus on the area of planning. We introduce a supporting layer for other planning algorithms, analyzing situations w.r.t. their suitability for lane changes. The related publication is:

- O. Scheel, L. Schwarz, N. Navab, F. Tombari, "Situation Assessment for Planning Lane Changes: Combining Recurrent Models and Prediction", Int. Conf. on Robotics and Automation (ICRA), 2018 [3]

Chapter 6 Eventually, in this chapter we examine the problem of changing domains and propose a framework for addressing this issue. In particular, we consider the task of predicting lane changes from Chapter 3, and showcase the transfer of a learned model to a new domain. The resulting publication is:

- O. Scheel, L. Schwarz, N. Navab, F. Tombari, "Explicit Domain Adaptation with Loosely Coupled Samples", submitted to Robotics and Automation Letters and Int. Conf. on Intelligent Robots and Systems (IROS), 2020 [6]

2 Fundamentals

This chapter provides an overview of concepts from machine learning and autonomous driving that will be used throughout the thesis. We begin by introducing fundamental machine learning techniques, such as the most commonly used neural network architectures. We then focus on RNNs, as these are employed for a majority of this thesis. Next topics are dealing with uncertainty and ambiguity and the field of transfer learning. Finally, we move to the area of autonomous driving. We introduce fundamentals, such as used sensors, and then give a brief overview over fields particularly relevant for us, namely prediction and planning.

2.1 Machine Learning Basics

General goal of machine learning is to design machines and algorithms, which learn to perform a task from experience. Often, this can be expressed as a function:

$$\mathbf{y} = f(\mathbf{x}, \mathbf{w}) \tag{2.1}$$

In this, $\mathbf{x}$ denotes input data given to the algorithm or machine, and $\mathbf{y}$ an expected output. Consider the problem of classifying images: Inputs in this case are images or image patches, and we want to obtain a classification of these, denoting what kind of object is depicted in the image (e.g., is it a cat or a dog). As the name suggests, essence of machine learning methods are models capable of learning. Assume we have a model designed for image classification. Initially, without any prior knowledge output of this model will be random, and useless to us. We can, however, train this model by showing it corresponding pairs of inputs $\mathbf{x}$ and associated labels $\mathbf{y}$, and update its internal weights or parameters $\mathbf{w}$ in such a way that its performance for this task increases.

Types of Learning Such training can be conducted in various ways: Most prominent methods are *supervised*, *unsupervised* and *reinforcement learning*. The previously introduced example of image classification is a typical application of supervised

learning. The task consists of corresponding pairs of inputs and known outputs. For each input, a label (also called ground truth), is given, which serves as training target for the learning algorithm. Models are trained on a dataset, with aims of predicting the correct output for as many samples as possible. If the desired output is discrete, we speak of *classification* problems (such as image classification). In case of continuous outputs the problem is called *regression*, and models try to predict continuous outputs as close to the ground truth as possible. One example for this is the prediction of stock prices based on historical data.

Aim of unsupervised learning instead is to learn patterns in the given data, without any explicitly given labels. One example application is clustering, in which clusters of corresponding data samples are to be found (imagine for example the detection of communities in social networks). Also hybrid methods are possible, such as (semi) supervised learning, in which partial labels for data subsets are available.

Reinforcement learning generates a training impulse via reward and punishment: Agents are able to freely explore certain environments and interact with these. No explicit labelling of good and bad actions is given; instead, certain states or actions are associated with a reward or punishment. Goal of the agent is to maximize its reward, this way it implicitly learns to exhibit desired behaviour. Consider the problem of learning to drive autonomously: Rewards could be allotted for reaching certain speeds and following desired driving lanes, whereas crashes and similar would induce a high punishment.

A distinction between discriminative and generative models exists, mostly in the field of supervised learning: Discriminative methods directly learn a decision boundary based on posterior class probabilities. Via this they assign most probable classes to every input sample. One well-performing example are *Support Vector Machines* (SVMs). Generative models, on the other hand, explicitly model and employ joint probability distributions. Using these, still class-membership can be inferred, but additionally it is also possible to sample new data points. Especially for image domains fascinating results are achieved using e.g. *Generative Adversarial Networks* (GANs), such as transforming objects or full scenes.

Deep Learning The term deep learning describes the usage of so-called deep models, i.e. models consisting of many successively stacked layers. Often, these are neural networks. Despite their perception as fundamentally new domain, artificial neural networks have been around for decades: When Rosenblatt invented the Perceptron in 1957 [9], it sparked huge media interest, with people assuming algorithms for general intelligence would soon be available. However, when Minsky and Papert showed, that (single-layer) perceptrons cannot model arbitrary functions [10], this drastically decreased interest in this field. It should be noted that results only hold for single-layer perceptrons though, and multi-layer perceptrons with non-linear activation functions are universal function approximators. This realization and others, like the introduction of the backpropagation algorithm, led to a short resurgence of neural networks in the 80s. As they, still, proved to be hard to train and lacked

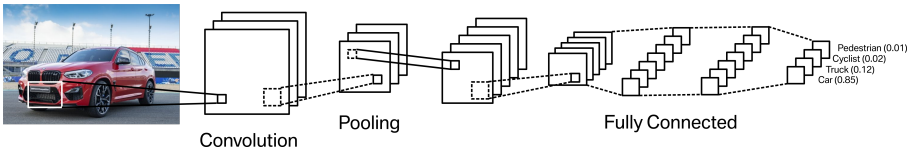


Figure 2.1: Sample usage of a CNN for classifying images. A convolution layer consists of moving a kernel over the image and recording responses. This is often followed by a pooling layer, which aggregates these responses in a certain way (e.g. max-pooling is common). Eventually, fully connected layers generate probabilities of possible classes and thus predictions.

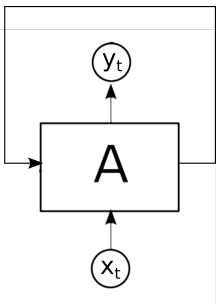


Figure 2.2: Depiction of a simple RNN: At timestep t, the network processes an input x_t, and generates an output y_t. Stored information, e.g. a hidden state, is fed back to the network for the next timestep, such that reasoning about temporal context is possible.

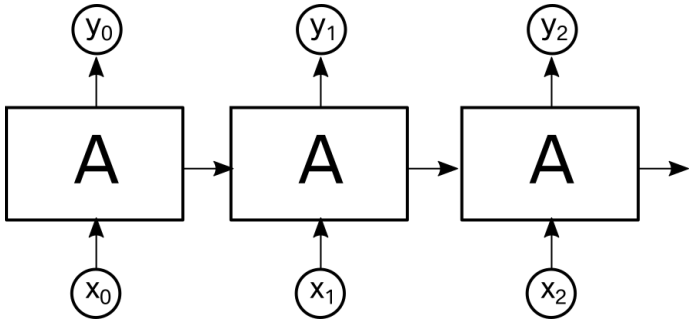


Figure 2.3: Unrolled visualization of the RNN from Figure 2.2.

intuition as to what inner layers of these networks actually learn, it was not before the first decade of the year 2000 that neural networks became truly feasible for practical problems. Reasons for this were vastly improved computing capabilities, among others using GPUs, which allowed better training of neural networks and further the processing and generation of large datasets. A major breakthrough was achieved on the ImageNet challenge in 2012, when a CNN outperformed traditional computer vision approaches [11].

Nowadays, deep neural networks are state-of-the-art for many problems, such as image classification and machine translation. Common network types are:

- Multi-layer Perceptron (MLP): An MLP is the simplest feed-forward neural network, and consists of input neurons for handling input, and output neurons for outputting results. In between these, a number of hidden layers can be implemented. For each neuron, a (possibly) non-linear activation function such as `tanh` and `sigmoid` can be used. Adding non-linearities in a network is important, as otherwise outputs are just linear combinations of the inputs. Despite their simplicity, MLPs are universal function approximators, and thus can approximate any function arbitrary well.

- Convolutional Neural Network (CNN) [12]: CNNs are exceptional in processing spatial, structured data such as images and are top-performing methods for many tasks, such as image classification, object detection and semantic segmentation. CNNs drastically reduce the amount of free parameters by sharing parameters (so-called *kernels*) across different locations. These kernels are stacked in depth to form *filters*, additionally pooling layers guarantee spatial invariance. See Figure 2.1 for an example.

- (Variational) Autoencoder (VAE) [13]: Autoencoders employ two inverse networks, first projecting inputs into a subspace and then trying to recover the original input from this latent representation. One possible application are learned compression algorithms. By enforcing some kind of regularity over the latent space (Variational Autoencoders), these models can be used as generative models and for sampling new data.

- Generative Adversarial Network (GAN) [14]: A typical GAN consists of two networks named *Generator* and *Discriminator*. The generator tries to generate data, s.t. this matches the actual ground truth data distribution as well as possible, while the discriminator tries to distinguish between generated and real samples. Using this, among others fascinating images can be "hallucinated".

- Recurrent Neural Networks (RNNs) [13]: So far, all mentioned models can only handle static, non-changing inputs of fixed length. But what, if length of the input data is not known in advance, for example in the case of sequential data streams? For this, RNNs were introduced. They excel in the processing of time-series data by sharing parameters over time and processing sequential inputs by storing a memory state. Figure 2.2 visualizes such a network. In the next section, we will introduce RNNs in more details.

2.2 Recurrent Neural Networks

Since most of our proposed methods employ RNNs, in this section we introduce basic motivation behind them, explore advanced forms, as well as discuss training

procedures and drawbacks. To understand why RNNs are an important addition to the zoo of deep learning architectures, consider how other models, like CNNs, process inputs: Input of such models is data of a fixed size, for example an image of size 28×28. It is possible to process sequences of images, however this is done independently: In each step, the CNN accepts one image as input, and produces one output. When the next image is processed, no information from the previous one is available. This differs from how humans think, and is contrary to relations available in many problems. When we read a sentence like "The wolf ate the deer, because it was hungry.", we know to which noun "it" refers to, as we are able to memorize information from previous timesteps. These principles, being able to process sequences of arbitrary length and saving information between states, describe the mode of operation of RNNs: RNNs are networks with looping connections, consider Figure 2.2. At timestep t, RNN network A processes input $\mathbf{x}_t$, and outputs a value $\mathbf{h}_t$. This information is fed back to the network, and is available when processing the next input $\mathbf{x}_{t+1}$. An alternative way of visualizing these relations is to *unroll* the network over time - this is depicted in Figure 2.3.

This intuition of a simple RNN can be formalized as follows: Let $\mathbf{x}_t$ be the corresponding input at timestep t. A way of memorizing information for the RNN is its hidden state $\mathbf{h}_t$. Its calculation depends on the previous hidden state as well as current input:

$$\mathbf{h}_t = f(\mathbf{U}\mathbf{x}_t + \mathbf{W}\mathbf{h}_{t-1}) \tag{2.2}$$

$\mathbf{U}$ and $\mathbf{W}$ are weight matrices of according size, f can be any function, often a non-linearity like `ReLU` or `tanh` is used. Finally, in each step an output $\mathbf{y}_t$ is produced:

$$\mathbf{y}_t = g(\mathbf{V}\mathbf{h}_t) \tag{2.3}$$

Again, $\mathbf{V}$ denotes a weight matrix, and g some function, such as `softmax`, in case a multi-class classification problem is addressed.

RNNs are trained with a variation of the standard *Backpropagation* algorithm, dubbed *Backpropagation Through Time* (BPTT). Let E_t denote the error at timestep t. When now for example calculating the gradient of E_t w.r.t. $\mathbf{W}$, i.e. $\frac{\delta E_t}{\mathbf{W}}$, this depends on the current state $\mathbf{s}_t$, but according to the chain rule also on all previous states $(\mathbf{s}_{t-1}, \ldots, \mathbf{s}_0)$. Similar to a standard, deep CNN, where gradients have to be backpropagated through all layers back to the input, here they have to be backpropagated over time, while keeping in mind that parameters over each step are shared. Due to this, gradients for $\mathbf{W}$ are summed at each timestep. One problem of this approach is its computational complexity and resource consumption: Sequences can easily consist of thousands of timesteps, making full BPTT very slow. A common way of tackling this problem is to truncate backpropagation to k steps through the network (*Truncated Backpropagation Through Time* - TBPTT). This though only results in an approximation of the true gradient, but empirical results indicate its success. Further, a meaningful selection of k helps training, in addition states can be saved in-between batches, s.t. in theory longer dependencies can be learned still.

However, aside from this algorithmic restriction two other problems arise when applying RNN architectures to time-series data: Recall that the unrolled networks become "deeper" the longer the sequences get. Therefore, due to the chain-rule, gradients are calculated by recurrent matrix multiplications. In consequence, gradients smaller than 1 tend to exponentially shrink towards zero, while gradients larger than 1 bloat towards infinity. The first case is known as the *Vanishing Gradient* problem, and results in models being unable to learn longer sequences. Case number two is called *Exploding Gradients*, and also prevents any meaningful learning to happen. Note that these problems can arise for any deep learning architecture, but it can be argued that RNNs are more prone to it, due to their internal structure and large number of timesteps.

The problem of exploding gradients can be addressed by a rather simple fix: In each step, clip each occurring gradient s.t. its norm does not exceed a certain maximal value. This common technique is known as *Gradient Clipping*.

Alleviating the problem of vanishing gradients, and in general discovering and learning long-term dependencies in sequential data, unfortunately is much harder. In the next section we introduce one possible method of doing so.

2.2.1 Long Short-term Memory Networks

One way of addressing the issue of learning long-term dependencies is the usage of so called *Long Short-term Memory Networks* (LSTMs) [15]. They were introduced by Schmidhuber and Hochreiter in 1997, and since then have been employed successfully for many problems from different fields. (We should mention, that there are in fact many proposed architectures for handling long-term dependencies - such as *Gated Recurrent Units* (GRUs) [16] - but since we will be using LSTMs throughout, we will not go into details of these.) Recall the definition of a simple RNN network (Equation 2.2): A state $\mathbf{s}$ is kept and updated over time, to model memory of such network. LSTMs follow a similar principle, but use two separate values for storing temporal information, namely *cell state* ($\mathbf{c}$) and *hidden state* ($\mathbf{h}$). Encapsulating with `LSTM` the internal calculations of one step, Equation 2.2 thus becomes:

$$(\mathbf{h}_t, \mathbf{c}_t) = \texttt{LSTM}(\mathbf{x}_t, \mathbf{h}_{t-1}, \mathbf{c}_{t-1}) \tag{2.4}$$

In addition, LSTM networks contain several *gates*, such as an input and forget gate, which control the flow of information. A general gating function $\mathbf{g}$ of a vector $\mathbf{v}$ returns an element-wise `sigmoid` function, and its output subsequently is multiplied with $\mathbf{v}$. Due to the range of `sigmoid` being $(0, 1)$, this resembles an element-wise filter-out or pass decision for each value stored in $\mathbf{v}$.

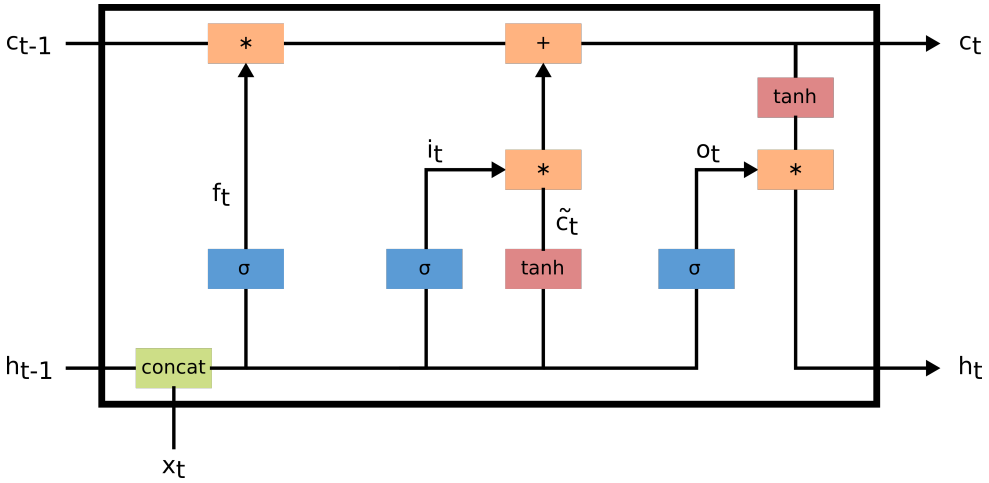


Figure 2.4: Depiction of an LSTM cell: The complete flow of information as explained in Equation 2.5 is shown. In each timestep, previous cell and hidden state c_{t-1} and h_{t-1} are passed to the cell, as well as the input x_t of timestep t. Using the described gates, a new cell state is calculated, from which a new hidden state is derived, which serves as output at timestep t (visualization motivated by [1]).

We first list all equations describing the internal operations of one LSTM cell, and then explain each in detail:

$$
\begin{aligned}
\mathbf{f}_t &= \sigma(\mathbf{W}_f \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f) \\
\mathbf{i}_t &= \sigma(\mathbf{W}_i \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i) \\
\tilde{\mathbf{c}}_t &= \texttt{tanh}(\mathbf{W}_C \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_C) \\
\mathbf{c}_t &= \mathbf{f}_t * \mathbf{c}_{t-1} + \mathbf{i}_t * \tilde{\mathbf{c}}_t \\
\mathbf{o}_t &= \sigma(\mathbf{W}_o \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o) \\
\mathbf{h}_t &= \mathbf{o}_t * \texttt{tanh}(\mathbf{c}_t)
\end{aligned}
\tag{2.5}
$$

$\mathbf{f}$, $\mathbf{i}$ and $\mathbf{o}$ denote the forget, input and output gate of the cell. All are identical in structure, and just differ in used weight matrices and biases: Based on the previous hidden state $\mathbf{h}_{t-1}$ and current input $\mathbf{x}_t$, output of $\mathbf{f}$ and $\mathbf{i}$ can be interpreted as a quantitative measure of how much old information shall be forgotten and how much new information shall be stored, respectively. $\tilde{\mathbf{c}}_t$ describes a potential new cell state candidate: Based on $\mathbf{h}_{t-1}$ and $\mathbf{x}_t$, decide which new information shall be present in the new cell state. $\mathbf{f}_t$ and $\mathbf{i}_t$ are now used to balance between old state $\mathbf{c}_{t-1}$ and new information $\tilde{\mathbf{c}}_t$, resulting in $\mathbf{c}_t$, the resulting state of step t. Eventually, the output gate $\mathbf{o}$ decides how much of the cell state to output, resulting in the hidden state $\mathbf{h}_t$. See Figure 2.4 for a visualization of such cell.

2.2.2 Drawbacks of Recurrent Neural Networks

Although LSTMs work very well for the problems analyzed in this thesis, there exists a trend implying to replace RNNs, and we would like to mention it for completeness. Two main drawbacks of RNNs are their limited ability of understanding long-term dependencies, and their resource consumption, especially time required for training and inference. Although e.g. LSTM cells significantly lessen the first problem, still every two points in time are connected via linear paths, requiring information to travel long distances. In practise, it has been shown that LSTMs are much better in modelling such dependencies than RNNs, but still are limited in their capabilities, and nowhere near the theoretical limit of being able to model infinitely long dependencies. In addition, due to the sharing of states and weights, for both training and inference, all sequences have to be processed sequentially, any calculation for time step t can only be done once all previous time steps have been calculated.

To address these issues, several alternative architectures have been proposed, among others using only feed-forward networks and attention [17] or (dilated) 1D-convolutions [18]. All methods allow for a parallelization of calculation steps, and some additionally focus on reducing path lengths between timesteps to sub-linear.

To end, note that for problems, in which input is processed online, such as autonomous driving, during inference no speed-up can be achieved using these methods, as input $\mathbf{x}_t$ only becomes available at timestep t.

2.3 Predicting Multiple Hypotheses

For most problems models are commonly trained to select and output the best possible hypothesis, e.g. predicting the most likely class of an object present in the scene, or the best translation of a sentence to a different language. Often though, ambiguity is present in problems, and a single prediction is neither desired nor correct: For difficult image classification instances, we might have k likely guesses, while on the other hand being able to definitely rule out all other classes. In textual, video or trajectory sequences, many possible futures are thinkable. Thus, one important, and currently heavily researched aspect of machine learning is the ability of predicting multiple hypotheses, and measuring uncertainty in the resulting predictions.

Although, in theory, predicted probability distributions over outputs are available (consider e.g. the outputs of a `softmax` layer for classification), this distribution usually is biased and overconfident (due to several reasons, such as the existence of unbounded gradients when employing the common `ReLU` activation function). This gives rise to the need for models specifically designed for such a task. In Chapter 4 we discuss and explore a newly proposed framework, and here give a brief overview over existing methods.

One such concept are *Mixture Density Networks* (MDNs) [19]. Core of these is modifying common regression models to output parameters describing a Gaussian mixture distribution, namely n-dimensional means, variances and weighting coefficients. During training, the standard approach of maximizing likelihood is used, in

which now the used probability is modelled as said Gaussian mixture distribution. During inference, for each point the conditional probability of corresponding output variables is returned, s.t. one can sample from these or estimate uncertainty and ambiguity.

Another commonly used method is *Multiple Choice Learning* [20] (MCL). Here, M separate copies of a base model are used as an ensemble, in order to generate M outputs. In early stages, more complex training procedures were used for optimizing such models, before Lee et al. introduced a modification of backpropagation for efficient training [21]: Gradients are only backpropagated to the lowest-error predictors, which results in the forming of M experts for different inputs, and encourages diverse predictions.

(Conditional) Variational Autoencoders (CVAEs) are yet another general method of outputting multiple hypotheses. Variational Autoencoders differ from regular autoencoders in that they enforce some kind of structure over the resulting latent space, in particular the Kullback-Leibler divergence between the latent distribution and a Normal distribution is added to existing losses. This way, we now can sample from the latent space, and by employing the decoder part of the autoencoder generate multiple samples in the output space.

2.4 Transfer Learning

In early times of machine and deep learning, all problems and tasks were addressed separately and models trained independently: Models for classifying images of animals had no connections to models detecting cars, language translation models were kept separate from question answering models. However, in more recent times, there is an always-growing interest of leveraging knowledge between domains and tasks. This is similar to how we humans learn: When faced with a new challenge, like a new ball sports, typically we can access a large amount of prior knowledge, such as physical constraints and skills in related disciplines. What makes humans exceptional at grasping new concepts, should also work for machines: Transfer learning describes the field of re-using existing knowledge and models between domains and tasks. Possible advantages are faster learning, and the lesser need of data, due to already stored prior knowledge. The latter point is especially important for deep learning techniques, as these commonly require huge amounts of data for training from scratch.

In the following, we give a brief overview over transfer learning techniques, in the process closely following [22]. In general, one can distinguish between *inductive*, *transductive* and *unsupervised* transfer learning methods:

- Inductive Transfer Learning: In inductive transfer learning, source and target domain are identical, but the tasks to be solved differ. One example is multi-task learning, in which a model simultaneously learns to solve different tasks on the same dataset.

- Transductive Transfer Learning: Here, in both domains identical tasks are addressed, but the domains differ. Often, less data is available in the target domain, resulting in the need for transfer learning techniques.
- Unsupervised Transfer Learning: This resembles inductive transfer learning, where domains are similar but tasks different. However, no labelled data is available in any domain.

In our works, we focus on the fields inductive and transductive transfer learning.

Now that we defined the taxonomy of transfer learning, we take a look at what kind of methods exist for actually implementing algorithms. There exists a multitude of methods, each of which can be applied in at least one of the above introduced use cases. We concentrate on methods applicable to deep learning architectures:

- Off-the-shelf feature extraction / fine-tuning: Assume, we are given a complex model M trained on some domain and task. To transfer this knowledge to a new domain or task, cut off the last L layers of M, and train a new, shallow model (such as an SVM) on this intermediate representation. Alternatively, we can freeze most parts of M, only retraining layers L on our new domain or task.
- Domain Adaptation: In contrast to retraining models on new domains to achieve better performance, the field of domain adaptation aims at minimizing differences between domains, s.t. models trained on the source domain also work in the target domain. Possible methods range from simple mean and variance transformations [23] to more complex, generative ones [24].
- Zero-shot / one-shot / few-shot Learning: Methods from these fields can be counted to the field of transfer learning, as well, since their purpose is transferring prior knowledge to unseen or rare data samples. Although many possibilities exist, a common one is using Siamese networks to first learn similarities between objects from the training set, and then applying this knowledge for categorizing unseen ones [25].
- Multi-task learning: Core idea of multi-task learning is learning different tasks simultaneously while using a shared base architecture. Possible applications are the joint learning of semantic segmentation and depth estimation or combining perception and planning losses in end-to-end autonomous systems. Sharing parameters over different tasks has been shown to improve performance, among others due to the additional amount of labelled data and exploitable relations between tasks [26, 27].

2.5 Autonomous Driving

Autonomous driving undoubtedly is one of the most exciting fields of applied research nowadays, with the potential of changing the world as we know it: Through

using powerful sensors as well as computing resources and removing human errors from the equation, fully autonomous driving is expected to lower the number of traffic accidents drastically - some even share a vision of zero accidents. Currently, unfortunately more than a million people each year die in road traffic incidents [28]. Furthermore, also the number of harmful emissions caused by traffic is expected to reduce significantly: Vehicle-to-vehicle as well as vehicle-to-infrastructure communication allow a better traffic control and planning, avoiding congestions and unreasonable acceleration peaks. Fully autonomous driving even reduces the needed number of parking possibilities in urban areas, as vehicles might drive autonomously to designated, compact parking areas in less crowded environments - thus freeing up space for people and nature.

History The dream of autonomous vehicles has been pursued for over thirty years now: Already in the 1980s, vehicle Alvinn conducted first autonomous drives on highways [29]. By the end of the 90s, an autonomous shuttle bus, billed the "world's first driverless vehicle", was put into operation, transporting people on designated routes by making use of artificial magnetic reference points [30]. In the next century, DARPA's Grand Challenge resulted in much progress in the field: In its first holding, vehicles were required to autonomously follow a 150-mile course through the Mojave Desert, but no team made it [31]. A year later, several vehicles finished the course, and 2007 the challenge was moved to an urban environment [31]. In 2012, researchers from BMW managed to routinely drive in full autonomous fashion from München to Ingolstadt through dense highway traffic [32]. In recent years, progress and interest in the field has grown even more, with many major tech companies investing in and developing self-driving vehicles, to just name a few: Google, Apple, Intel, Uber and Lyft. By 2020, Waymo's autonomous vehicles had driven 20 million miles on public roads [33]. Still, no fully autonomous vehicle is commercially available yet, and plans for doing so have been pushed backwards as well as confined to certain scenarios and locations, such as highways and specific cities. Experts agree though that the goal is in graspable reach, and that deep learning will be one of the main drivers for this.

Levels of Automation Autonomous driving typically is partitioned into five levels, which model a successive transfer of responsibility from driver to vehicle: *Level 0* describes a classical vehicle without any driving assistance functions. A *Level 1* vehicle contains such functions, although these mostly are of indicatory nature, informing and warning the driver. *Level 2* describes partial automation, for the first time actively accessing the vehicle actuators: Steering and control assistants can guide the vehicle, e.g. in lane-following or traffic jam scenarios, such as an ACC. Earlier, most such functions focused on longitudinal control, while now also lateral control is possible (e.g. lane changes can be executed autonomously). Still, the driver bears full responsibility for the vehicle and has to watch and monitor the traffic situation at any time, immediately being able to regain control once the

system reacts wrongfully. Most commercially available vehicles nowadays contain Level 2 functions. *Level 3* vehicles exhibit conditional automation: Under certain circumstances, drivers are allowed to divert their attention from the road and focus on other tasks. Still, within a limited time period, drivers need to be able to re-focus on the road, controlling and guiding the vehicle through situations it is not able to handle. One difficulty when developing such functions is equipping these with ways of knowing, whether automation for the next n seconds is possible. One distinction between Level 3 and *Level 4* automation is the vehicle's ability to handle failures and unforeseen situations: While in Level 3 drivers have to regain control of the vehicle, a Level 4 system will always stay in a safe state even without the driver's intervention. This could mean the vehicle coming to a safe stop on the road side, when it is not able to safely continue on its trip. Therefore, drivers need to be able to drive in general, moving their vehicle across critical passages until automation is possible again. *Level 5* describes full automation, absolutely no human interaction is necessary. Level 5 vehicles could actually be produced without any typical input components we see today, such as steering wheels or pedals.

Sensors To be able to progress through these levels of automation, vehicles need to understand complex traffic scenes. Crucial for this is the usage of powerful sensors, s.t. the vehicle can perceive its environment. Most common sensors are cameras, Radar and LiDAR, each coming with their own distinct advantages as well as disadvantages. It is believed that a combination of all sensor types will be necessary to achieve full automation, although Tesla seems to choose a different path [34]. Cameras are a well-working and mature technology. Combing them with deep neural networks results in state-of-the-art methods for many tasks needed for autonomous driving, such as object detection and semantic segmentation. There are downsides though, which actually closely resemble those of our human eye: Adverse weather conditions, such as rain or snow, severely influence camera performance. Radar sensors emit electromagnetic waves, determining range but also velocities of objects based on reflection patterns. Electromagnetic waves are hardly influenced by weather conditions, thus Radar is ideal in these scenarios. The ability to determine relative velocities can come in handy, as well. On the downside, current Radar sensors only offer limited resolution, resulting in blurry images and making it hard to react precisely to small, localized threats. LiDAR sensors, on the other hand, are able to generate incredibly detailed 3D maps of the environment. They work by sending out laser pulses and measuring their reflection. With such accurate measurements, one can achieve very precise localization of vehicles in High Definition (HD) maps using Structure-from-Motion (SfM) techniques. However, LiDAR sensors exhibit a limited range only, and additionally are very expensive, in terms of production costs but also needed computation power.

Architectures For building a self-driving vehicle, two philosophies exist, with various implementations in-between: More traditional approaches employ a stack-based

architecture consisting of multiple layers, on the other hand data-driven end-to-end systems aim at solving the whole problem with a single component. A stack-based architecture generally consists of the following layers:

- Perception: The perception layer handles processing of information from all available sensors and a subsequent detection of objects and other points of interest. Typically at this stage such processing still is done separately per sensor, e.g. outputting a list of detected objects or other relevant information per sensor.
- Fusion: The fusion layer then fuses these into a complete understanding of the surrounding environment. Such understanding ideally should be as complete as possible, and include among others lane markings, detected pedestrians and vehicles, as well as traffic lights and symbols.
- Prediction: Goal of the prediction layer is to transform the previously built static environment understanding into a dynamic one. How will the scene unfold, where might other agents go?
- Planning: Based on these assumptions and observations, one now can plan a possible path for the ego vehicle. One can further distinguish short- and long-term planning: Long-term planning describes the finding of certain waypoints the vehicle has to approach, such as highway exits or turning lanes. Complementary, short-term planning calculates trajectories to meet these waypoint requirements.
- Control: Purpose of the control layer is to translate the given trajectories into actual commands for on-board electronic control units (ECUs), i.e. actuate steering and acceleration.

Motivated by breakthroughs in other fields, such a image classification and speech recognition, there also exist approaches to replace this multi-step pipeline by just a single component. End-to-end learning describes exactly this, namely going from raw sensor inputs to steering commands. Possible approaches are [35, 36, 37, 27], which use camera images, LiDAR point clouds or some intermediate representation as inputs, and employ IL and RL techniques. Both approaches, stack-based and end-to-end systems, exhibit their own advantages and disadvantages: A stack-based system more resembles principles from classical software engineering. Common functionality is bundled and encapsulated into different components, which can be developed and tested independently. This allows parallel development and enables a better understanding and verifiability of the system. However, in each abstraction layer most likely information is lost, and each component is designed from scratch, neglecting the potential many complex interactions between them. On the other hand, an end-to-end architecture leverages these interactions, all components profit from all available data and assumptions and share their knowledge. However, training such a model is much more complex, and finding errors is more tedious, as know the whole system essentially is a black-box model.

This work more fits into a stack-based architecture and mainly considers the fields planning and prediction. Thus, in the following, we take a closer look at these. Of course, mixed solutions are possible and applicable, and any end-to-end system implicitly models planning and prediction targets. Therefore also methods from these areas will be addressed.

2.5.1 Prediction

The field prediction turns a static scene understanding into a dynamic one. As agents typically move around a scene, just examining a static snapshot is not sufficient. Predicting future movements thus is a crucial part for any autonomous system and builds important foundations for later planning steps. Methods used for prediction range from simple ones, such as predicting linear follow-up trajectories or modelling rational drivers in lane following scenarios [38] to more complex ones, e.g. using deep learning techniques.

We can distinguish between discrete and continuous prediction: Discrete prediction aims at predicting discrete manoeuvres, such as lane changes and turning manoeuvres [39, 4]. This often gives a good high-level understanding of dynamic scenes, and is beneficial in situations where such priors are available. Urban or highway environments are one such example, as here designated lanes are available which are followed by drivers. Restricting predictions to discrete, possible manoeuvres drastically simplifies the hypotheses space compared to allowing arbitrary, continuous predictions. Continuous prediction requires less prior knowledge, but often comes with a greater expressiveness and flexibility. Aim is predicting continuous trajectories. Formally, a trajectory is defined as the path an object follows through its spatial environment in relation to a temporal component. Thus, a trajectory defines a spatial path annotated with temporal information, i.e. expressing at what time an object reaches which coordinates.

Many works regarding trajectory prediction involve predicting movements of VRUs [40, 41]. Possible solution techniques include encoder-decoder RNNs [40, 41] and generative models [42, 43]. Often, agent-agent interactions as well as agent-environment interactions are taken into account, sometimes coupled with an attention mechanism to quantify the strength of such interactions [41, 40, 43].

2.5.2 Planning

The planning layer builds on the previously established dynamic scene understanding and tries to find an optimal path considering different constraints for the ego vehicle. The found solution should take into account physical limits and estimate felt driving comfort given different velocity profiles, lead towards a specified goal (such as a navigation target) and naturally avoid accidents and result in safe, law-abiding driving. We can further subdivide the planning layer: At a very high level, navigation information is used to calculate a route towards a certain target. This could include roads and highways to use and specifying which exits to take. One ab-

straction level below, this is further specified by finding discrete waypoints to model such route: Which lanes need to be used to reach a certain exit or turning lane on an intersection, which manoeuvres need to be executed to end on these waypoints? Lastly, a continuous trajectory needs to be generated, satisfying all previously described constraints and expressing a path containing all designated waypoints.

Common approaches are search algorithms and some kind of rule-based systems: Algorithms like A* or Dijkstra's algorithm can determine a navigation path via search in discretized environments, logic-based systems generate optimal manoeuvres given certain situations, and mathematical optimization is used to find continuous trajectories. Possible data-driven solutions are IL and RL. Depending on used inputs and outputs, these techniques can be applied in stack-based architectures as well as end-to-end systems.

IL is a supervised learning method, in which models try to imitate expert behaviour. In the case of autonomous driving, large amounts of data are recorded daily by companies all over the world. This data essentially defines a ground truth of (near) optimal driving behaviour, which can be used to train complex models [27]. Downsides of this technique are potential overfitting to certain expert behaviour, as well as compounding errors, which lead to a degradation of trajectories over time from which the system cannot recover.

Core principle of RL, in contrast, is allowing the model to teach itself by rewarding or punishing it for certain actions. Typically, this is done in some kind of simulator, in which any kind of driving behaviour can be executed safely. A reward function is designed (consider, e.g. driven kilometers without traffic accidents), and the model discovers optimal policies by itself [36]. While this sounds very promising, training is not easy, and we need to handle the domain gap between simulated and real-word driving data. Further, safety guarantees for learned behaviour are even harder to determine than for other deep learning methods.

Part II

Prediction

3 Manoeuvre Prediction

In most countries, traffic is highly regulated - designated lane markings confine vehicles to certain areas, numerous traffic rules and signs control its flow. It thus makes sense to discretize prediction space, and in our first prediction application we will do exactly that: Goal is to predict discrete manoeuvres, in particular predict lane change manoeuvres on highways. Obtaining a precise prediction for these manoeuvres is extremely important, also for many driving assistance functions commercially available today, such as an ACC: in lane following situations, the ego vehicle needs to react to vehicles merging into one's driving lane. The earlier and more accurate the prediction, the smoother the reaction to this can be controlled.

For this, we require a prediction model which is able to predict forthcoming lane changes early, while simultaneously minimizes the number of false positive predictions, as these could lead to wrong and abrupt breaking manoeuvres. We develop a powerful RNN achieving promising prediction results. To better quantify such cases and further relate this to driving comfort felt by the driver, we additionally introduce novel and better suited metrics.

In summary, our contributions are:

- We introduce a novel RNN model for predicting lane changes. We further extend this by an attention mechanism, which results in improved prediction performance as well as allows insights into which features the model deems important. Such capability of being able to explain decisions is a very interesting field of research, and especially important in safety-critical applications.
- We introduce novel metrics, specifically designed for the problem of predicting lane changes. These directly relate to felt driving comfort, and thus allow exact quantification how changes in prediction performance effect customer satisfaction.
- We provide extensive experiments and benchmarks of different algorithms, comparing them on multiple datasets and further also introduce the study of various corner cases.

3.1 Introduction

In the field of autonomous driving, predicting lane change manoeuvres is a field of great interest, due to its high importance for many driving assistance functions. More generally, it strongly relates to activity anticipation. Jain et al. introduce a framework for predicting driving manoeuvres of the ego vehicle, making use of internal and external driving features, such as driver's head pose and map information [39]. They use different, decoupled RNN cells and achieve impressive results for this task, and greatly motivated our work. For predicting lane changes, often more classical, frame-based methods are used: Random Forests offer good results [44], further SVMs [45] and Bayesian Networks [46] are employed. Schlechtriemen et al. [47] analyze the expressive power of different input features, concluding that the features distance to the lane boundaries, lateral velocity and relative velocity to the preceding vehicle are most decisive.

We propose to use an RNN made out of LSTM cells for tackling this problem, as it is our strong belief, that lane change prediction essentially is a time-series problem, and thus ideally suited for recurrent models. Understanding temporal dynamics of the problem and differences between frames naturally gives a greater understanding and should allow better and more stable predictions. Consider a driver oscillating around lane boundaries: While classical, frame-based models might predict alternating lane changes, a recurrent model could learn this behaviour and smoothly predict a follow manoeuvre. Additionally, we employ an attention mechanism. This further improves performance as well as allows insights into the decision-making process of our model.

In accordance with many works in this field, we work with intermediate feature representations instead of raw senor data. In particular, turn signals are not part of our feature set, as these are hard to detect and often unreliable [48]. Using such high-level representations generally leads to good performance, and allows for better transfer to different domains, and is compatible with the concept of a stack-based architecture.

Often, metrics motivated by information-retrieval applications, such as *Precision* and *Recall*, are used to assess lane change prediction algorithms. However, it is not clear, how these numerical results translate to real driving behaviour on the road. Therefore, we introduce novel metrics, directly tying our prediction performance to actual driving experience, when such a prediction model is used as input for different control functions.

As stated before, results indicate a high importance of few core features, such as distances to the lane boundaries and lateral velocity. In addition to their obvious expressiveness, another reason for this is that a large amount of lane changes can be classified as "easy" to predict: In many cases, drivers will smoothly initiate and execute a lane change manoeuvre, and otherwise follow their intended lanes in an orderly fashion. This makes improving existing models hard, and also complicates a precise evaluation, as many metrics will be dominated by this large amount of "easy" manoeuvres. To combat this, we introduce a novel, scenario-based evaluation, for

the first time standardizing different driving scenarios and listing performances of different algorithms over multiple datasets.

In strong correlation with that, our models employ a multitude of different features, ranging from features of the target vehicle (such as lateral velocity) to dynamic environment features (distances to neighbouring vehicles) and static environment features (such as map information). These should greatly increase prediction performance of our model in certain corner cases, observable through the previously introduced scenario-based evaluation.

3.2 Related Work

Jain et al. employ LSTM networks for anticipating driving manoeuvres [39]. They use various sensory inputs, partitioned into inside and outside features: Inside features describe the driver's behaviour, such as head pose. Outside features conversely represent the outside world and contain among others map information, such as distances to intersections. To better be able to model different modalities of such feature streams, the authors use two separate RNNs and later fuse the results. Using this scheme, manoeuvres can accurately be predicted around 3.5s before they are actually executed by the driver.

Schlechtriemen et al. analyze a multitude of features for predicting lane changes, coming to the conclusion that the features distances to the lane boundaries, lateral velocity and relative velocity towards the preceding vehicle are most telling for such a prediction [47]. Based on these features, they employ Naive Bayesian methods as well as extend them with a Hidden Markov model. In another work, Schlechtriemen et al. consider the problem of predicting trajectories [44]. For this, lane change prediction is queried as a helping subtask, and solved by using Random Forests. Weidl et al. employ Dynamic Bayesian Networks (DBNs), and use hand-crafted safety distances as additional inputs to model physical plausibility of lane change manoeuvres [46]. Further deepening this topic, Woo et al. propose usage of an energy field to express relationship between vehicles and free space [45]. This combined with other features serves as input to an SVM model. Combining scene understanding with deep learning, Patel et al. apply Structural-RNNs to model lane change behaviour [49]. Three LSTM cells are used to model the target's driving lane as well as its adjacent lanes to the left and right. Information is exchanged between these cells, s.t. a full scene understanding is achieved. Zeisler et al. approach the problem differently, using raw video inputs as prediction cues, in particular optical flow [50].

Attention is widely used in fields like machine translation and image processing [51, 52, 53]. Basic idea is to allow models a selective focus onto different input parts. Consider a machine translation application: When generating a word in the target sentence, specific key words in the source sentence are particularly telling regarding which translation is correct. Attention mechanisms weigh these words according to their importance. In general, needed values are called queries, keys and values: W.r.t. a query, different importance scores for all keys are calculated,

and an accumulated representation of all values based on these scores is determined. More formally, we assign an importance weight a to each value $\mathbf{v}_i$, which is based on a score between the query $\mathbf{q}$ and the keys $\mathbf{k}$:

$$a(\mathbf{v}_i) = \frac{\texttt{exp}(\texttt{score}(\mathbf{q}, \mathbf{k}))}{\sum_{\mathbf{k}'} \texttt{exp}(\texttt{score}(\mathbf{q}, \mathbf{k}'))} \tag{3.1}$$

Eventually, based on this a weighted accumulation $\mathbf{c}$ is calculated:

$$\mathbf{c} = \sum_{\mathbf{v}_i} a(\mathbf{v}_i) \cdot \mathbf{v}_i \tag{3.2}$$

For calculating score functions, a multitude of possibilities exist, such as [53]:

$$\texttt{score}(\mathbf{q}, \mathbf{k}) = \begin{cases} \mathbf{q}^T\mathbf{k} & \text{dot} \\ \mathbf{q}^T\mathbf{W}\mathbf{k} & \text{general} \\ \mathbf{v}^T\texttt{tanh}(\mathbf{W}[\mathbf{q};\mathbf{k}]) & \text{concat} \end{cases} \tag{3.3}$$

For some applications, just using attention even outperforms RNNs, like shown in the Transformer [17]. Here, queries, keys and values are obtained by projecting similar input values into different latent spaces (self-attention), and this in combination with feed-forward networks results in state-of-the-art machine translation performance.

3.3 Recurrent Models for Predicting Lane Changes

In this section we introduce our proposed model. We begin by formally introducing the problem and used feature set.

3.3.1 Scene Representation

A driving scene may contain N vehicles. For each vehicle n, we collect a sequence s_n of all frames it occurs in, $s_n = \{F_1, \ldots, F_L\}$. Each frame F_t, $t \in \{1, \ldots, L\}$, is a numerical representation of the driving scene at time t and is centered around vehicle n. This way, vehicle n acts as a "target" vehicle in s_n, and s_n is used for obtaining predictions about its behaviour. We subdivide each frame into three feature categories G_t^Z, G_t^E, G_t^M:

- $G_t^Z = (m, v_{lat}, v_{long}, a_{lat}, h)$. G_t^Z describes *features of the target vehicle*. They are, in order, distance to the lane's center line, lateral velocity, longitudinal velocity, lateral acceleration and heading angle. All are measured in Frenet coordinates: These follow a lane's center line, describing a point by its longitudinal distance along this (possibly curved) line and lateral offset.
- $G_t^E = (dt_X \text{ for } X \in \{\text{PV}, \text{RV}, \text{PLV}_L, \text{PLV}_R, \text{PFV}_L, \text{PFV}_R\})$ describes *dynamic environment features*, namely relations of the target vehicle to neighboring vehicles. In particular, we measure temporal distances, which is the

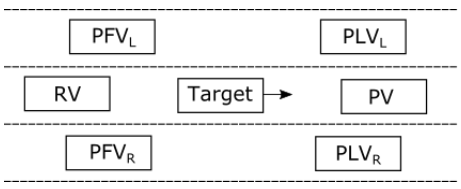


Figure 3.1: Depiction of all considered neighbouring vehicles of vehicle "target": On each lane, a preceding and following vehicle is observed and used for further calculation (image from [2]).

spatial distance divided by the velocity of the trailing vehicle. Any car can have at most six neighbours, a preceding and following vehicle on its left and right neighbouring as well as current driving lane. See Figure 3.1 for a visualization.

- G_t^M denote *static environment features*, which slightly depend on the dataset used and amount of information available. Overall, available features in this category are lane ID (l_{ID}), distances to next on- and off-ramps, and speed limit information.

Additionally, each frame is given a label l_t, which serves as training target during training and needs to be correctly predicted in each step during inference. Four possible labels exist: *Left* (L), *Follow* (F), *Right* (R) and *Ignore* (I). Models need to output one prediction every timestep. Predicting L or R indicates the model's belief, that a lane change to the respective side is imminent. Conversely, predicting F expresses the model's belief that the vehicle will continue in its driving lane. What models understand as "imminent", strongly corresponds to our labeling scheme, which we will introduce in the next paragraph. Label I is used to assign a weight of 0 to certain frames, and thus effectively removing them from training and test set.

Labelling Scheme We use *automatic* and *manual* labelling techniques. Assume, a lane change happens at time step t_1 (that is, the target's center of mass crosses lane boundaries). In the automatic labelling scheme, we label all frames in the window t_1 - 3 seconds to t_1 with the corresponding lane change label. Frames in $[t_1 - 5, t_1 - 3)$ are labelled with I. This way, we do not punish early predictions. In the manual labelling scheme, lane change frames are labelled via human labelers. They assign lane change labels when humans perceive a lane change as such, and thus the duration of a lane change is variable. In both schemes, we label 5s after a lane change with I to give models time to adjust. Additionally, we filter out "noisy" lane changes, i.e. driving sequences where drivers oscillate around lane boundaries. All other frames are labelled with F.

3.3.2 Models

In this section we introduce used models for predicting lane changes. We first introduce two "baseline" LSTM networks, which we then extend by an attention mechanism. In accordance with Section 2.2.1 we use the following short-hand notation to encapsulate internal calculations of an LSTM cell per timestep:

$$(\mathbf{h}_t, \tilde{\mathbf{c}}_t) = \texttt{LSTM}(\mathbf{x}_t, \mathbf{h}_{t-1}, \tilde{\mathbf{c}}_{t-1}) \tag{3.4}$$

3.3.2.1 Baseline Models

In our first proposed recurrent model (named *LSTM*), a single LSTM cell processes all available input features G_t^Z, G_t^E, G_t^M. A subsequent fully connected layer then generates a manoeuvre prediction in each step. This is expressed mathematically as follows:

$$\begin{aligned} (\mathbf{h}_t, \tilde{\mathbf{c}}_t) &= \texttt{LSTM}(\texttt{concat}(G_t^Z, G_t^E, G_t^M), \mathbf{h}_{t-1}, \tilde{\mathbf{c}}_{t-1}) \\ \mathbf{y}_t &= \texttt{softmax}(\mathbf{W}_s \mathbf{h}_t + \mathbf{b}_s) \end{aligned} \tag{3.5}$$

Our next variant is motivated by the approach from Jain et al. for predicting driving manoeuvres [39]: Core idea is partitioning features into groups, s.t. intra-group correlation is high and inter-group correlation low, and then processing these feature groups by separate LSTM cells, later fusing the results. This way, LSTM cells can focus on learning feature-specific patterns, such as different temporal modalities of the input streams. In our extended model (dubbed *E-LSTM*), we employ three different LSTM cells, which process the feature groups G_t^Z, G_t^E and G_t^M separately:

$$\begin{aligned} \mathbf{u}_t &= \mathbf{W}_F[\texttt{concat}(\mathbf{W}_X \mathbf{h}_t^X + \mathbf{b}_X)] + \mathbf{b}_F \\ \mathbf{o}_t &= \texttt{tanh}(\mathbf{W}_u \cdot \mathbf{u}_t + \mathbf{b}_u) \\ \mathbf{y}_t &= \texttt{softmax}(\mathbf{W}_o \cdot \mathbf{o}_t + \mathbf{b}_o) \end{aligned} \tag{3.6}$$

In this, $\mathbf{X} \in \{G^Z, G^E, G^M\}$ denotes the specific feature group. An overview over the model is given in Figure 3.2.

3.3.2.2 Attention Mechanism

We now extend both previously introduced base LSTM models with an attention mechanism. This way, models can focus on specific parts of the scene, and weigh input features accordingly. This approach is motivated by how humans assess scenes, especially while driving: Also humans use selective focus, and normally do not spend their full attention on full scenes, but often on the most important cues, such as vehicles in front or approaching on- and off-ramps. Modelling this algorithmically should help improve prediction quality, and also result in interpretable models. One often used side-effect of attention (or design purpose, depending on the interpretation) is a direct indication, what the model deems important in its decision-making. In machine translation, for each translated word in the target sentence, attention indicates

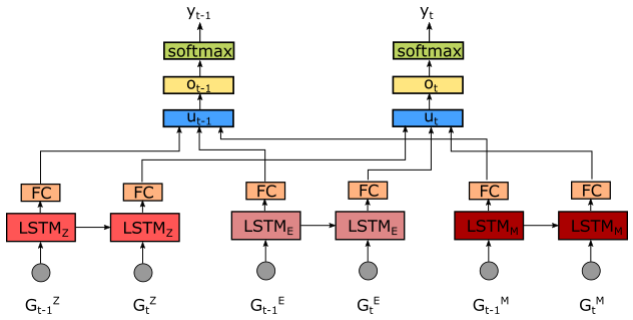


Figure 3.2: Visualization of *E-LSTM*. Each feature group is processed by an independent LSTM cell and subsequent fully connected layer (FC). The outputs are eventually fused together to return the final prediction $\mathbf{y}$ (image from [2]).

which words of the source sentence are most influential. In image processing, attention allows an insight into which image regions are considered for each classification. This becomes even more important in safety-critical applications, like autonomous driving, as assuring safety guarantees for black-box models such as neural networks is hard. For completeness, we should note though that there are other methods for gaining deeper insights into neural networks, as well. Some of these also are applied in practise with good success, such as *Saliency Maps* for CNNs [54]. We compare against such methods in Section 3.4, but find that results for RNNs are relatively unreliable.

We refer to Section 3.2 for a general introduction of attention and the terms queries, keys, and values. Here, hidden states of the respective LSTM cells serve as queries. We embed features into a higher-dimensional space, these representations simultaneously serve as keys and values, meaning we calculate importances based on them and accumulate them in weighted fashion. As score function we use Luong's general function [53]:

$$\Psi(\mathbf{W}, \mathbf{v}, q, k) = \mathbf{v}^T \texttt{tanh}(\mathbf{W}[\mathbf{q}; \mathbf{k}]) \tag{3.7}$$

We first introduce the attention extension of *LSTM*, and then apply the same concept to *LSTM-E*. A temporal window of T values for each feature is embedded into a higher dimensional space $\mathbf{E}$, and them embedded once more ($\mathbf{J}$), in order to achieve more comparable subspaces. With $\mathbf{J}$ as keys and hidden state $\mathbf{h}$ of the LSTM cell as query, the embeddings $\mathbf{J}$ (the values) are assigned importances via the score function Ψ. Based on these, after applying a `softmax` function, a weighted sum of the values is calculated, now denoting a compressed scene representation. This vector is concatenated with $\mathbf{h}$ and serves as base for the classification. Let $F = [G_t^Z, G_t^E, G_t^M]$ be the set of all 14 input features, and further let $\mathbf{X}_t^i$, $i \in F$, denote input value of feature i at timestep t. Then, the following describes the calculation of attention formally (while $\mathbf{W}$'s and $\mathbf{b}$'s denote weight matrices and

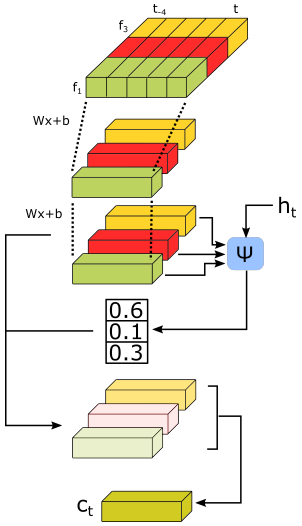


Figure 3.3: Visualization of our used attention scheme. For the sake of simplicity, in this example 3 features and a temporal window of size 5 is used. For each feature, these windows are embedded into a higher dimensional space, and projected once more. The resulting serves as key for the attention function Ψ, and the hidden state h of the LSTM cells is used as query. Ψ returns weighting factors for each such embedding, which are then changed accordingly in intensity. Eventually, this is summed together, resulting in an accumulated representation of the scene, the context vector $\mathbf{c}$ (image from [2]).

bias vectors):

$$\begin{aligned}
\mathbf{E}_t^i &= \mathbf{W}_{Ei}[\mathbf{X}_{t-T}^i; \ldots; X_t^i] + \mathbf{b}_{Ei} \\
\mathbf{J}_t^i &= \mathbf{W}_{Ji}\mathbf{E}_t^i + \mathbf{b}_{Ji} \\
\gamma_t^i &= \Psi(\mathbf{W}_{Fi}, \mathbf{v}_{Fi}, \mathbf{h}_t, \mathbf{J}_t^i) \\
\gamma_t &= \texttt{softmax}([\texttt{concat}(\gamma_t^i)]) \\
\mathbf{c}_t &= \sum_{i \in F} \gamma_t^i \cdot \mathbf{J}_t^i \\
\mathbf{u}_t &= [\mathbf{h}_t; \mathbf{c}_t]
\end{aligned} \tag{3.8}$$

Figure 3.3 visualizes the calculation steps involved in the attention mechanism. Eventually, using two fully connected layers, a prediction is obtained based on this:

$$\begin{aligned}
\mathbf{o}_t &= \texttt{tanh}(\mathbf{W}_o \cdot \mathbf{u}_t + \mathbf{b}_o) \\
\mathbf{y}_t &= \texttt{softmax}(\mathbf{W}_s \cdot \mathbf{o}_t + \mathbf{b}_s)
\end{aligned} \tag{3.9}$$

For *E-LSTM*, calculations are identical, except $\mathbf{u}_t$ now contains a concatenation of the hidden states of all three LSTM cells, and this is further used as query. We denote the extensions of *LSTM* and *E-LSTM* by *LSTM-A* and *E-LSTM-A*, respectively.

Training Scheme Similar to [39], we use an exponentially growing loss during training, increasing weighting of frames exponentially more the closer the moment of crossing lane boundaries. This way, correct classifications become more important the closer the lane change, modelling the huge cost of not detecting imminent lane changes. On the other hand, wrong predictions earlier in time are penalized much less, accepting that such predictions are much harder, and implicitly encouraging models for early anticipation. As loss function we use standard cross-entropy, weighing each frame t with w_t, which is inversely proportional to the class' relative frequency. Additionally, lane change manoeuvres are weighted by the previously introduced exponential weighting scheme, multiplying w_t with $\alpha \cdot \exp(-T)$, assuming the moment of lane change execution is T seconds away. We choose α s.t. the average value of $\alpha \cdot \exp(-T)$ over all frames of a lane change equals 1.

As many lane change manoeuvres could be predicted well using only a few very telling features, such as distance to the lane boundaries and lateral velocity, models could fall back to these, ignoring complex interactions and especially ignoring the attention scheme: Thus, we introduce dropout in the calculation of $\mathbf{u_t}$:

$$\mathbf{u_t} = [\mathbf{W}_{Drop,Fusion};\ \ \mathbf{W}_{Drop,c}] \cdot [\mathbf{h}_t;\ \ \mathbf{c}_t] + \mathbf{b}_{drop}$$

In this, default values of $\mathbf{W}_{Drop,Fusion}$ and $\mathbf{W}_{Drop,c}$ are 1, but set to 0 with a probability of 1/3 each, thus forcing models to only focus on direct feature inputs or attention, and creating a meaningful gradient flow in call cases. In addition to difficulties during training time, the large number of "easily" detectable lane changes also causes problems during inference and evaluation: Most simple metrics would be dominated by such lane changes, indicating that in order to fully understand our models, it is necessary to look at certain corner-cases. In the next section we describe methods for this.

3.4 Evaluation

In this section we describe detailed experiments evaluating the performance of our proposed LSTM models and compare them against other baselines. We begin by introducing used datasets and then explain used metrics, which are particularly suited for the problem of predicting lane changes. Eventually, we give detailed evaluations of a multitude of experiments, also analyzing feature importances and specific corner-cases, in order to better understand prediction models.

3.4.1 Datasets

NGSIM The publicly available NGSIM dataset [55] contains four subsets, we use US 101 and I-80. These are recorded on the respective highway / interstate in the United States. All contain traffic recordings of a certain area, recorded from a bird's eye view camera. Raw image data is available, as well, but also processed information, such as lane boundaries, vehicle coordinates and velocities. We use

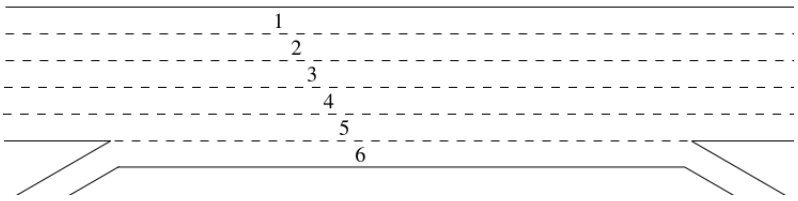


Figure 3.4: Depiction of the highway segment recorded in the US 101 dataset. Vehicles drive from left to right on 6 lanes, if also counting an auxiliary lane for merging from the existing on-ramp or towards the off-ramp. Size proportions are not realistic in this drawing, image taken from [3].

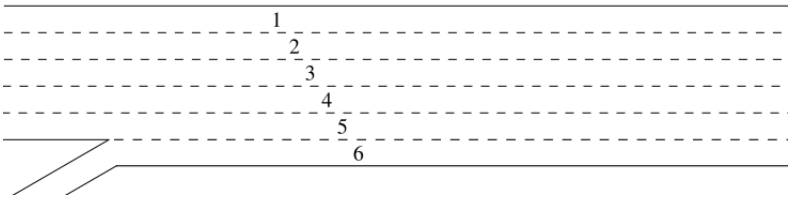


Figure 3.5: Visualization of the I 80 subset. As can be seen, the area of interest is similar to that in US 101, except missing the off-ramp.

the latter, using and calculating missing ones s.t. finally we obtain all features G_t^Z, G_t^E, G_t^M. The relevant segments of US 101 and I-80 are multi-lane roads, with one on-and off-ramp or just an off-ramp. Only one direction of traffic is recorded. Figures 3.4 and 3.5 show simplified drawings of US 101 and I-80. Recordings are done with a frequency of 10Hz. After filtering out noisy trajectories, we are left with 3180 lane changes. In addition to l_{ID}, we add the distances to the on- and off-ramp (d_{on}, d_{off}) to the feature set G^M.

highD In terms of structure and available features, the highD dataset [56] is very similar to the NGSIM dataset. Traffic is recorded at 25Hz on different German highway segments from a bird's eye view, and similar high-level features are calculated. In contrast to NGSIM though both traffic directions are recorded. Due to our used feature representation, this does not affect our models, but needs to be considered in the preprocessing step (i.e., normalize direction of travel and signs of velocity vectors etc.). After filtering, 7572 lane changes are available. We resample all tracks to 10Hz. Since no on- and off-ramps are present but speed limit information is available, in this case $G^M = \{l_{ID}, v_{max}\}$.

Fleet data In addition, for our experiments we use proprietary BMW fleet data, which comes from recordings of in-production cars. They are equipped with several camera and Radar sensors to allow a 360° perception of the environment. Data is collected at 25Hz, but again resampled to 10Hz. From the sensory inputs we calculate all needed high-level features. Figure 3.6 shows a sample recording of a

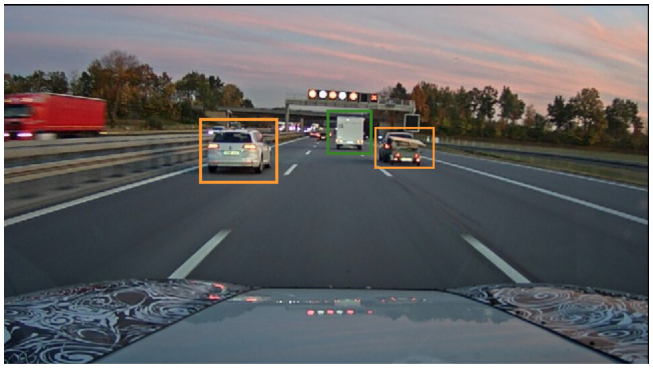


Figure 3.6: Sample scene recording from the front camera of a fleet vehicle. The target vehicle, which is doing a lane change, is marked by the green bounding box, while its neighbouring vehicles are highlighted in orange (image from [4]).

traffic scene from the front camera. After filtering, we obtain 830 lane changes. Here, G^M only contains $\{l_{ID}\}$. While for all other datasets we use an automatic labelling scheme, our fleet data is humanly labelled.

3.4.2 Metrics

Most works concerning the prediction of lane changes, but also manoeuvres in general, apply metrics such as Precision and Recall, which are motivated by information retrieval problems. These exhibit two downsides when applied to the problem of predicting lane changes: For one, they discard the fact that ground truth labels for such manoeuvres are event-wise continuous. This should also be reflected by used metrics, however, fast-changing and spiky predictions obtain identical scores as continuous ones, when the overall percentage of correctly classified frames is identical. Second, although such metrics do give a rough estimate of prediction performance, there is no direct connection to what drivers and passengers inside vehicles experience. When using lane change prediction models in (partially) autonomous vehicles, controlling functions have to react to predictions, especially cut-ins to the ego-lane: The earlier and more accurate the prediction, the smoother the controller's reaction. When predictions are wrong, for example a lane change is wrongly predicted, this results in unnecessary breaking and acceleration behaviour, which significantly reduces drivers' comfort. Ideally, our metrics should be able to capture such situations, and help us quantify the behaviour of our models, such as "on average the vehicle drives 100 kilometres without false positive predictions". Therefore, we employ metrics specifically tailored for predicting lane changes. These are:

- ***Delay***: *Delay* measures the earliness of prediction for lane change events, i.e. the number of seconds passed between the ground truth label and first correct corresponding prediction of a manoeuvre. A perfect algorithm would exhibit a *Delay* of 0. When using our automatic labelling scheme, a *Delay* of 3s is maximal.

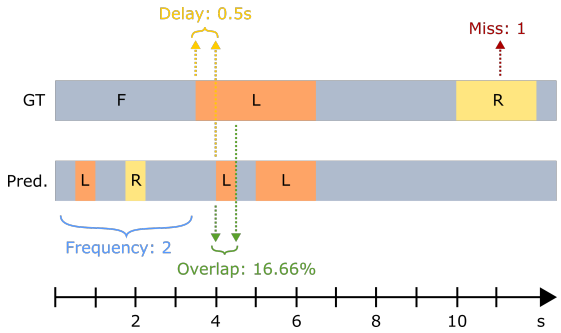


Figure 3.7: Depiction of a sample sequence to visualize the proposed metrics *Delay*, *Overlap*, *Frequency* and *Miss*. The sample sequence spans over 12s, ground truth labels are depicted on top, model predictions in the bottom. The sequence begins with a *Follow* period, followed by a lane change to the *Left*, and ends with a lane change to the *Right*.

- ***Overlap***: Controllers often react (or have to) to the first lane change prediction due to safety reasons. Thus, this is of great importance, and the metric *Overlap* measures the percentage of overlap of the first lane change prediction and the ground truth. Ideally, *Overlap* is 1, indicating that the model's first prediction corresponds with the full manoeuvre, i.e. all frames are predicted correctly.
- ***Frequency***: *Frequency* indicates the absolute number of times an event is predicted per ground truth event. For *Follow* events, this is also known as the false positive rate (FPR). A high *Frequency* indicates a spiky and discontinuous prediction, and, as stated before, this significantly reduces driver's comfort, among others because of provoking stop-and-go behaviour. A *Frequency* of 1 is optimal.
- ***Miss***: *Miss* describes the percentage of lane changes completely missed, i.e. no frame is correctly classified. A large number of misses obviously also has a high negative influence on the drivers' comfort, as they potentially have to intervene in critical situations.

Figure 3.7 visualizes these metrics. Further, we adapt the well-known metrics ***Precision*** and ***Recall*** to the event-wise continuous interpretation. Figure 3.8 explains this principle: For both metrics, we split all tracks (which are all frames of a specific target vehicle) into homogeneous chunks. For *Precision*, we split w.r.t. the model's prediction, for *Recall* w.r.t. the ground truth label. Then, we calculate the fraction of correctly predicted frames for each chunk and average (image from [2].

3.4.3 Results

Model *LSTM* consists of a single LSTM cell with size 128. In *E-LSTM* a single layer per cell is used, as well, the hidden sizes of $LSTM_Z$, $LSTM_E$ and $LSTM_M$ (the

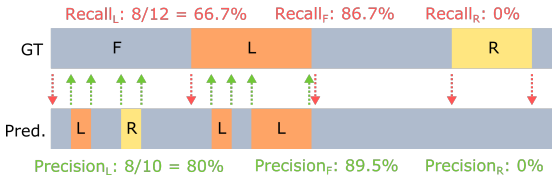


Figure 3.8: Depiction of the calculation of *Precision* and *Recall* per manoeuvre. The same sample scene as in Figure 3.7 is used. Calculation of precision is "bottom-up": The fraction of correctly predicted frames per homogeneous prediction chunk is calculated and averaged. Conversely, the calculation of *Recall* is "top-down": For this, the fraction of correctly predicted frames per ground truth label is averaged (image from [2]).

cells responsible for processing the input feature groups G^Z, G^E and G^M) are 128, 128 and 32, respectively. Layer **o** has size 32. In the attention scheme, a temporal window of 10 frames (1s) is used. Both the layers **E** and **J** live in a 16-dimensional space. We further employ L2 regularization with weighting factor 0.0000001, and perturb samples during training by adding independent Gaussian noise on feature inputs (ranging from -5% to +5% of original values).

Baseline Methods We implement several baseline methods to compare our algorithms against.

- Random Forest: We apply Schlechtriemen's [44] Random Forests, just tuning hyperparameters ourselves and extending them for a temporal understanding. In the simple form, Random Forests consisting of 10 decision trees with a maximal depth of 10 examine each frame separately (using features G_t^Z, G_t^E, G_t^M) for obtaining a prediction (RF). To allow a better understanding of temporal dependencies, we introduce RF-L, which only differs from RF by its input features, which now are a concatenation of 10 equidistant frames spanning over 2 seconds.

- Naive Bayes (NB): Similar to [47], we fit a Naive Bayesian approach for prediction, considering input features m, v_{lat} and PV_{dt}.

- Structural RNN (S-RNN): We implement an S-RNN according to [49]. Three LSTM cells process target's driving lane as well as left and right adjacent lanes. Each cell is given input features Q of three vehicles, namely target, as well as its two neighbours on that lane (PFV_L / PLV_L - PV / RV - PFV_R / PLV_R). Q contains absolute vehicle coordinates, lateral and longitudinal velocity, heading angle and number of lanes to the left and right. The outputs of the three LSTM cells are processed by another cell, which eventually outputs the prediction.

To evaluate and compare different algorithms, we use the metrics introduced in Section 3.4.2: *Delay* (De), *Overlap* (Ov) and *Miss* are only evaluated for lane change

manoeuvres, and usually averaged over lane changes to the left and right. *Frequency* (Fr) is only reported for *Follow* events, due to the huge impact of false positive predictions on felt driving comfort. *Precision* (Pr) and *Recall* (Re) are evaluated for *Follow* events as well as lane changes, again averaged over directions. To better compare our results with previous works, we additionally report standard *Accuracy* (Acc) for all manoeuvre classes. We then sort algorithms, one metric at a time, assigning points based on the resulting ranking, and sum all points, resulting in *Rank*. The algorithm with the lowest points performs best when averaging over all metrics.

3.4.3.1 Comparison of Models on Full Datasets

In this section we compare all algorithms on full datasets. Tables 3.1, 3.2 and 3.3 show results on the NGSIM, fleet and highD dataset, respectively.

Differences between models are smallest on the NGSIM dataset. Random Forests, the Naive Bayesian approach and our LSTM models perform comparably. Experi-

Table 3.1: Results of all models on the NGSIM dataset.

Algorithm	Acc LC	Acc F	Miss LC	De LC	Ov LC	Pr LC	Re LC	Fr F	Pr F	Re F	Rank
NB	0.697	0.886	**0.003**	0.705	**0.618**	-	-	7.271	-	-	43
RF	**0.72**	0.935	**0.003**	0.597	0.584	0.037	**0.724**	6.843	**0.973**	0.907	36
RF-L	0.71	0.935	0.006	0.66	0.584	0.034	0.712	6.017	0.972	0.898	45
S-RNN	0.515	0.905	0.167	0.789	0.461	-	-	6.686	-	-	61
LSTM	0.671	0.973	0.004	0.777	0.584	0.056	0.678	3.442	0.961	0.918	47
E-LSTM	0.689	**0.975**	0.006	0.731	**0.596**	0.074	0.696	**3.38**	0.958	0.93	38
LSTM-A	0.687	0.974	0.005	0.7	0.568	0.068	0.696	4.075	0.96	0.932	44
E-LSTM-A	0.71	**0.975**	0.005	0.653	0.594	**0.078**	0.718	4.064	0.962	**0.935**	**22**

Table 3.2: Results of all models on the fleet data.

Algorithm	Acc LC	Acc F	Miss LC	De LC	Ov LC	Pr LC	Re LC	Fr F	Pr F	Re F	Rank
NB	0.823	0.773	0.044	0.155	0.691	-	-	3.461	-	-	54
RF	0.81	0.905	0.067	0.151	0.602	0.097	0.761	2.984	0.943	0.84	60
RF-L	0.836	**0.946**	0.101	0.184	0.631	**0.149**	0.76	2.038	0.932	**0.891**	50
S-RNN	0.860	0.834	0.052	0.223	0.777	-	-	2.28	-	-	52
LSTM	0.897	0.894	0.038	0.19	0.808	0.112	0.84	1.722	0.967	0.803	38
E-LSTM	**0.922**	0.882	**0.028**	**0.13**	**0.826**	0.078	**0.872**	1.779	**0.976**	0.777	**31**
LSTM-A	0.878	0.937	0.04	0.185	0.772	0.144	0.827	**1.619**	0.961	0.864	32
E-LSTM-A	0.887	0.932	0.042	0.18	0.78	0.136	0.83	1.653	0.969	0.852	31

Table 3.3: Results of all models on the highD dataset.

Algorithm	Acc LC	Acc F	Miss LC	De LC	Ov LC	Pr LC	Re LC	Fr F	Pr F	Re F	Rank
RF	0.962	0.989	**0.0**	0.088	0.926	0.242	0.966	1.093	0.949	0.984	45
RF-L	0.951	0.99	0.007	0.118	0.909	0.211	0.94	1.107	0.941	0.984	56
LSTM	0.988	**0.995**	**0.0**	0.027	0.988	0.28	0.99	1.006	0.956	0.98	28
E-LSTM	0.986	**0.995**	**0.0**	0.036	0.987	**0.308**	0.988	**0.994**	0.954	0.973	30
LSTM-A	0.99	0.994	**0.0**	0.024	0.989	0.294	0.991	1.023	0.959	0.989	23
E-LSTM-A	**0.992**	**0.995**	**0.0**	**0.019**	**0.99**	0.288	**0.992**	1.02	**0.96**	**0.99**	**14**

ments with *S-RNN* did not yield fruitful results. Overall though, *E-LSTM-A* performs best. In general we observe, that splitting input features into groups and using different LSTM cells to process these improves performance over the base variant, same holds for the introduction of attention. *Frequency* is much lower (around 50%) for our proposed LSTM variants than for all other models. This confirms our assumption, that LSTMs are better suited for time-series data, and usually yield more stable, time-consistent predictions. LSTMs and RNNs in general can be seen as low-pass filters, filtering out noise and fluctuations with high frequency in the predictions. In particular, consider the example of drivers wiggling around their driving lanes. While frame-based models, like Random Forests, might result in alternating lane change predictions, LSTMs can learn such behaviour and smoothly predict *Follow*. In fact, *RF-L* worsens the performance of *RF*, indicating that, although Random Forests have proven to be universal and well-working classification models, in the understanding of temporal sequences they are inferior to recurrent models. Thus, we argue that RNNs should be considered state-of-the-art for manoeuvre and especially lane change prediction problems.

Results on our fleet data are similar, except now all our LSTM variants perform better than other baseline methods.

This trend continues on the highD dataset. The gap between recurrent models and frame-based approaches grows, and *E-LSTM-A* now outperforms all other methods. This indicates, that such a complex model strictly is more powerful and reduces prediction bias, and that, given enough data, this method performs better than all others.

3.4.3.2 Scenario-Based Evaluation

In this section we discuss more fine-grained evaluations of examined models. For this, we focus on our proposed LSTM models and pick Random Forests as best-performing baselines. As hinted before, many lane changes are "easy" to predict, and many simple models might do well by just focusing on features like distances to lane boundaries and lateral velocities. These models, however, would fail in more complex situations, e.g. when lateral movement to a lane boundary is detected but the lane is blocked by another vehicle, which makes a lane change rather unlikely. As such situations only constitute a minority of situations of recorded datasets, concern is, that such results are neglected by standard evaluations. Therefore, we introduce a number of standardized scenarios, spanning different corner-cases motivated from important real-world examples, and analyze our models separately for each. The introduced scenarios are:

- ***Left***: All frames labelled with L.
- ***Follow***: All frames labelled with F.
- ***Right***: All frames labelled with R.

- ***Left-PV***: Same as *Left*, but additionally a close preceding vehicle (PV) is present. In particular, a frame i is contained in this scenario, if the average value of $dt_{\texttt{PV}}$ over a 3s-window before i is less than 1.5s, and this distance reduces over time, i.e. the target vehicle is approaching PV.
- ***Right-PV***: Similar to *Left-PV*, but containing frames labelled with R.
- ***Left-Blocked***: Same as *Follow*, but putting additional constraints on lateral movements: In particular, this scenario contains *Follow* situations, which easily might be misclassified as lane changes. A strong lateral velocity needs to be observed (greater or equal to 0.2 m/s towards the lane boundary on average over a 0.5s window before a frame), and the average distance to the left lane boundary needs to be less or equal to 40% of the lane width. Additionally, the left lane needs to be blocked, for which we require the average values of $dt_{\texttt{PLV}_L}$ or $dt_{\texttt{PFV}_L}$ over the same window to be less or equal to 0.3s
- ***Right-Blocked***: Same as *Left-Blocked*, but mirrored.
- ***L1***: Frames in the scenario fulfill the lateral movement criterion from *Left-Blocked*, but only contain vehicles on the left-most lane, where no lane change to the left is possible.

As evaluating all models in all situations generates numerous tables, we only highlight specific results in detail and summarize the rest in compressed form. Further, we adapt resulting tables to only contain meaningful metrics: As all scenarios only contain frames with a certain label, we only include metrics concerning this label, and further remove *Precision*, as this always evaluates to 1.

To begin, let us analyze the scenarios *Left*, *Follow* and *Right* on the highD dataset, which are shown in Tables 3.4 to 3.6. This again proves the superiority of RNN models when enough data is present, especially for *Follow* scenarios, as here the gap is largest.

One assumption, why attention models produce good results, is that they are particularly suited to understand complex situations, and could for example assign

Table 3.4: Results of all algorithms in scenario *Left* on the highD dataset. Number of frames in this scenario: 11579.

Algorithm	Acc L	Miss L	De L	Ov L	Re L	Rank
RF	0.964	**0.0**	0.081	0.92	0.968	22
RF-L	0.953	0.001	0.112	0.918	0.951	31
LSTM	0.995	**0.0**	0.012	0.994	**0.996**	9
LSTM-E	**0.996**	**0.0**	**0.011**	**0.996**	**0.996**	**6**
A-LSTM	0.991	**0.0**	0.021	0.99	0.992	17
A-LSTM-E	0.994	**0.0**	0.016	0.99	0.994	14

high weights to neighbouring vehicles when these block a lane, but lateral movements towards that lane is observed. Thus, let us examine *Left-Blocked* on the NGSIM dataset, as shown in Table 3.7. However, this does not fully support our claim. Although results are close, attention does not provide a significant increase in performance. Note though, that the amount of data for each scenario is limited, and thus likely exhibits some variance in the results.

Table 3.5: Results of all algorithms in scenario *Follow* on the highD dataset. Number of frames in this scenario: 1067662.

Algorithm	Acc F	Fr F	Re F	Rank
RF	0.989	1.093	0.984	15
RF-L	0.99	1.107	0.984	15
LSTM	**0.995**	1.006	0.98	9
LSTM-E	**0.995**	**0.994**	0.973	9
A-LSTM	0.994	1.023	0.989	11
A-LSTM-E	**0.995**	1.02	**0.99**	**6**

Table 3.6: Results of all algorithms in scenario *Right* on the highD dataset. Number of frames in this scenario: 18222.

Algorithm	Acc F	Miss R	De R	Ov R	Re R	Rank
RF	0.96	**0.0**	0.095	0.932	0.964	22
RF-L	0.949	0.013	0.123	0.9	0.929	31
LSTM	0.982	0.001	0.041	0.982	0.984	18
LSTM-E	0.976	**0.0**	0.06	0.978	0.979	18
A-LSTM	0.989	**0.0**	0.027	0.988	0.99	10
A-LSTM-E	**0.99**	**0.0**	**0.021**	**0.989**	**0.991**	**6**

Table 3.7: Results of all algorithms in scenario *Left-Blocked* on the NGSIM dataset. Number of frames in this scenario: 18372.

Algorithm	Acc F	Fr F	Pr F	Re F	Rank
RF	0.85	2.068	**0.999**	0.855	16
RF-L	0.826	2.147	**0.999**	0.835	19
LSTM	0.924	1.484	0.998	0.927	14
E-LSTM	0.951	**1.314**	0.998	0.954	8
LSTM-A	0.944	1.511	0.998	0.947	13
E-LSTM-A	**0.954**	1.362	0.998	**0.956**	**7**

Tables 3.8 and 3.9 show summarized results on the NGSIM and highD dataset. This allows a detailed and complete understanding of examined algorithms. We observe, that for the NGSIM dataset Random Forests are comparable to LSTMs, and in lane change situations often perform better. Still, in *Follow* scenarios we again observe a superiority of recurrent models, and would like to emphasize the importance of the metric *Frequency*: False lane change predictions have a significant negative impact on felt driving comfort, as these in many situations lead to abrupt and unsmooth behaviour. It can be argued, that this metric is more important than other ones, e.g. in comparison to *Miss* and *Delay*: *Miss* anyways often is negligibly low, and small *Delay* differences might not be observable to the driver.

Table 3.8: Summarized scenario results for the NGSIM dataset. For each model, *Rank* is listed for each scenario, and then summed over all scenarios in row *Total*.

Scenario	RF	RF-L	LSTM	E-LSTM	LSTM-A	E-LSTM-A
Left	11	12	18	30	23	15
Follow	18	16	11	**6**	9	**7**
Right	**6**	16	26	19	23	17
Left-PV	13	25	13	19	17	8
Right-PV	12	24	25	15	17	8
Left-Blocked	16	19	14	8	13	**7**
Right-Blocked	16	19	**7**	8	12	15
L1	18	**8**	15	10	**8**	8
Total	110	139	129	115	122	**85**

Table 3.9: Summarized scenario results for the highD dataset. For each model, *Rank* is listed for each scenario, and then summed over all scenarios in row *Total*.

Scenario	RF	RF-L	LSTM	E-LSTM	LSTM-A	E-LSTM-A
Left	22	31	9	**6**	17	14
Follow	15	**15**	9	9	11	**6**
Right	22	31	18	18	10	**6**
Left-PV	23	25	18	8	14	21
Right-PV	22	26	18	14	**6**	**6**
Left-Blocked	**9**	19	**8**	14	11	16
Right-Blocked	20	20	9	9	11	11
L1	17	22	12	10	9	8
Total	150	189	101	88	89	**88**

When more data is available, like for the highD dataset, RNN models overall improve, now outperforming Random Forests in nearly all categories. Again *E-LSTM-A* performs best overall.

3.4.3.3 Determining Feature Importances

Especially for safety-critical applications like autonomous driving, treating neural networks as black-box models is problematic: It is not clear, how one can verify their behaviour and guarantee provable performance bounds. One plausible way of doing so is using statistical tests, i.e. calculating or estimating accident rates of a system, and then comparing this to human drivers or other used systems. Still, no guarantees exist about what neural networks will do in certain, especially unexpected, situations. Although not solving this problem, understandable models, which are able to explain their decisions, still are a step in the right direction: When models explain their decisions, and such reasoning seems logical, one might trust such systems easier, particularly in critical situations. When e.g. using a visual attention mechanism in traffic light detection algorithms, we expect it to focus on actual traffic lights and their colours. When, however, attention puts much emphasis on, e.g., lower parts of the supporting pole of traffic lights or other road elements, this could indicate, that our model has overfit certain situations from the training set (maybe all traffic lights at that intersection showed a green signal). Analyzing which features are most relevant for certain decisions, or using attention mechanisms, is one possibility of doing so, thus in this section we will examine such methods.

Schlechtriemen et al. [47] analyze feature importances for the problem of predicting lane changes, using Naive Bayes and Hidden Markov models. They conclude, that the features distance to the lane's center line, lateral velocity and relative distance to the preceding vehicle are most relevant. Here we examine feature importances of RNNs for this problem, evaluating these among others using our attention mechanism. We further detail this analysis by using the scenarios introduced in the previous section, finding valuable results for different driving scenarios. The methods used for evaluating feature importances are:

- *Train-k*: We train model *LSTM* (due to simplicity) while leaving out one feature in each training. By doing so for all features and evaluating the results, we can rank features w.r.t. their importance. The ranking is obtained by using column *Rank* as before, here we additionally normalize values.

- *d-LSTM*: Similar to Saliency Maps for CNNs [54], we form the derivative of the prediction by each feature X_i, i.e. $\frac{d\mathbf{y}_t}{dX_i^t}$. This indicates the sensitivity of the prediction to each feature, thus indicating how much each feature contributes to the result.

- *LSTM-A / E-LSTM-A*: Our proposed attention schemes directly output feature importances, namely the values γ_t^i. These indicate the weights used when

accumulating the feature embeddings, more important features are assigned a higher weight.

In the following, feature names are identical as introduced in Section 3.3.1, except features dt_X are abbreviated by X.

We begin by analyzing feature importances over the full highD dataset, as shown in Table 3.10. *Train-k* deems very meaningful features as most important overall, the top-3 ranked features are: distance to the lane's center line, lateral acceleration and temporal distance to the preceding vehicle. This is nearly identical to [47], and a first assuring, but not surprising result: It proves, that these features indeed are the most relevant ones for predicting lane changes, and that such results do not depend on the model being used. Results of the derivative and attention methods are slightly different, but overall similar. Attention particularly weighs distances to adjacent vehicles on neighbouring lanes highly, which interestingly models behaviour of a human driver, regularly checking safety distances. More detailed evaluations are needed though, thus we continue with a scenario-based assessment. Another downside of such general evaluation is that, again, *Follow* frames dominate the results.

For this, we examine two sample scenarios on the NGSIM dataset. Table 3.11 shows results for scenario *L1*. Here, *Train-k* deems m and l_{ID} as most important, which makes a lot of sense, as the lane ID should have a great positive influence on a *Follow* prediction. *LSTM-A* and *E-LSTM-A* also indicate m and l_{ID} as most relevant features, showing the applicability of our attention mechanism. *d-LSTM* however only ranks l_{ID} in the lower half of all features. This as well as other experiments (consider for example the small changes w.r.t. Table 3.12) indicate,

Table 3.10: Feature importances for the highD dataset.

Train-k	d-LSTM	LSTM-A	E-LSTM-A
m (0.115)	a_{lat} (0.158)	m (0.265)	PFV_L (0.232)
a_{lat} (0.114)	m (0.153)	PLV_R (0.119)	PLV_R (0.161)
PV (0.112)	v_{lat} (0.098)	PLV_L (0.094)	v_{long} (0.138)
l_{ID} (0.109)	l_{ID} (0.096)	PV (0.057)	RV (0.114)
v_{long} (0.096)	RV (0.085)	RV (0.057)	PLV_L (0.088)
v_{lat} (0.072)	PFV_R (0.074)	h (0.047)	PFV_R (0.064)
PFV_L (0.069)	h (0.068)	v_{long} (0.042)	d_on (0.063)
PLV_R (0.062)	PFV_L (0.063)	PFV_R (0.037)	l_{ID} (0.059)
PLV_L (0.062)	v_{long} (0.056)	PFV_L (0.027)	PV (0.027)
h (0.055)	PV (0.050)	d_on (0.026)	h (0.015)
PFV_R (0.046)	PLV_L (0.041)	v_{lat} (0.022)	m (0.014)
RV (0.046)	PLV_R (0.033)	a_{lat} (0.015)	a_{lat} (0.010)
d_{on} (0.041)	d_{on} (0.018)	l_{ID} (0.009)	v_{lat} (0.006)

that forming derivatives through a complex LSTM cell - as opposed to through a CNN - is hard, among others due to the complex internal structure, as well as the temporal component of the problem. We note, that *Train-k* and attention are useful methods for explaining made decisions, however, only attention can be applied online, particularly while driving on the road. Thus, our proposed attention mechanism is a valuable contribution in this field.

Table 3.12 shows results of a different scenario, namely *Right-Blocked*. In this scenario, *Train-k* deems PFV_R as most important feature. This seems very plausible, as the existence of such vehicle strongly suggests against predicting a lane change to the right. However, for attention, results are less conclusive: Although not "wrong" or totally counterintuitive - e.g. h and a_{lat} are useful features - these features differ from ones picked as most relevant by humans. This shows, that attention does not yield "satisfying" results in all situations, and we would like to improve on that

Table 3.11: Feature importances for the NGSIM dataset in scenario *L1* (only showing top-8 features). Number of frames in this scenario: 107370.

Train-k	d-LSTM	LSTM-A	E-LSTM-A
m (0.131)	v_{long} (0.129)	l_{ID} (0.220)	a_{lat} (0.233)
l_{ID} (0.116)	m (0.105)	PLV_R (0.165)	l_{ID} (0.193)
RV (0.103)	a_{lat} (0.096)	v_{long} (0.135)	PFV_L (0.186)
h (0.101)	v_{lat} (0.093)	m (0.095)	d_{off} (0.143)
d_{off} (0.090)	$PFVL$ (0.082)	PFV_L (0.093)	h (0.057)
d_{on} (0.088)	PV (0.070)	d_{on} (0.093)	m (0.046)
PV (0.078)	RV (0.063)	PV (0.071)	v_{long} (0.045)
PFV_R (0.068)	l_{ID} (0.062)	h (0.057)	RV (0.040)

Table 3.12: Feature Importances for the NGSIM dataset in scenario *Right-Blocked* (only showing top-8 features). Number of frames in this scenario: 202560.

Train-k	d-LSTM	LSTM-A	E-LSTM-A
PFV_R (0.120)	v_{long} (0.129)	PLV_L (0.170)	a_{lat} (0.171)
l_{ID} (0.111)	m (0.105)	h (0.169)	RV (0.168)
d_{on} (0.102)	a_{lat} (0.095)	v_{long} (0.160)	h (0.161)
h (0.100)	v_{lat} (0.092)	PV (0.085)	PLV_L (0.131)
PLV_L (0.095)	PFV_L (0.079)	d_{on} (0.083)	d_{off} (0.084)
v_{lat} (0.091)	PV (0.071)	PLV_R (0.071)	v_{long} (0.080)
d_{off} (0.063)	l_{ID} (0.063)	PFV_L (0.061)	m (0.049)
RV (0.061)	RV (0.063)	l_{ID} (0.056)	l_{ID} (0.044)

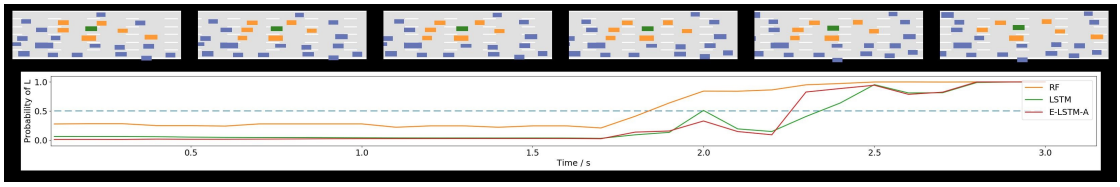


Figure 3.9: Depiction of a lane change to the left on the NGSIM dataset. The target vehicle is drawn in green, neighbouring vehicles in orange and other vehicles in blue. The direction of travel is to the right. Below, predicted probabilities of a lane change from the models *RF*, *LSTM* and *E-LSTM-A* are plotted over time (image from [2]).

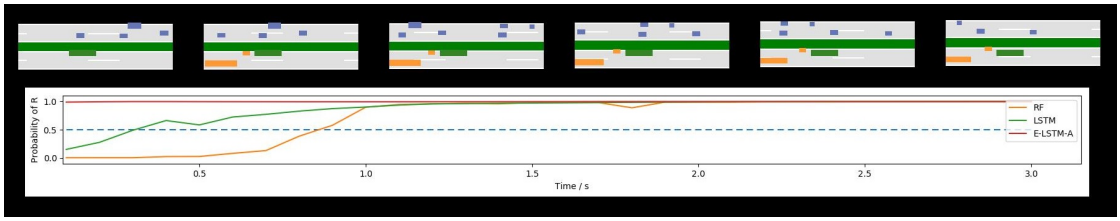


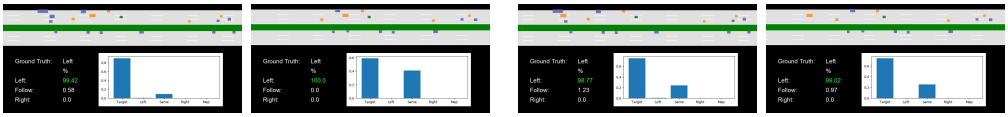
Figure 3.10: Depiction of a lane change to the right on the highD dataset. The target vehicle is drawn in green, neighbouring vehicles in orange, other vehicles in blue. Note that on the highD dataset, vehicles are going in both directions, respective lanes are separated by a grass strip in the middle: Vehicles in the upper half drive towards the left, vehicles in the lower one towards the right. Below, the predicted probabilities of a lane change to the right, as predicted by the models *RF*, *LSTM* and *E-LSTM-A*, are plotted over time (image from [2]).

in the future. Note however, that there is no "right" or wrong: Our experiments factually return attention weights. Clever models might learn correlations not visible to humans, or noise from training data (this way, giving developers a tool for early detection). Further, in the next section we show that attention models indeed react intuitively to such situations in a qualitative way.

3.4.3.4 Qualitative Results

In this section we show qualitative results of different prediction algorithms. Figures 3.9 and 3.10 show sample lane changes on the NGSIM and highD dataset. Both figures contain 6 scene visualizations of the lane change at equidistant timesteps, ranging from 3s before the manoeuvre until the point of crossing lane boundaries. Below, the predicted probabilities of a lane change to the respective side for the models *RF*, *LSTM* and *E-LSTM-A* are depicted.

Figure 3.9 shows a lane change to the left on the NGSIM dataset. In accordance with the tabular results from the previous section, *RF* predicts first. Note the close distances to neighbouring vehicles though, which could be accounted for by the more complex LSTM models.



(a) Target vehicle rapidly approaches PV, and (b) In comparison to scene a, in absence of PV a human could anticipate a forthcoming lane the prediction of a lane change is later and less change. So does our model *E-LSTM-A*, and confident, additionally the weight of *Same* is accordingly assigns a high weight to category less.
Same.

Figure 3.11: Two scenes of a lane change to the left on the highD dataset are shown. The target vehicle is drawn in green, neighbouring vehicles in orange, other vehicles in blue. Note that on the highD dataset, vehicles are going in both directions, respective lanes are separated by a grass strip in middle: Vehicles in the upper half drive towards the left, vehicles in the lower one towards the right. Scene a and b stem from the same lane change, but b) is an artificially modified version of a): We increased the distance to PV. For both scenes, the left image shows a frame early in the lane change, while the right shows a frame close before the point of crossing lane boundaries (image from [2]).

Figure 3.10 depicts a lane change to the right on the highD dataset. In this, the trailing vehicle is approaching target rapidly, which, by German law, is required to give way to the right. This interaction seems to be understood well by *E-LSTM-A*, which predicts a lane change even before any lateral movement can be observed. *LSTM* follows, while *RF* predicts last, nearly one second later than *E-LSTM-A*.

Figures 3.11 and 3.12 depict outputs of our proposed attention scheme. Next to the scene visualization, the predicted manoeuvre probabilities of *E-LSTM-A* and the calculated attention weights γ are shown. For a better overview, we group features into the feature groups G_t^Z, G_t^E and G_t^M, but split G_t^E into the three lanes. Thus, we obtain the groups *Target* (G_t^Z), *Map* (G_t^M) and *Left* / *Same* / *Right* (features of G_t^E regarding left / current / right driving lane). Figure 3.11a depicts a lane change to the left on the highD dataset. The scene is represented with by two images, one early in the lane change (left) and one shortly before crossing lane boundaries (right). *Target* is given high weight, which is plausible, as the corresponding features naturally are good lane change indicators. We observe, that *Same* is given more weight the more target approaches PV. Figure 3.11b shows the same scene, except we artificially increased PV_{dt}. Now, a lane change prediction occurs later and less confidently, and additionally the weight of *Same* is less. This coincides with our human understanding of the problem.

The left image of Figure 3.12 visualizes a frame of a lane change to the right on the NGSIM dataset. The right image visualizes the same frame in a similar situation, except we artificially reduced distances to preceding and following vehicles on the right lane, effectively blocking it. Due to the exhibited strong lateral movement to the right, this constitutes a sample of scenario *Right-Blocked*. We mentioned this particular scenario in Section 3.4.3.3, as an example where attention does not yield

Figure 3.12: Two frames showing a lane change to the right on the NGSIM dataset are shown, both recorded at similar timesteps and from identical scenes. Again though, in the scene corresponding to the right image distances to neighbouring vehicles on the target lane were reduced artificially. The target vehicle is drawn in green, neighbouring vehicles in orange and other vehicles in blue. The direction of travel is to the right. *E-LSTM-A* predicts a lane change to the right when corresponding gaps are large enough, and *Follow* when these are too small. In unison with that, *Right* is given a higher weight, indicating that the model correctly considers these features in its decision-making process (image from [2]).

expected results. However, using this visualization we see, that although *Right* is not ranked most important feature, still it reacts appropriately to the change in scenes: When the right lane is blocked, *Right* is given a much higher weight, additionally, our model predicts *Follow* instead of lane change.

3.5 Conclusion

We proposed LSTM models to tackle the problem of predicting lane changes, which perform better than existing methods. We extended these via attention mechanisms, further improving performance. In particular, we observed larger performance gaps between our recurrent models and classical prediction methods, such as Random Forests, and between attention models and their corresponding baseline LSTMs, when more data is available. This shows the power and capacity of such models, and that they should be regarded state-of-the art.

Additionally, we introduced metrics to better assess felt driving comfort. These allow a better understanding of what drivers feel inside autonomous vehicles, and how changes in prediction performance affect actual driving behaviour. Furthermore, we proposed much more fine-grained evaluation settings, introducing different scenarios motivated by common driving situations occurring in the wild.

Also, we ranked input features used for predicting lane changes by importance, using different methods, such as differentiating through the network and attention mechanisms. Such results are useful for understanding internal functioning of neural networks, which are traditionally viewed as black-box models. Attention proved to

be a valuable tool for reaching these insights, although not yielding expected results in all cases. Improving on this and obtaining even better explainable models is a future goal of ours.

Recent works in different fields replace recurrent neural networks by feed-forward networks and attention mechanisms [17, 18], which mitigates some downsides of RNNs: Training and inference is costly and slow, due to the sequential nature of calculation steps, which cannot be parallelized. Different techniques, such as (dilated) 1D-convolutions, prove to be capable of processing time-series data, and with increasing step size are also able to capture long-term dependencies, all that while allowing parallel computations. We note that parallelism during run-time actually is irrelevant for problems in the field of autonomous driving, as data anyways is measured and processed sequentially. Still, experimenting with successful feed-forward architectures, especially combining 1D-convolutions and attention mechanisms, seems like a worthy field of study, and we would like to test the effectiveness of such for predicting lane changes. Additionally, predicting full scenes in one step, i.e. predicting all future driving manoeuvres of all agents jointly, as well as combining manoeuvre and trajectory prediction, could and should be addressed in the future.

4 Considering Ambiguity

In the previous section we saw a first successful prediction model, which predicts the most likely manoeuvre in every timestep. In many situations though, multiple outcomes are likely, especially in the field of autonomous driving: It is hard to predict future driving manoeuvres of irrationally behaving drivers; at intersections, multiple future trajectories are possible and likely. Scope of this section is introducing methods for dealing with such ambiguity and uncertainty.

Many works in this field focus on static, non-dynamic problems, such as Rupprecht et al. [8]. Here, we extend their proposed Multiple Hypothesis Prediction Framework (MHP) to recurrent models, showing its extension to different state-of-the-art recurrent architecture types. We address the previously introduced problem of predicting lane changes, but also dive into the field of prediction more generally, discussing trajectory forecasting models and sequence generation tasks.

In particular, our contributions are:

- We extend the MHP framework to recurrent models, showing its extension for three successful recurrent architectures, namely sequence-to-sequence prediction, encoder-decoder models and sequence generation tasks.
- We analyze applications to several real-word problems in detail, such as manoeuvre prediction, trajectory forecasting and generation as well as text generation.
- We introduce novel metrics to better capture and describe ambiguity and uncertainty in problems. Core idea is the re-labelling of data points, to account for multiple possible labels.

4.1 Introduction

Ambiguity and uncertainty are present in many tasks of static and sequential nature: In image classification, often we might not be sure, what kind of specific breed of dog

is depicted in a picture, but we can definitely discard the possibility of the object being a human. Sequential problems inherently exhibit multiple possible futures: Vehicles approaching intersections might choose any of the possible path and turns, in text generation, used e.g. for auto-completion on mobile phones, multiple words are equally likely.

To model such relations, a multitude of techniques are available: Techniques from *Variational Bayesian Inference* (VI) and generative models both approximate full posterior and joint probability distributions. Commonly applied methods are *Normalizing Flows* [57, 18, 58], *Variational Autoencoders* and *Generative Adversarial Networks*. Multimodal discriminative methods exist, as well, often an ensemble of different expert predictors is employed, like in *Multiple Choice Learning*. Rupprecht et al. introduce a similar framework for predicting multiple hypotheses, which actually offers mathematical insights into why MCL models work, and uses a Voronoi tesselation of the label space [8].

Here, we put special emphasis on recurrent models, as autonomous driving essentially is a temporal problem, and extend Rupprecht's model to sequential architectures. In particular, we examine three recurrent model architectures, which we consider state-of-the art for many problems: We begin with a sequence-to-sequence prediction model, which processes inputs at every timestep and generates a prediction for each. One example application is the previously introduced prediction of driving manoeuvres, which we consider. Next, we examine encoder-decoder models. These excel in many tasks, such as machine translation and trajectory prediction. We focus on trajectory prediction. Lastly, we consider sequence generation models. Using these, we generate trajectories as well as text. For all these different architectures, we showcase how to extend them to MHP models, allowing the prediction of multiple hypotheses.

Additionally, we introduce a novel metric, specifically designed for multimodal problems. Many previous works consider some form of oracle metric, which only considers best predictions during evaluation. Although theoretically sound, this has drawbacks, as it could be easily fooled by guessing solutions and always prefers diverse predictions. Our metric consists of different steps, one of which is re-labelling data points, possibly assigning multiple labels, and then requiring predictions to match all of these.

4.2 Related Work

Over the last years, deep learning methods, especially CNNs, have emerged as powerful, general function approximators, outperforming traditional methods in many fields, and even rivalling or surpassing human performance [59, 60]. However, most works treat problems as learning a one-to-one mapping from input to output, yielding one result per input. For many problems, this is neither desired nor correct. In addition, it has been shown, that especially CNNs are often overconfident, valuing the most likely hypothesis overproportionally - diminishing their capabilities

of creating multimodal outputs. Further, this provides one reason why they are susceptible to adversarial attacks.

Rupprecht et al. introduce a general framework for predicting multiple hypotheses, dubbed Multiple Hypothesis Prediction framework, which is based on a Voronoi tessellation of the label space [8]. Since their focus is on feed-forward architectures and static problems, we here extend this framework to sequential problems and models. In this section we introduce related, relevant works. We begin with a brief overview over the full field, before listing specific works.

Variational Bayesian Inference (VI) Although more being a general technique and optimization method, we still would like to mention VI here, as it provides ways of generating probability distributions, from which we can sample to generate diverse outputs. VI methods in general can be applied for finding intractable integrals, which often occur in posterior distributions $p(z|x) = \frac{p(z,x)}{\int_z p(z,x)}$. While Monte Carlo sampling methods, consider e.g. Gibbs sampling, yield numerical approximations, VI methods calculate exact solutions of approximate distributions. Different approaches exist, such as minimizing the Kullback-Leibler divergence between an approximate distribution q and $p(z|x)$, and then using the *Evidence Lower Bound* (ELBO) to estimate the intractable integral. VAEs make use of a similar reformulation. Another technique are Normalizing Flows, which make use of the Change of Variable Theorem to iteratively approximate complex distributions [57, 18, 58]. However, our framework is more similar to common deep learning architectures and machine learning models (than to concepts from probability estimation), thus we focus more on these. One distinguishes generative and discriminative models.

Generative Models Generative models aim at estimating full joint probability distributions $p(x, y)$, s.t. one can can sample from these to generate diverse outputs. (C)VAEs learn data-conditional latent distributions, employing an encoder and decoder to project into the latent space and reconstruct samples [61]. Via reformulating the ELBO one arrives at the loss term $E[\log p(x|z)] - \mathrm{KL}(q(z|x)||p(z))$. The first part describes the reconstruction loss of the decoder, prompting usable and accurate reconstructions. The second contains the KL divergence between our chosen approximation q of the true posterior, and a prior over z, which typically is a Normal distribution. Gregor et al. introduce sequential VAEs [62], Jain et al. apply a similar principle to generate questions from images [63]. GANs are another common and powerful generative model [14], yielding fascinating results in many areas, such as image generation [64]. A generator tries to generate data, starting from a random noise vector, while a discriminator tries to distinguish this from real data. The better both players get, the more realistic-looking the created data is. Usually, they are hard to train though, due to the involved min-max game between generator and discriminator. Further, for many problems data is concentrated on a lower dimensional manifold. As often output of the generator describes such a manifold as well, in the standard loss formulation this causes problems when these

do not intersect: It can be shown, that the original GAN formulation describes the Jensen-Shannon divergence between actual data distribution and generated samples, which is non-continuous and non-differentiable in these cases. Using the Wasserstein distance mitigates the latter problem [65].

Discriminative Models With discriminative models we instead seek to find the decision boundary $p(y|x)$. In this setting, MCL offers multiple predictions [21], by creating an ensemble of predictors and only backpropagating gradients to the best error predictors. Whereas early versions required costly retraining [20], Lee et al. improve on this so to enable usage of standard Stochastic Gradient Descent by backpropagating gradients only to the lowest error predictors for each example [21]. Consequently, their loss formulation is very similar to those of Rupprecht et al. and thus ours. Indeed, the MHP framework offers mathematical insights into why such a backpropagation scheme works. Rupprecht further generalize the framework to also allow regression problems, reduce parameters and allow information exchange between predictors by sharing weights, instead of using M separate models [8]. In their work, Lee et al. also examine sequential problems in the form of image caption generation, for which they use LSTMs [21]. MDNs replace a standard regression output with parameters of a Mixture-of-Gaussian distribution [19]. For M Gaussians, the neural network predicts M weighting coefficients, variances and means, thus modelling said distribution. This way, instead of only predicting the desired regression target, we arrive at a posterior distribution $p(y|x)$, which includes information about uncertainty and ambiguity. Rupprecht et al. show that MDNs can be hard to train in high-dimensional spaces due to numerical instabilities [8].

We now take a look at some specified works, handling ambiguity and uncertainty for concrete applications, grouped by area of application:

Image Classification Image classification is a big motivator for multimodal prediction, as often many objects are present in images and multiple labels possible [66]. Wang et al. e.g. combine CNNs with RNNs for outputting multiple labels [67]. Rupprecht et al. also consider image classification problems [8].

Sequential Problems Ambiguity in sequential problems has been addressed less frequently in research. Most works propose problem-specific solutions, relying on GANs, VAEs, RL or other kinds of sequential modelling techniques [68] for producing ambiguous outputs. Lee et al. combine RNNs and GANs for predicting VRU trajectories [41], Gupta et al. GANs with a special diversity loss for the same problem [42]. Shi et al. employ Inverse RL to avoid mode collapse when generating text [69], while Shao et al. apply stochastic beam search with sequence-to-sequence models [70]. Bazzani et al. introduce Recurrent MDNs for obtaining a saliency map of visual attention in videos [71].

Ambiguous Labelling Several works have address the topic of dealing with multiple labels. Kalyan et al. incorporate multiple labels for dealing with sparse annotations [72]. They jointly learn model and an embedding function which defines similarity, i.e. the concept of neighbourhood in the input space. This allows modifying the loss function in order to take labels of neighbouring data points into account. On the one hand, this resembles our minimum formulation, although Kalyan et al. explicitly calculate distances in the label space and re-label data points, whereas our approach implicitly learns the underlying ambiguity, thereby addressing ambiguous predictions rather than ambiguous annotations. On the other hand, this mirrors motivation for our newly proposed metric, clustering points in label space and reassigning labels. While their mapping to new labels is learned in combination with the model, ours is fixed and deterministic, intended to be used as a problem- and model-independent metric. Likewise, other papers dealing with multiple labels within model formulations [73, 74] do not propose any general-purpose metric to handle ambiguity.

Rhinehart et al. employ two entropy terms in their loss formulation for predicting trajectories, which encourage predictions to be precise while still simultaneously encouraging diversity [75]. Again, this is very similar in motivation, but also is no metric, and we introduce ours in a more general way: For example, we also discuss classification problems and show how arbitrary metrics, such as Precision and Recall, can be extended to multimodal settings.

4.3 Multiple Hypothesis Prediction Framework

In this section we describe our proposed extensions to common recurrent architectures. We begin by introducing the original MHP framework though, defining needed notation and preliminaries. We then describe our contributions, and eventually introduce the novel multimodal metric.

4.3.1 Prerequisites

Let $\mathcal{X}$ and $\mathcal{Y}$ be vector spaces of input and output variables (also called labels) of an examined machine learning problem, and let N be the number of available data points. Further, let $p(x, y)$ denote the joint probability distribution over those spaces. In a classical supervised setting, in which samples (x_i, y_i) are drawn from $p(x, y)$, we are interested in training a predictor $f_\theta : \mathcal{X} \to \mathcal{Y}$, which minimizes the empirical error (empirical risk)

$$\frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(f_\theta(x_i), y_i), \tag{4.1}$$

where $\mathcal{L}$ is an arbitrary loss function and the predictor is characterized by its parameters θ. Equation (4.1) is an approximation of the continuous formulation (actual

risk)

$$\int_{\mathcal{X}} \int_{\mathcal{Y}} \mathcal{L}(f_\theta(x), y) p(x, y) \; dy \; dx. \tag{4.2}$$

which is minimized by the conditional average

$$f_\theta(x) = \int_Y y \cdot p(y|x) \; dy. \tag{4.3}$$

The MHP framework [8] now extends such a one-to-one prediction function f to output M predictions, or hypotheses:

$$f(x) = (f^1(x), \ldots, f^M(x)). \tag{4.4}$$

In the calculation of the loss $\mathcal{L}$, only the best among the M predictions is considered, extending Equation 4.2 to:

$$\int_{\mathcal{X}} \sum_{j=1}^{M} \int_{\mathcal{Y}_j(x)} \mathcal{L}(f^j(x), y) p(x, y) \; dy \; dx. \tag{4.5}$$

In this, $\mathcal{Y} = \cup_{i=1}^{M} \mathcal{Y}_i$ is the resulting Voronoi tesselation of the label space, when using M generators $g^j(x)$ and the loss $\mathcal{L}$ as distance function:

$$\mathcal{Y}_j(x) = \{y \in \mathcal{Y} : \mathcal{L}(g^j(x), y) < \mathcal{L}(g^k(x), y) \; \forall k \neq j\}. \tag{4.6}$$

[8] proves, that in order for Equation 4.5 to be minimal, the generators have to equal the M predictors, and that these have to equal the conditional means of the Voronoi cells they define. A full minimization scheme is given in [8], however such procedure can be drastically simplified and implemented efficiently by introducing a meta-loss on top of any existing loss function $\mathcal{L}$:

$$M(f(x_i), y_i) = \sum_{j=1}^{M} \hat{\delta}(y_i \in \mathcal{Y}_j(x_i)) \mathcal{L}(f^j(x_i), y_i). \tag{4.7}$$

The Kronecker delta δ returns 1, if its enclosed condition is true, otherwise 0, and is used to select the best hypothesis. In practise we have to relax this condition, as all predictors could be initialized so far from the target, that all hypotheses lie in a single Voronoi cell, making this the only one to be updated. Thus, a relaxation using a weight ϵ $(0 < \epsilon < 1)$ is used:

$$\hat{\delta}(a) = \begin{cases} 1 - \epsilon, & \text{if a is true,} \\ \frac{\epsilon}{M-1}, & \text{otherwise.} \end{cases} \tag{4.8}$$

Note that M is a hyperparameter, which needs to be set manually, but that many multimodal prediction methods require such parameter [21, 19].

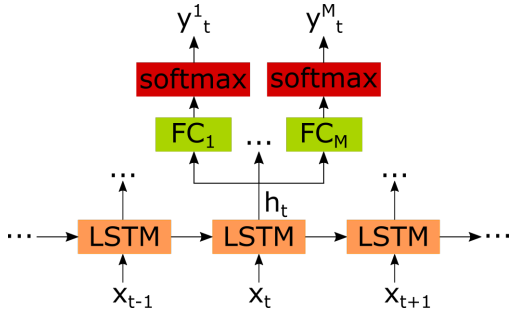


Figure 4.1: Depiction of the proposed sequence-to-sequence architecture. M fully connected layers process the hidden state of the LSTM cell to generate M predictions (image from [5]).

4.3.2 Sequence-to-Sequence Prediction

Our first proposed MHP extension concerns a classical sequence-to-sequence prediction architecture. In this, an input is processed at each timestep, as well as an output generated. Exemplary applications are prediction of driving manoeuvres [39], as well as, in case all frames have to be classified, human activity recognition [76]. To obtain an MHP model, we replace the last fully connected layer used for classification with M copies of it, which do not share weights (see Figure 4.1):

$$\begin{aligned} \mathbf{y}_t^1 &= \texttt{softmax}(\mathbf{W_1 h}_t + \mathbf{b_1}) \\ &\dots \\ \mathbf{y}_t^M &= \texttt{softmax}(\mathbf{W_M h}_t + \mathbf{b_M}) \end{aligned} \tag{4.9}$$

Thus, our loss function needs to be applicable to sequential data. In our analyzed problems we use sequential cross-entropy loss: Let $\tilde{\mathbf{y}}_{i,t}^j$ be the predicted class probabilities at time t, then for a single sample sequence (x_i, y_i) the (binary) cross-entropy loss accounts to $-log(\tilde{\mathbf{y}}_{i,t}^j)$, hence:

$$\mathcal{L}(f^j(x_i), y_i) = \frac{1}{n} \sum_{t=1}^{n} -log(\tilde{\mathbf{y}}_{i,t}^j) \tag{4.10}$$

4.3.3 Encoder-Decoder Architecture

Encoder-decoder models yield promising results for many tasks, such as trajectory generation [42, 41] and machine translation [51]. They consist of two parts, an encoder and a decoder: The encoder processes the input sequence $x_i = (x_{i,1}, \dots, x_{i,n})$ (such as a sentence in the source language), and generates a compressed representation of it (**enc**, e.g. the last hidden state of the LSTM cell). This is fed to a decoder (often, the last hidden state of the encoder is set as beginning state of the decoder), which then generates an output sequence $y_i = (y_{i,1}, \dots, y_{i,m})$ (such as the translation of the source sentence in the target language). We extend this to an MHP model by

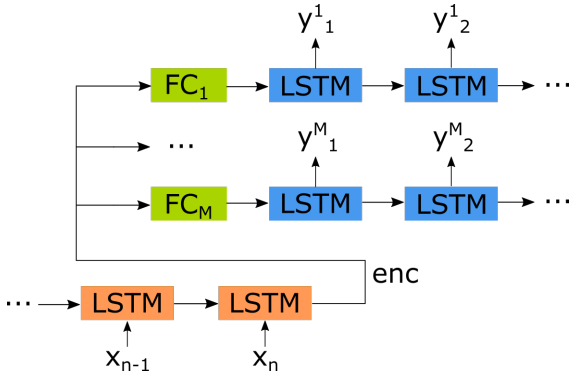


Figure 4.2: Visualization of the proposed encoder-decoder model. M linear layers process **enc**, generating M different starting symbols. With these, the same decoder is run M times (image from [5]).

introducing M fully connected layers in between encoder and decoder, transforming **enc** into M different tokens, and running the decoder M times with these as starting input. Thus, the M fully connected layers have independent weights, while there is only one set of decoder weights. See Figure 4.2 for a visualization:

$$\begin{aligned} \mathbf{enc} &= \texttt{Encoder}(x_i) \\ \mathbf{y}_i^1 &= \texttt{Decoder}(\mathbf{W_1 enc} + \mathbf{b_1}) \\ &\dots \\ \mathbf{y}_i^M &= \texttt{Decoder}(\mathbf{W_M enc} + \mathbf{b_M}) \end{aligned} \tag{4.11}$$

For the loss, again any suitable loss function for sequential problems may be used. In our experiments, we use the L2-loss $\mathcal{L}(f^j(x_i), y_i) = \frac{1}{m}\sum_{t=1}^{m}(\mathbf{y}_{i,t}^j - y_{i,t})^2$.

4.3.4 Sequence Generation

Although for the problem of generating sequences samples still are sequence pairs (x_i, y_i), focus of this task is learning conditional probabilities $P(x_{i,t}|x_{i,t-1}, \dots, x_{i,t-m})$. This way, after training the model, it can be run closed-loop, iteratively generating new data points and sequences of arbitrary length. One can imagine the wide variety of possible application scenarios, such as [77]. We use a similar model as proposed in Section 4.3.2, and train it in identical fashion.

During inference though, we would like to generate sequences of arbitrary length: When now naively predicting M predictions in each step, we generate a tree of possible sequences, in which the number of hypotheses grows exponentially in each step. This is not feasible. Therefore, we introduce several functions, deciding when to merge predictions into one, and when to split into M. This way, we generate M sequences of arbitrary length: As long as predictions are close together, these sequences are identical. Only when predictions differ significantly, we split hypotheses, and from that point in time on follow these independently, merging predictions for

each. Note that these functions can be chosen arbitrarily and in a problem-specific way, and especially do not need to be differentiable, as they are only used during inference. We could, however, also learn them jointly with our network. Although not generating ambiguous outputs, the problem of exploding hypotheses for sequential problems has been known before, thus we would like to refer to this work for comparison [78].

Inference starts with a single hypothesis H, which can be empty, or contain any number of starting symbols. In each step, d future timesteps are inferred. Due to the prediction of M hypotheses in each timestep, a tree of depth d and branching factor M is generated. When points in this tree are, in some problem-specific understanding, close together, the first layer of the tree is merged into one prediction, which is then appended to H. Such merging is done by a function `Merge`, while checking the tree's diversity is by `CheckSplit`. When this function regards the tree diverse enough, function `ChooseTreePaths` identifies M paths $h_1, \ldots, h_M$ of length d through the tree, representing distinct hypotheses. H is subsequently split into M hypotheses, and each extended with one of $h_1, \ldots, h_M$. From now on, these M paths are followed independently, repeating the first steps of our algorithm: The inference tree is formed for each hypotheses, but, as splitting was already done, in every step predictions are merged via `Merge`, and results appended to the respective sequences. Concrete implementations of the black-box functions mentioned here are given in Section 4.4.1. Algorithm 1 contains pseudocode of our describe scheme:

4.3.5 Multimodal Metric

Most works in the field of multimodal prediction use some form of oracle metric, considering only the best hypothesis for evaluating the metric. Formally, if a model returns a prediction set X_i, the oracle version of a standard metric l is given by $l_{oracle} = (X_i, y_i) = min_{x \in X_i} l(x, y_i)$, when y_i is the corresponding ground truth label. Although this metric is minimized by the "correct" prediction and encourages diverse results, better scores could simply be achieved by guessing or predicting a multitude of hypotheses. In terms of information-retrieval terms, oracle metrics reward models for high Recall, while disregarding Precision. Further, in real-world applications sometimes multiple hypotheses are equally likely and correct, and we would like models to predict all of these. When averaging hypotheses or ground truth labels, this often also is no good indicator: An advantage of predicting multiple hypotheses cannot be measured, as the loss often is minimized by predicting the (incorrect) mean. Consider the following sample: There are n trajectories of cars traversing a 2-way intersection, half of which take a left turn and the other a right turn. Before the turn initiation such direction is not yet known and a correct prediction should contain both these cases. A standard single hypotheses model will only predict one, and achieve an accuracy of 50% (when formulating this as a classification problem). An ambiguous model predicting both would also score an accuracy of $0.5 \cdot (0.5 \cdot 1 + 0.5 \cdot 0) + 0.5 \cdot (0.5 \cdot 0 + 0.5 \cdot 1) = 0.5$ (both hypotheses are correct for 50% of all samples and wrong for the other).

Algorithm 1 Inference for MHP Sequence Generation, starting with *start* and inferring for a total of l steps. *BuildTree* expects the starting element as well as the desired tree depth as input.

procedure INFER($start, d, l$)
  `DoneSplit` = False;
  `Predictions` = *start*;
  for i in 1 .. l **do**
    if not `DoneSplit` **then**
      p = last item of `Predictions`
      `tree` = `BuildTree`(p, d);
      if `CheckSplit`(`tree`) **then**
        $[(p_1^1, \ldots, p_d^1), \ldots, (p_1^M, \ldots, p_d^M)]$ = `ChooseTreePaths`(`tree`):
        Append $([p_1^1, \ldots, p_1^M], \ldots, [p_d^1, \ldots, p_d^M])$ to `Predictions`;
        `DoneSplit` = true;
      else
        $[p^1, \ldots, p^M]$ = First layer of `tree`;
        p = `Merge`($[p^1, \ldots, p^M]$);
        Append p to `Predictions`;
    else
      $[p_1, \ldots, p_M]$ = last item of `Prediction`;
      for m in 1 .. M **do**
        p = `BuildTree`(p_m, 1);
        p_m = `Merge(p)`;
      Append $[p_1, \ldots, p_M]$ to `Predictions`;
return `Predictions`

Thus we propose a novel, multimodal metric (M2) for ambiguous problems. Core idea is re-labelling data points, possibly assigning multiple labels to points, and requiring predictions to match all of them. Formally, we extend classical, point-based metrics $l(x_i, y_i)$ to set-based formulations $l_{M2}(X_i, Y_i)$. Our extension can be applied to most existing metrics, regarding sequential and non-sequential problems. We distinguish our derivation for discrete and continuous labels.

4.3.5.1 Discrete Labels

First step towards our new metric is re-labelling data points. For this, we define a polytope $P(c)$ in the input space, containing all points with label c (denoted by $\lambda(x) = c$). Then, for each point x we define its new label set by $Y_i = \{y_i\} \cup \{c \mid c \in C,\ x_i \in P(c)\}$, where C is the set of all possible labels. The polytope P could for example be the convex hull. However, this does not scale well to many dimensions, and the size of the polytope is not reducible. Outliers, in particular, inflate the convex hull, which is not desired here. Therefore, we use $P(c) = \{x \mid \lambda(x) = c,\ min_\tau \leq x \leq max_\tau\}$, and x, min_τ and max_τ are vectors of the d-dimensional input space. Via the threshold τ we can control the size of the polytope, a value of τ indicating that a fraction of (approximately) τ points labelled c lie in the polytope. To achieve this, set min_τ^d and max_τ^d equal to the $([1 - \frac{\tau}{2}] \cdot 100)$th and $(\frac{\tau}{2} \cdot 100)$th percentile, respectively, in each dimension d. The threshold τ thus is freely choosable, and is best set in a problem-specific way, s.t. the resulting labelling resembles our human understanding of ambiguity in the problem. Nevertheless, given a specific threshold results obviously are comparable.

This procedure is applicable to arbitrary high-dimensional spaces (consider e.g. treating MNIST images as 784-dimensional vectors). However, in some cases it can be beneficial to work with lower-dimensional latent spaces, e.g. using autoencoders or intermediate layers of CNNs, and re-labelling according to these spaces (similar to [72]). In Section 4.4.1 we show resulting polytopes of different problems.

Most metrics can be extended using these principles. Here, we exemplarily show this using the widely-used metrics Precision (Pr) and Recall (Re). The extension to multimodal problems is motivated by the same principles behind these metrics in non-ambiguous situations: All predicted hypotheses should be contained in the new label set for a Precision of 1. Analogously, a Recall of 1 should indicate, that all labels of the new label set were predicted. Thus we postulate

$$pr(f(x_i), Y_i) = \frac{|f(x_i) \cap Y_i|}{|f(x_i)|} \tag{4.12}$$

and

$$re(f(x_i), Y_i) = \frac{|f(x_i) \cap Y_i|}{|Y_i|} \tag{4.13}$$

for single samples i and ambiguous prediction $f(x_i)$, and overall $Pr_{M2} = \frac{1}{N} \sum_{i=1}^{N} pr(f(x_i), Y_i)$, $Re_{M2} = \frac{1}{N} \sum_{i=1}^{N} re(f(x_i), Y_i)$.

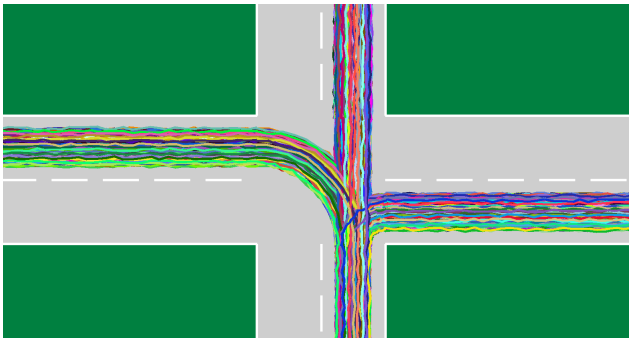


Figure 4.3: Visualization of the used toy intersection. Vehicles approach the intersection from the bottom, and consequently follow of one three possible paths over the intersection (image from [5]).

4.3.5.2 Continuous Labels

For continuous labels, our proposed extension still is motivated by a set-based interpretation of predictions and labels: Each prediction needs to be close to at least one ground truth label, and for every possible label there needs to be at least one prediction nearby. However, as opposed to the discrete case, we cannot define the polytope P immediately: Instead, we first conduct clustering in the label space to obtain discrete classes, and then analogously define $P(c)$ and $Y_i = \{\mu(c) \mid c \in C,\ x_i \in P(c)\}$. Here, $\mu(c)$ denotes the center of cluster c (see Section 4.4.2 for a visualization of this process). Define the extension of a metric $l(x_i, y_i)$ by

$$\begin{aligned} l_{M2}(x_i, Y_i) = &\frac{\sum_{x \in f(x_i)} min_{y \in Y_i} l(x, y)}{|f(x_i)| + |Y_i|} \\ &+ \frac{\sum_{y \in Y_i} min_{x \in f(x_i)} l(x, y)}{|f(x_i)| + |Y_i|} \end{aligned} \tag{4.14}$$

4.4 Evaluation

We begin this section by introducing examined problems and used datasets, and then describe our findings of applying the proposed MHP extensions to these.

4.4.1 Problems and Datasets

4.4.1.1 Toy Intersection

To motivate usage of multiple hypotheses and proposed M2 metric, we begin with a simulated traffic intersection, which offers clear and structured data. We synthetically simulate vehicles crossing the intersection, choosing one of the possible three ways (see Figure 4.3 for a visualization). Trajectories are fully described by a list of 2D-coordinates: $t = ([c_1^1, c_2^1], \ldots, [c_1^n, c_2^n])$

Toy Classification Goal of the classification task is to predict the vehicle's destination (left, straight or right) in every timestep. For this, we apply the sequence-to-sequence model from Section 4.3.2. Denoting with l_i the correct label of sample i, we obtain $x_i = ([c_1^1, c_2^1], \ldots, [c_1^n, c_2^n])$ and $y_i = (l_i^1, \ldots, l_i^n)$. In particular, we already expect models to predict the correct vehicle's intention before it moves in a certain direction, thus introducing an ambiguous space in the input domain, and a non-ambiguous one later on.

Toy Prediction Aim of this task is predicting future vehicle trajectories, conditioned on past ones. We apply the encoder-decoder as well as sequence generation model from Section 4.3.3 and 4.3.4. For both, an initial part of a trajectory, $x_i = ([c_1^1, c_2^1], \ldots, [c_1^m, c_2^m])$, $m < n$, serves as input, while the desired output is the continuation of that trajectory, $y_i = ([c_1^{m+1}, c_2^{m+1}], \ldots, [c_1^n, c_2^n])$. In the encoder-decoder model, x_i is used as input to the encoder, and y_i the expected output of the decoder.

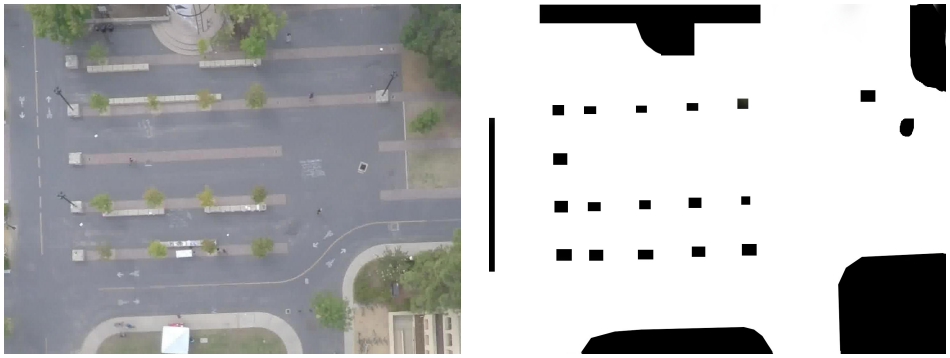
The sequence generation model is trained with full trajectories, in hopes of training it to correctly output further points conditioned on past ones. During inference, we initialize it with all points in x_i, and then run it closed-loop for $n - m$ steps, expecting a trajectory similar to y_i as output. In each step, a tree of depth 8 is created. Function `CheckSplit` calculates the maximal distance between any two points in the last layer of the tree, and outputs a *split* decision, when this exceeds a certain threshold. `Merge` simply returns the mean of M points. `ChooseTreePaths` sorts points in the last layer of the tree by angle, and then returns M equidistant points in the angle space.

4.4.1.2 Lane Change Prediction

We showcase the application of MHP models to the problem of predicting lane changes, discussed extensively in Chapter 3. For this, we use the sequence-to-sequence prediction model from Section 4.3.2. This essentially is a direct multi-modal extension of model *LSTM* from Section 3.3.2, but for simplicity, here we only consider input feature group G^Z. For evaluation we use the NGSIM dataset.

4.4.1.3 Trajectory Prediction

Goal of this task is predicting trajectories of VRUs in different scenes. For this, we employ the encoder-decoder model from Section 4.3.3, and apply it to the Stanford Drone Dataset (SDD) [79]. This, very similarly to the previously used NGSIM and highD datasets, contains birds-eye recordings of different scenes, in which, among others, pedestrians, cyclists and skateboarders walk and ride. In addition to the raw video data, agent positions for every timestep are available. We convert these 2D-coordinates into relative distances, expressing a trajectory by its absolute starting position, and then via successive x- and y-offsets to the previous position. In addition, we generate simple semantic maps of the environment: In these, black



(a) Actual scene image. (b) Semantic map.

Figure 4.4: Comparison of semantic map and captured drone image.

pixels denote obstacles, for example buildings and trees, while white pixels denote free space, such as roads and parks. In every timestep, for each agent we feed a small crop of said map, centered at the agent's location, through a small CNN, and pass the output to our recurrent model. Additionally, the relative 2D-coordinates are used as input. Figure 4.4 shows a sample scene from the SDD dataset along with its corresponding semantic map.

4.4.1.4 Text Generation

To evaluate the performance of text generation models we use the Penn Tree Bank Dataset (PTB) [80], which is a collection of headlines from the Wall Street Journal. Admittedly, this does not allow for as well-sounding results as one would get by, e.g., training on Shakespeare's works, due to the brevity of sentences and abundance of context. However, it is a commonly used and standardized dataset. A vocabulary of the 10000 most frequent words in the corpus is build up, other words are replaced by an *unknown* token. We generate sequences by using the generation model from Section 4.3.4. For the ambiguous model simple functions are used: The tree is built for one step ahead, `Merge` returns the most frequent prediction of the M leaves. `CheckSplit` returns true if all M are different and `ChooseTreePaths` returns the M leaves of the tree.

4.4.2 Results

In this section we describe findings of our experiments and compare proposed MHP extensions to their SHP counterparts and state-of-the art baselines.

To point out advantages and necessity of predicting multiple hypotheses, we compare MHP models against non-ambiguous ones (SHP). The SHP baseline models are identical to the ones introduced in Sections 4.3.2 to 4.3.4, except missing the multimodal part, e.g. only exhibiting one fully connected output layer instead of M. Still, for a fairer comparison, we use multiple predictions from these models, if

possible - consider e.g. taking the n-most likely classes in classification problems. Further, we competitively compare our models against other well-working models for predicting multiple hypotheses. Related, successful models are MDNs, MCL and VAEs. Due to the focus of MDNs on regression, and their difficulty of integration into encoder-decoder models, we do not consider them here. However, Rupprecht et al. conduct experiments comparing MDNs with MHP models, concluding that MHP models compare favourably [8]. Thus, we compare against MCL methods and VAEs. For both of these, we try using identical structures as for the MHP models, i.e. only differing in how ambiguity is handled, but being similar otherwise (e.g. using identical input features and having identical hidden size). Implementation details about each of these models are given in the respective sections.

For analyzing running times and comparing resource consumption, we report wall clock times needed for training (t_t) and inference (t_i) per mini-batch in ms. All experiments are done using a standard laptop containing an NVIDIA Quadro M2000 GPU. Adam optimizer is used throughout with a learning rate of 0.001. Data is split into training, validation and test set with a 3:1:1 ratio, and models trained on the training set, employing early stopping based on the validation set. In the following, all results are reported on the test set.

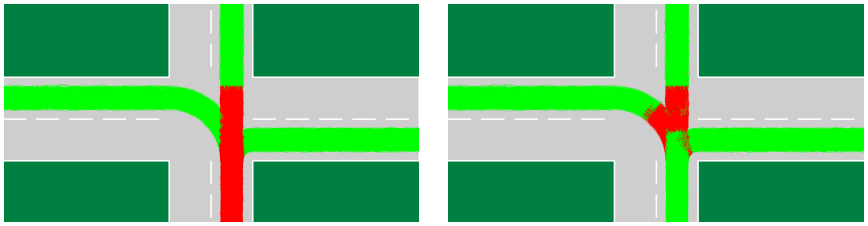
4.4.2.1 Classification

For classification problems we introduce a simple extension of a standard SHP model, dubbed SHP^*, to output multiple predictions: For this, all hypotheses, whose predicted probabilities exceed a certain threshold γ, are returned. We test $\gamma = 0$ as well as a problem-specific γ, which yields best results. To evaluate models, we use the common metrics Precision (Pr) and Recall (Re) as well as the corresponding F1 score ($F1 = 2 \cdot \frac{Pr \cdot Re}{Pr + Re}$), once using an oracle interpretation (Pr_O and Re_O), and once our multimodal interpretation from Section 4.3.5.

We further implement an MCL model by using M separate copies of the SHP model. During training we only backpropagate gradients to the best predictor (thus resembling our training scheme with $\epsilon = 0$). For classification problems we do not see any meaningful VAE extensions.

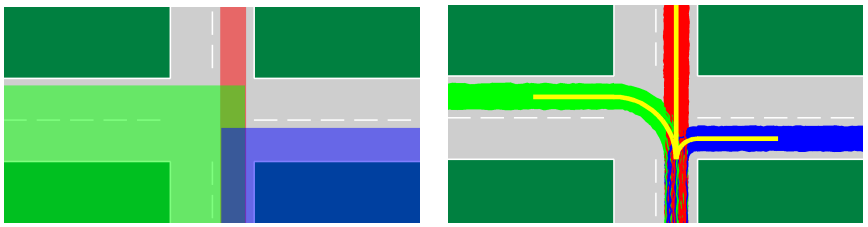
Toy Classification For all models an LSTM cell with hidden size 512 is used, training is done with a mini-batch size of 32. Due to the simplistic nature of the data, the M2 metric parameter τ is set to 1, the MHP relaxation parameter ϵ to 0.15. The used synthetic dataset is made up of 10000 trajectories, each consisting of 75 data points.

Table 4.1 shows quantitative results. The SHP models predict a random hypothesis when initially all three are equally likely, but later on converges to the correct one. This shows through oracle Precision and Recall scores of roughly 2/3. The MHP model performs better, achieving scores of nearly 1, as it correctly predicts all three hypotheses in the beginning. Examining SHP^* with $\gamma = 0$ shows a first downside of using oracle metrics: Now, the model always outputs three predictions,



(a) Results of the SHP model. (b) Results of the MHP model.

Figure 4.5: The correctness of predictions w.r.t. to the M2 metric is visualized: If the set of predictions is identical to the set of labels obtained through the M2 procedure, the point is drawn in green, otherwise in red (image from [5]).



(a) The three resulting polytopes defining new labels are depicted for the classification task (meaning labels are discrete) on the toy intersection.

(b) Also using the toy intersection, this image visualizes results for the continuous trajectory prediction task: The clustering results are shown, the cluster centers are drawn in yellow.

Figure 4.6: Visualization of the re-labelling step conducted for the M2 metric (image from [5]).

which is correct in the ambiguous part of the problem, but wrong later on. Nevertheless, as oracle metrics only consider the best hypothesis, Pr_O and Re_O are 1. Such behaviour is penalized, as expected, by the M2 metric, Pr_{M2} is lower. Conversely, the standard SHP model achieves near perfect M2 Precision, as the single predicted hypothesis nearly always is correct, but a lower Recall. Figure 4.5 visualizes these observations. As discussed, the SHP model is wrong in areas with ambiguous outcomes, the MHP model only in a small area around the center of the intersection.

Table 4.1: Results of the toy classification task (γ repoted in brackets).

Name	Pr_O	Re_O	Pr_{M2}	Re_{M2}	$F1_{M2}$	t_t	t_i
SHP	0.683	0.683	**0.999**	0.649	0.787	**205**	**105**
SHP* (0)	**1.0**	**1.0**	0.676	**1.0**	0.807	205	105
SHP* (0.01)	0.903	0.903	0.750	0.893	0.815	205	105
MCL	**1.0**	**1.0**	0.677	**1.0**	0.807	535	237
MHP	0.999	0.999	0.980	0.984	**0.982**	210	113

This could be due to the fact, that these samples are the hardest to classify, but also stemming from artifacts caused by the rectangular shape of the polytopes used in the M2 metric. Figure 4.6a shows the polytope being used in the re-labelling step.

Similar to SHP^* with $\gamma = 0$, the MCL model always outputs three distinct predictions, achieving similar results - perfect oracle scores, but deductions in the M2 metric. This is due to the fact, that no penalty for too diverse predictions exists ($\epsilon = 0$).

As the MCL model consists of M full models, the number of parameters as well as time needed for training and inference grows. Compared to that, our MHP model only exhibits minimal parameter overhead, and clocks similar times as the simple SHP model. Although in some categories other models rank better, overall our proposed MHP model does show the best understanding of ambiguity for this problem, clearly identifying ambiguous and non-ambiguous situations, and yields best results overall.

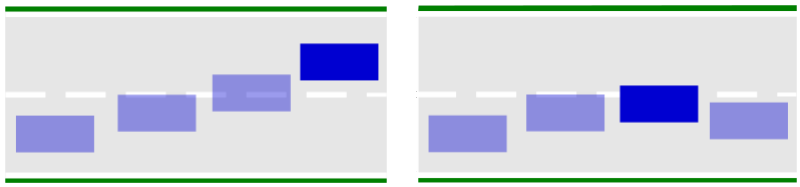
Lane Change Prediction As mentioned in Section 4.3.5, parameter τ should be chosen such that the resulting labeling resembles our human understanding of ambiguity in the problem. We find $\tau = 0.85$ to be well suited for this. Figure 4.7 shows two samples and their respective new labels, which coincide with human labeling. From the NGSIM dataset we use the subsets US 101 and I80, which contain about 1.200.000 single frames. Table 4.2 shows the number of samples of each possible class before and after the re-labelling. All models use a hidden size of 128, a mini-batch size of 256 and $\epsilon = 0.15$. Table 4.3 lists results. Overall, results are very similar to the toy prediction task, with our MHP model performing best.

Table 4.2: Number of samples, ordered by label class. Originally, each frame is given one of the labels L, F or R. After re-labelling, similar to the toy classification task, multiple combinations of labels are possible.

Label	# Samples
L / F / R (original)	13241 / 1179384 / 3636
LFR	1403
LF / RF / LR	94358 / 179085 / 178
L / F / R	12590 / 906532 / 3447

Table 4.3: Results on the NGSIM dataset (γ listed in brackets).

Name	Pr_O	Re_O	Pr_{M2}	Re_{M2}	$F1_{M2}$	t_t	t_i
SHP	0.766	0.766	**0.797**	0.747	0.771	**388**	**263**
SHP* (0)	**0.952**	**0.952**	0.352	**0.952**	0.514	388	263
SHP* (0.55)	0.799	0.799	0.572	0.779	0.660	388	263
MCL	**0.952**	**0.952**	0.364	**0.952**	0.527	1005	637
MHP	0.905	0.905	0.732	0.906	**0.810**	658	506



(a) A lane change to the left is executed smoothly, the sample is labeled solely with L.

(b) This sample is labeled with LF, also a human might be unsure about the driver's intentions.

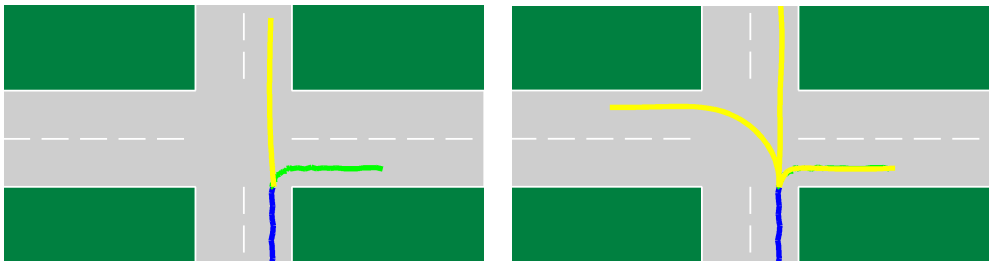
Figure 4.7: Depiction of the new labels generated by the M2 metric.

4.4.2.2 Regression

In this section the results of encoder-decoder models are shown, applied to a toy and a real problem. Standard SHP encoder-decoders cannot easily be modified to generate multiple outputs, thus in this section we do not consider extensions regarding multiple predictions. We implement MCL and VAE models though: Again, the MCL model consists of M full copies of the SHP encoder-decoder model. In the VAE model, the encoder learns a mapping into a 20-dimensional latent space, modelled by a variable z, from which we sample M times and generate M trajectories via the decoder. Internally, the latent distribution is modelled via calculating mean and variance, and sampling is done via the common reparametrization trick. As loss the average error over all hypotheses is used, combined with the Kullback-Leibler divergence between z's distribution and a Normal distribution, similar to [41]. To assess model performances, we use the two metrics *Average Displacement Error* (ADE) and *Final Displacement Error* (FDE), which measure metric distances between trajectories, averaging them over all time steps or only considering the last one, respectively. To cluster trajectories, as is required for the M2 extension of metrics, we convert trajectories of n timesteps to a $2n$ dimensional space, and here apply mean-shift clustering.

Toy Prediction We again use the toy intersection introduced in Section 4.4.1.1 to obtain first results of the prediction task and for motivating the need for multi-modality. The used dataset consists of 10000 simulated trajectories containing 60 data points each. The first 30 are used as input to the encoder (x), while the remaining 30 are to be generated by the decoder (y). Encoder and decoder LSTMs consist of 64 hidden units, we use $\epsilon = 0.15$, $\tau = 1$ and a mini-batch size of 256. The bandwidth used for mean-shift clustering is left to be determined automatically by the algorithm.

Predictions for one sample are depicted in Figure 4.8. In this, the simulated vehicle takes a right turn, while the SHP model predicts a random trajectory. The MHP model nicely generates all 3 possible and equally likely outcomes, also matching the actual ground truth trajectory nearly perfectly. Quantitative results are listed in



(a) Prediction of the SHP model. (b) Prediction of the MHP model.

Figure 4.8: Resulting prediction for a sample trajectory, in which the corresponding vehicle is making a right turn. The decoder's input is depicted in blue, the ground truth in green, and the predictions in yellow (image from [5]).

Table 4.4. Since the SHP model only outputs a single, random (averaged) prediction, this mostly is distant from the actual ground truth, resulting in bad metrical scores overall. Similar behavior is observed for the VAE model: Inputs are close together, resulting in averaged, low-variance distributions of the latent variables. The MHP and MCL models perform much better: both are able to generate diverse predictions covering all possible trajectories. However, again the resource consumption of the MCL model is higher. Figure 4.6b shows the resulting clusters used in the M2 metric.

Trajectory Prediction In this section we describe our findings of predicting trajectories using the SDD. In particular, to better compare against state-of-the art models, we use the SDD part of the TrajNet dataset [81]. Here, trajectories of the SDD have been filtered (overly linear trajectories are filtered out) and preprocessed to a standard format: 8 time steps (3.2s) are to be used for initializing the encoder, 12 time steps (4.8s) need to be predicted. Labels for the test set are not public (yet), as there are ongoing challenges to find best-performing trajectory prediction models and so far, only submissions predicting single hypotheses are supported. Thus we created our own test set by splitting the publicly available training part, which might explain the slightly different (better) results of our re-implementations. We experiment with 3 and 5 hypotheses, and use an LSTM size of 256. Further, we use

Table 4.4: Results of the toy prediction task.

Name	FDE_O	ADE_O	FDE_{M2}	ADE_{M2}	t_t	t_i
SHP	2.82	4.60	2.29	3.64	**112**	**49**
MCL	**0.15**	**0.16**	0.53	**0.53**	320	147
CVAE	2.76	4.53	1.99	2.95	218	83
MHP	0.26	0.22	**0.51**	**0.53**	210	88

(a) Prediction using 5 hypotheses. (b) Prediction using 3 hypotheses. (c) Prediction of the SHP model.

Figure 4.9: Comparing predictions of different models in an ambiguous scene of the SDD (image from [5]).

(a) Prediction using 5 hypotheses. (b) Prediction using 3 hypotheses. (c) Prediction of the SHP model.

Figure 4.10: Resulting predictions in a non-ambiguous scene of the SDD.

$\epsilon = 0.15$, $\tau = 1$, a mini-batch size of 256 and a bandwidth of 250 for the mean-shift clustering.

Figures 4.9 pictures results of a sample scene, which exhibits large amounts of ambiguity and uncertainty: The agent in question traverses a round-about, and has multiple possible ways of continuing its path. The SHP model simply predicts a lin-

Table 4.5: Results on the SDD.

Name	FDE_O	ADE_O	FDE_{M2}	ADE_{M2}	t_t	t_i
SHP	35.94	17.76	49.14	40.36	**238**	**143**
MCL 3	27.52	14.46	45.02	41.89	338	187
MCL 5	25.31	13.58	43.81	42.52	450	227
VAE 3	35.07	17.16	44.06	38.86	311	166
VAE 5	34.06	16.62	**41.86**	**38.23**	377	193
MHP 3	27.20	14.36	42.66	39.03	312	175
MHP 5	**24.89**	**13.48**	41.96	39.11	355	187
DESIRE	34.05	19.25	-	-	-	-
SoPhie	29.38	16.27	-	-	-	-

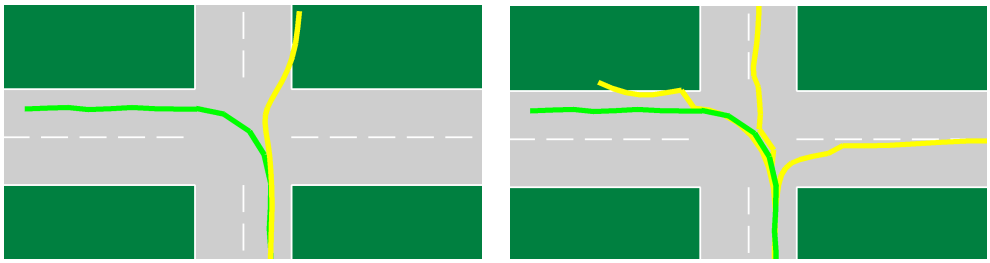
ear follow-up trajectory, while the MHP models generate diverse outputs, accurately matching predictions to possible targets in the environment. More predictions yield more fine-grained hypotheses. In Figure 4.10 more linear motions are considered, the agents mostly follow the existing sidewalk. All predictions thus lie in a narrow space, predicting a straight trajectory. Note though, that when employing 5 hypotheses, there is one which ends nearly immediately. This could represent a rare but plausible outcome of a person stopping, e.g. to chat.

In Table 4.5 quantitative results are shown, overall proving that proposed MHP models perform better than their SHP counterpart, yielding better results for all metrics. Compared to the toy task, the VAE models seem to improve, generating more diverse outputs. Still, their predictions lack the diversity of our MHP models, as indicated among others by higher oracle scores. The MCL models perform comparably to our MHP extensions, still yielding slightly worse results. In terms of running times, similar findings from previous experiments repeat: MCL models clock slowest times, while MHP and VAE models are close together, with MHP models scaling slightly better with growing M. Further, we compare against more complex models we consider state-of-the art for VRU trajectory prediction, namely SoPhie GAN [43] and the DESIRE framework [41] (results for this obtained from the interpolation in [43]). As no source code or full implementation details for these models are made public, we simply report numbers from the corresponding papers. The results are relatively similar to the ones obtained in our experiments, although our models fare slightly better. We would like to point out again, that we used a different test set due to our need for multimodal evaluation, but argue, that overall the findings confirm capacity of our MHP extension to accurately capture and model uncertainty and ambiguity, also for highly complex datasets.

4.4.2.3 Sequence Generation

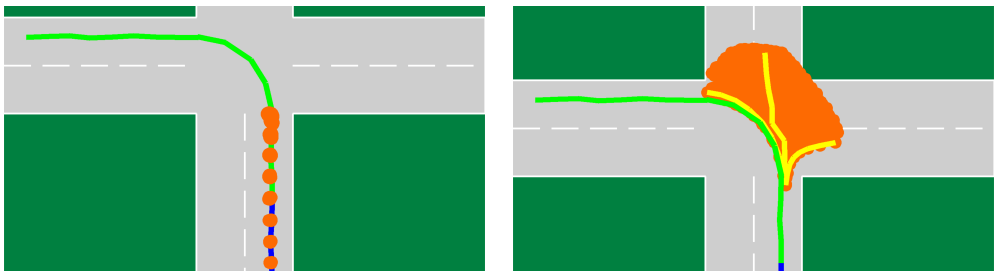
In this section we demonstrate our MHP extensions for sequence generating problems, namely predicting trajectories on the toy intersection and text generation. For the former we again use the metrics FDE and ADE, for the latter a measurement from information theory, i.e. Perplexity.

Toy Prediction Extending common models to generate multiple sequences, while simultaneously controlling the exponentially growing number of hypotheses, requires different changes in the examined models. Thus we here only test our proposed MHP extension as sole representative of ambiguous models, and compare this against a standard SHP model. The synthetic dataset involving the toy intersection contains 10000 simulated trajectories with 25 points each. Both LSTMs used have a hidden size of 256, we use a mini-batch size of 256, as well as $\epsilon = 0.15$ and $\tau = 1$. After training, we initialize the models with 5 sample points, and infer for 20 more. In Figure 4.12 two trees created at different timepoints, one early in the scene and one right at the intersection, are shown. In the first case, the resulting tree is very narrow, as there is no ambiguity in the predictions. In the second case the tree correctly



(a) Resulting prediction of the SHP model. (b) Resulting predictions of the MHP model.

Figure 4.11: Comparison of generated trajectories using the SHP and MHP models. The ground truth is depicted in green, the predictions in yellow (image from [5]).



(a) Tree at step 1. (b) Tree at the beginning of the intersection.

Figure 4.12: Visualization of the created tree at different timesteps. Points of the tree are drawn in orange, the resulting hypotheses produced by `ChooseTreePaths`, in case `CheckSplit` returns true, in yellow.

covers the full possible label space of the intersection, and the function `CheckSplit` returns true. Figure 4.11 shows the resulting predictions for one sample. Again we observe, that the SHP model (expectedly) generates an averaged trajectory, while our MHP extension is able to correctly infer all three possible directions. Table 4.6 shows quantitative results. Again, the MHP model outperforms the SHP model. As training schemes for both models are identical, so are their clocked times for training. During inference though, while SHP models only need to follow a single hypothesis, MHP models generate hypotheses trees in each step, significantly increasing inference time.

Table 4.6: Results on the toy sequence generation task.

Name	FDE_O	ADE_O	FDE_{M2}	ADE_{M2}	t_t	t_i
SHP	18.26	5.07	14.07	4.13	**59**	**188**
MHP	**5.45**	**1.99**	**5.48**	**2.06**	67	24433

Text Generation A widely used metric for evaluating generated sequences is Perplexity [82], which describes how well the predicted distribution matches the target distribution and is defined by $2^{H(p)}$, where $H(p)$ is the entropy. To assess the performance of a model, this is often evaluated on the test corpus. However, since we want to show how to employ MHP models for practical uses and assess upon multiple hypotheses, we directly measure the Perplexity of the generated samples. This can be done using n-grams (patterns of n consecutive words): the n-gram Perplexity is given by $p_n = \sqrt[N]{\prod_i^N \frac{1}{P(w_i|w_{i-1},\ldots,w_{i-n}}})$. Here, $P(w_i|w_{i-1},\ldots)$ denotes the probability of word w_i following the words $(w_{i-1},\ldots)$. A lower perplexity is better, i.e. describing a model outputting more realistic sequences. Motivated by the BLEU score [83], we use the Perplexity measure $P = p_2 \cdot p_3 \cdot p_4$. Since now we have a tool to immediately measure the quality of created hypotheses, there is no need to use oracle or M2 variants of the metric.

To better understand the extent of ambiguity in this problem, we propose different evaluation schemes. For each, we start with a random starting word and produce words until the end of sentence token is generated or a fixed maximum length is reached. We repeat each experiment 10000 times, using the same starting words for the SHP and MHP model:

1. Generate one sentence from each starting word. Use the top prediction from the SHP model in each step, and a random one from the MHP model.

2. Generate a tree containing all possible predictions of length 8, inserting the M predictions from the MHP model as children in each step. For the SHP model, the M-most probable predictions are used.

3. Generate M full sentences, by using the problem specific `split` and `merge` functions. For the SHP model again the M-most probable predictions are used.

Two LSTM layers with hidden size 128 are used and $\epsilon = 0.05$. The results are shown in Table 4.7: for each method the mean perplexity is reported. For all evaluation scenarios, the MHP model outperforms the SHP one, even for evaluation scheme 1. This is somewhat surprising, as the SHP model uses its best prediction in each step whilst the MHP model a random out of M possible ones. This indicates the ambiguous nature of this problem and that a sequence generation model should not be trained by maximizing likelihood using only a single hypothesis. For scheme 2, the average Perplexity returned by the MHP model is again lower, but the variance higher: This shows, that overall the MHP model is able to explore a wider space of possible sentences. In scheme 3 mean and variance of the MHP model are lower, the MHP model seems to be better suited for understanding when to split into different paths and when to merge, due to its better understanding of the prediction space. Figure 4.13 shows some generated sample sentences.

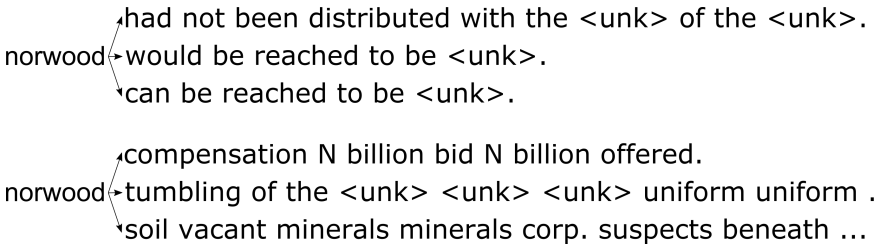


Figure 4.13: Sample sentences created with scheme 3 (top: MHP model, bottom: SHP model). The MHP model creates more coherent and less repetitive results.

4.5 Conclusion

Ambiguity and uncertainty are present in many machine problems, especially in the area of autonomous driving: Pedestrians as well as other vehicles might follow different, sometimes unexpected trajectories, in many situations multiple driving manoeuvres are equally likely. Still, many works focus on learning a one-to-one mapping between inputs and outputs and predicting the best hypothesis in every situation. Thus, we set out to introduce and model ambiguity in this field, extending the previously introduced prediction model from Chapter 3 as well as analyzed a multitude of different problems to show general applicability of our approach.

In particular, we extended the MHP framework [8], which was originally introduced for feed-forward networks, to recurrent architectures. Main theme of said framework is partitioning the label space using a Voronoi tesselation, and only backpropagating gradients inside Voronoi cells. In particular, we showed how to extend such principle to sequence-to-sequence prediction problems, encoder-decoder architectures and sequence generation models. We addressed a diverse set of problems, such as predicting lane changes (Chapter 3), trajectories and even generating text. In several experiments we compared proposed MHP models against other well-performing methods for predicting multiple hypotheses, such as VAEs and MCL, and state-of-the art methods for specific problems. Overall, results regarding our MHP models are promising, showing good performance and an accurate understanding of ambiguity. To better assess and compare models, we further introduced a novel multimodal metric, core of which is re-labelling data points, possibly assigning multiple labels and requiring predictions to match all of them.

Table 4.7: Means (and variances) of the Perplexity scores of the different evaluation schemes on PTB (lower Perplexity is better).

Name	Scheme 1	Scheme 2	Scheme 3
SHP	7.02	6.87 (3.40)	7.33 (1.94)
MHP	**6.02**	**5.79** (4.58)	**5.37** (0.58)

In the future, we would like to extend experiments, comparing different architectures for predicting and generating diverse outputs, such as GANs and methods from VI, on standardized datasets. Demasking the current black-box understanding of many deep learning models, building up knowledge regarding how and why they work, and particularly in which situations they excel and fail for ambiguous situations, seems like a worthy goal.

Part III

Planning

5

Assessing Situations

A traditional stack-based architecture for autonomous agents consists of the layers perception, fusion, prediction, planning and control. After having covered the prediction part in the previous chapter, we now turn our focus towards planning. Possible data-driven methods for solving this problem are RL and IL, in which models learn by themselves via trial and error or by trying to mimic designated experts' behaviour. While both have their merits and led to promising results in recent times [27, 36], also downsides are known: RL depends on the existence of a realistic simulator and still suffers from domain shift when transferring to the real world, as well as discards the large amounts of driving data available. IL is known to overfit specific expert's behaviour and suffers from compounding errors. Therefore, most companies still rely on conventional methods for planning and control, such as rule-based planners and mathematical optimization methods to calculate ideal trajectories. In addition, such "simpler", hand-crafted methods are better interpretable and thus verifiable, which is a strong requirement for safety-critical applications.

Thus, here we take a step back, and position ourselves between the prediction and planning layer: Goal is the introduction of a novel layer for understanding complex scenes, which consists of a deep learning architecture, thus making use of this powerful field. This then serves as supporting layer to other functions, such as planners, effectively offering the benefits of deep learning and combining it with any other potential algorithm, such as a safe, rule-based planner. Particularly, we revisit the already previously examined lane changes: Goal is training a model for answering the question, whether such a manoeuvre is safe and feasible at the moment. Such principle can easily be extended to other manoeuvres and tasks though, final goal is providing a complex layer, that can serve as "brain" of the car, preparing and yielding information and recommendations to a multitude of functions.

For this, we train an LSTM network in supervised fashion on actual driving data, and further extend it to a bidirectional model by incorporating the *Intelligent Driver Model* (IDM) [38] for explicit future state prediction. Summarized, our contributions are:

- We propose a bidirectional RNN, making use of the IDM for future state prediction, for understanding complex traffic scenes, analyzing situations w.r.t. their suitability for lane changes.
- We propose a novel labelling scheme for such problem and compare it to existing ones.

5.1 Introduction

Decision making is a crucial part in any autonomous system - in fact, it is the distinct factor providing such systems with the ability to move autonomously. Possible data-driven methods are RL and IL. While they show first promising results in the field of autonomous driving [27, 36] and excel in many other fields [60, 84], companies are not ready to rely on such mainly black-box models yet. In particular, downsides like the need for realistic driving simulators in RL and compounding errors in IL, make a practical application in the near future doubtful. We instead propose a new supporting layer, reformulating the problem as a binary classification problem. Such problems have been addressed before, e.g. using DBNs and SVMs [85, 86], mostly using only shallow, frame-based approaches though.

Here, we again exploit the sequential nature of such problem and driving scenarios in general, introducing an LSTM network for classifying dynamic driving situations into the categories suited and not suited for lane changes. Considering the sequential aspect of the problem should greatly help understanding of complex situations. We further extend our method with help of the IDM: In each timestep, we use this for predicting future scene configurations, and use this information in a bidirectional LSTM. This way, we incorporate prior knowledge about rational drivers, and help models by providing explicit assumptions about the future. The fact that this helps improving model performance can indeed be viewed with some curiosity, as all information is available to the unidirectional LSTM already, and it could learn to implicitly anticipate. However, breaking down complex models into well-defined parts with specific purposes is a general well-working optimization technique, and so is adding any prior knowledge. In fact, our method can be compared somewhat with *self-knowledge distillation* [87], in which models learn to optimize predictions by using their own outputs as feedback.

As in the previous sections, we use intermediate feature representations to model and describe the examined problem. Most works follow this principle, although some also use raw camera data as inputs [88].

Finding appropriate labels for such problems turns out to be a crucial task. Most works apply a principle we name “action-based labelling”, observing executed lane change manoeuvres and deriving labels based on this behaviour. While this results in strong positive labels, negative situations are problematic to define, as it is hard to distinguish between situations in which drivers do not want to lane changes and in which they want to but cannot. Therefore, we propose an alternate “automatic” labelling scheme, considering safety distances and deriving labels from these.

5.2 Related Work

Full data-driven methods for planning tasks employed in autonomous agents often make use of either IL or RL. Such techniques can be distinguished in terms of which features and abstraction layers are used (using raw sensory inputs [37] or intermediate representations [89, 27]), and can be applied for full end-to-end systems [36, 37] or "only" for the planing layer itself in a stack-based architecture [89]. Interesting candidates for true end-to-end driving are [27] and [37]. In [27], the authors define a custom RNN network, which is given fused sensory information and learns to output future trajectories of the ego vehicle, trained on actual driving behaviour. Different sensor inputs are fused into high-level representations of the environment in birds-eye view: Important information such as lane boundaries and other agents are plotted in different schematic images. Such representation has the advantage of being easily transferable to different domains, and that synthetic data can be generated easily. In addition to losses penalizing deviations of produced driving behaviour from the ground truth data, several auxiliary losses are introduced, e.g. prompting models to predict future positions of other agents. Such joint learning of different tasks is shown to help improve performance and robustness. Zeng et al. work on raw LiDAR inputs, also combining prediction and planning tasks [37]. From those inputs, future trajectories of agents are predicted, and simultaneously a cost volume is generated by sampling from actual and random trajectories. Eventually, a cost-optimal trajectory is sampled based on this volume.

As mentioned in the introduction, despite their impressive results such data-driven methods also have their weaknesses. Thus, we here follow a stack-based architecture and introduce an intermediate supporting layer to the planning layer. This way, we exploit the power of deep learning and circumvent any issues arising from using learned planners. Thus, most related to our work are classification models analyzing situations w.r.t. their suitability for lane changes. To put this into context, in existing works lane changes are often categorized into three classes: *Mandatory*, *discretionary* and *anticipatory* lane changes. Mandatory lane changes are forced upon the driver due to environmental conditions, such as terminating lanes, while discretionary lane changes are conducted to improve driving conditions, e.g. merging into faster lanes. Anticipatory lane changes are closely related, but done in anticipation, i.e. to improve future driving conditions (such as avoiding congestion on one lane). Further, execution of a lane change is often partitioned into multiple phases, beginning with the decision to change lanes and ending with the acceptance of a suitable gap to merge into. Our model effectively is such a gap acceptance model, but monitors complete scenes and also extends to more dynamic situations.

Nie et al. apply SVMs for such gap acceptance problem, using them to classify gaps as suited or not suited for lane changes [86]. Balal et al. address a similar problem, employing fuzzy logic rules for solving it [90], with aims of supporting drivers in their decision making. Jeong et al. also aim at classifying gaps, but by using raw image data and employing CNNs for the task [88]. Ulbrich et al.

apply DBNs for classifying situations, also analyzing whether those are suited and beneficial for lane changes [85].

5.3 Recurrent Models for Assessing Situations

First, we formally introduce inspected problem and used labelling schemes, and then explain models used for solving it.

5.3.1 Problem Definition

Goal of this task is deciding, whether the ego vehicle can safely change lanes. We formulate this as a supervised classification problem: At each timestep, our algorithm processes the current driving scene, and generates a binary output, indicating whether a lane change to the respective side is possible or not. Used input features are features from group G^E (compare Section 3.3.1), i.e. distances to neighbouring vehicles. We execute our algorithms twice, once for each side: In particular, when examining possible lane changes to the right, only features dt_{PV}, dt_{RV}, dt_{PLV_R} and dt_{PFV_R} are considered. This is hidden from the algorithms though by selecting appropriate features and labels beforehand during training and inference. For obtaining ground truth labels, we propose two labelling schemes:

Action-based Labelling Many existing works employ a principle we call "action-based labelling" [86, 90, 91]: Lane changes conducted by human drivers serve as positive examples, marking situations suitable for lane changes. Unsuited situations unfortunately are harder to find. We improve upon this in Chapter 6. In particular, a lane change is considered started once a vehicle moves with a lateral velocity of more than 0.213 m/s towards a lane boundary without oscillations, and ends, once it crosses this lane boundary. We call this period T_P and label all contained frames with Y (*Yes*), indicating suitability for lane changes. To obtain negative labels, it is assumed that drivers assess situations, looking for suitable gaps, before executing a lane change. Thus every T_P is preceded by a time period T_N, in which drivers want to lane changes but cannot yet. We choose a duration of 5s for T_N, labelling each contained frame with N (*No*). In addition, to give models time to adjust and not force hard boundaries, we introduce a time period of duration 1s (T_I) in between, which is given label I (*Ignore*). To further avoid false negative labels, we try to formally quantify suitability for lane changes by evaluating temporal distances: Intuitively, we only would like to consider situations in our training set, in which a significant situational change can be detected from T_N to T_P. If such change is not observable, our assumption is likely to be wrong. Conversely, a strong change might indicate changes in traffic swaying drivers' decision from keep lane to change lane.

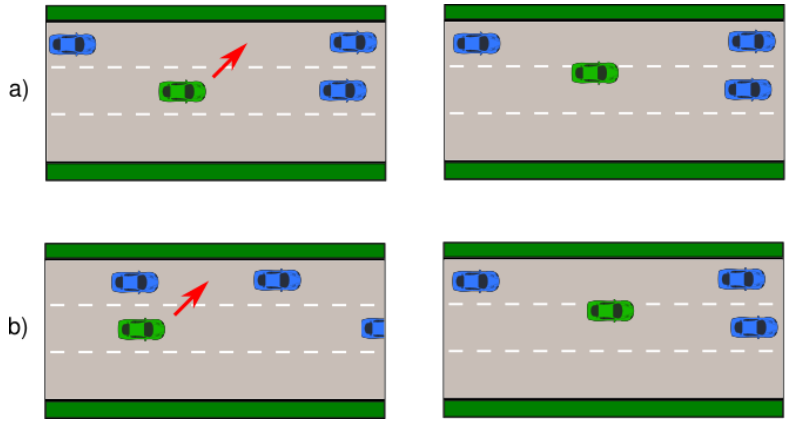


Figure 5.1: Depiction of two sample lane changes, one is considered in our training set, while the other is not. In both scenes, the lane change of the target car (drawn in green) happens in the second frame. In scene a), frames during T_N and T_P are nearly identical, thus this sample is removed from the training set. In scene b) however, we observe a significant change in situations, increasing the chance of this example having meaningful label (image from [3]).

We formalize this by defining for any driving scenario:

$$\begin{aligned} ad &= d_{PV} + \gamma * d_{PLV} + d_{RV} + \gamma * d_{PFV} \\ &\quad + \beta * (|v_{PV}| + \gamma * |v_{PLV}| + |v_{RV}| + \gamma * |v_{PFV}|) \\ sd &= v_{PV} + \gamma * v_{PLV} - v_{RV} - \gamma * v_{PFV} - ad \end{aligned} \tag{5.1}$$

d denotes spatial distances from the corresponding vehicles to the ego vehicle and v relative velocities (target's velocity subtracted from ego velocity). ad is a crude approximation for the scene's general state, high values indicate among others large safety gaps. sd further approximates a scene's suitability for lane changes: sd is negative, when ego is slower than preceding vehicles and faster than trailing vehicles, and when distances overall are large. Thus, scenes with small sd are more likely to be suitable for lane changes than scenes with high sd. In particular, motivated by observed driving behaviour and to weigh the importance of current and desired driving lane, we use $\gamma = 2$ and $\beta = 1.8$ and only include the pair (T_N, T_P) in our training set if $\frac{|ad_{t_{n_0}} - ad_{t_{p_0}}|}{ad_{t_{n_0}}} \geq 0.35$ (1) and $sd_{t_{n_0}} \geq sd_{t_{p_0}}$ (2) (index 0 denoting that first frames of each period are considered). (1) forces a minimal change in situations, while (2) guarantees the better suitability of frames in T_P. Figure 5.1 visualizes this principle.

To prevent LSTM networks from memorizing the fixed structure (T_N, T_I, T_P), we introduce augmentations in our training set, clipping sequences to random intervals.

Automatic Labelling As alternative to such behaviour-oriented labelling, we propose an automatic labelling scheme, which considers future safety distances on the target lane to identify safe manoeuvres. When labelling a frame t, distances to

preceding and following vehicle on the respective target lane ($dt_{\texttt{PLV}_L}$ / $dt_{\texttt{PFV}_L}$ and $dt_{\texttt{PLV}_R}$ / $dt_{\texttt{PFV}_R}$) are considered: If these do not fall short of 1s over the course of the next 3s, the label is set to Y, otherwise N. This follows the assumption, that a lane change can be executed safely, if all involved participants continue driving in their intended manner, i.e. no strong acceleration or breaking is necessary. In particular, if such safety gaps of 1s are adhered to given the actual, future driving behaviour, it is possible to merge into the respective target lane without influencing other agents much.

Advantages of such automatic labelling are the number of labelled data frames, as well as avoiding falsely labelling frames negatively. Using this scheme, all frames of the dataset are labelled, while using the action-based scheme, only frames adjacent to lane change manoeuvres are labelled, and the rest ignored. The minimal required safety gaps can be modified to model more passive or aggressive driving behaviour. However, in practise, especially in dense traffic, such strict gaps might not be realistic nor applicable: Often, collaboration between agents is necessary, humans tend to anticipate and negotiate future driving manoeuvres. Such behavior could be learned by demonstration, and we consider this in Chapter 6.

5.3.2 Models

In this section we first introduce a standard LSTM network for addressing the discussed problem of analyzing situations. Then, we extend this to a bidirectional LSTM. This effectively combines two LSTM networks traversing information in different temporal directions. Since during inference, future information is not available, we propose including a prediction component, namely the IDM.

5.3.2.1 LSTM Network

As inputs to our LSTM model serve distances to preceding and following vehicles on current and desired driving lane in discretized fashion: We convert each continuous distance to a one-hot vector, summarizing 100m with a resolution of 10m. An embedding is used to project into a higher-dimensional space, which is then passed to the LSTM network. Finally, a fully connected layer followed by a `softmax` function outputs the classification result. Note, that strictly speaking we do not use the feature set G^E as input, but a slight deviation from this (using actual distances d instead of temporal distances dt). The model is visualized in Figure 5.2, and described formally as follows:

$$\begin{aligned} \mathbf{e}_t &= \texttt{emb}(\mathbf{W}_{emb}, \mathbf{b}_{emb}, [\mathbf{g}_t^{pv}; \mathbf{g}_t^{rv}; \mathbf{g}_t^{pfv}; \mathbf{g}_t^{plv}]) \\ (\mathbf{h}_t, \mathbf{c}_t) &= \texttt{LSTM}(\mathbf{e}_t, \mathbf{h}_{t-1}, \mathbf{c}_{t-1}) \\ \mathbf{o}_t &= \texttt{softmax}(\mathbf{W}_o \mathbf{h}_t + \mathbf{b}_o) \end{aligned} \tag{5.2}$$

In this, $\mathbf{g}_t^x$ denotes the one-hot vector for vehicle x as given by the discretization of d_t^x described above, and $\texttt{emb}(\mathbf{W}, \mathbf{b}, [x_1, x_2, \ldots]) = [\mathbf{W}x_1 + \mathbf{b}; \mathbf{W}x_2 + \mathbf{b}; \ldots]$.

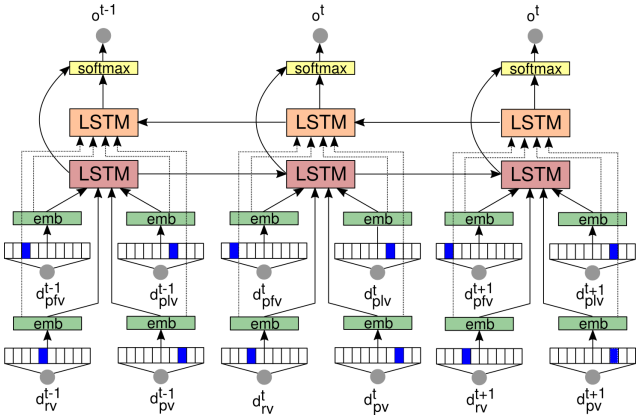


Figure 5.2: Visualization of the used LSTM networks. The distances to neighbouring vehicles on current and desired driving lane are used as inputs and encoded as one-hot vectors. These are embedded and passed on the LSTM network, a final fully connected layer and `softmax` function generates the resulting prediction. For the standard LSTM network, only the LSTM cell colored in red is used. In the bidirectional LSTM, also the orange colored LSTM is used, which processes the information in reversed temporal fashion (image from [3]).

5.3.2.2 Bidirectional LSTM Extension

A bidirectional LSTM network is composed of two separate LSTM networks, the extra one processing data backwards. In each step, the output of both networks is combined and used for prediction. Thus, in each timestep past as well as future information can be used for generating an output.

For our application, during training real future trajectories are used, while during inference they are generated with help of the IDM. This can be seen as introducing prior knowledge, e.g. regarding physical movement constraints as well as behavioural characteristics of rational drivers, into the model, and employing an inductive bias towards expected predictions. Consider a situation involving a breaking vehicle following the ego vehicle on the target lane: Initially, a lane change looks unfeasible, but due to the braking manoeuvre will become possible. Such calculations could be done implicitly by the unidirectional LSTM model from Section 5.3.2.1. Often though, inducing inductive bias, using prior knowledge and carefully choosing network architectures is beneficial for performance. Our bidirectional model queries a separate, specialized prediction component, explicitly making use of the prediction of future states, and, assuming such prediction is reliable, reducing complexity of the remaining problem for the LSTM network.

Intelligent Driver Model To obtain an online usable bidirectional LSTM network, a prediction component is required. Many possibilities for this exist, such as learned prediction algorithms. For simplicity and due to good experimental results we use the IDM though, which essentially is a set of equations describing a rational driver

in car-following scenarios. Core behavioural assumptions are the desire to keep safe distances to preceding vehicles, but otherwise accelerating to a certain, desired target velocity if such distances allow it. Let X be the vehicle being described by the IDM, and Y its preceding vehicle. x and s denote velocities, $s_X = x_Y - x_X - l_Y$ and $v_X = v'_X - v'_Y$. Then, the behaviour of X according to the IDM is fully described by the following differential equations:

$$\begin{aligned} \dot{x_X} &= \frac{dx_X}{dt} = v'_X \\ \dot{v'_X} &= \frac{dv'_X}{dt} = a(1 - (\frac{v'_X}{v_0})^\delta - (\frac{s^*(v'_X, v_X)}{s_X})^2) \\ s^*(v'_X, v_X) &= s_0 + v'_X T + \frac{v'_X v_X}{2\sqrt{ab}} \end{aligned} \tag{5.3}$$

An agent's behavior is further controlled by the following freely choosable parameters:

- v_0: desired velocity
- s_0: minimal distance between two vehicles
- T: desired temporal distance
- a: maximal acceleration
- b: comfortable braking behaviour
- δ: factor to control magnitude of used exponentiation.

We then employ the IDM to predict future positions and velocities for the vehicles of interest (PV, RV, $PFV_{L/R}$ and $PLV_{L/R}$, as well as ego), in order to feed such assumptions about future traffic scenes into the bidirectional LSTM network. For the ego vehicle as well as vehicles RV and $PFV_{L/R}$ prediction over multiple steps actually becomes an autoregressive problem, as their preceding vehicles are also being predicted by the IDM. In general, this can be beneficial due to more accurate predictions, or the opposite due to compounding errors. For the remaining vehicles, preceding vehicles are only included in the initial step and then modelled as constant-velocity objects. Modern Radar and LiDAR sensors are able to capture surrounding vehicles as well as potentially their trailing or preceding vehicles, in case these are close enough - thus giving us all the needed information (if vehicles are out of our sensor range, their influence in the IDM's prediction is negligible anyways).

The bidirectional LSTM model is also depicted in Figure 5.2, and expressed mathematically as:

$$\begin{aligned} \mathbf{e}_t &= \texttt{emb}(\mathbf{W}_{emb}, b_{emb}, [\mathbf{g}_t^{pv}; \mathbf{g}_t^{rv}; \mathbf{g}_t^{pfv}; \mathbf{g}_t^{plv}]) \\ (\mathbf{h}_t^F, \mathbf{c}_t^F) &= \texttt{LSTM}(\mathbf{e}_t, \mathbf{h}_{t-1}, \mathbf{c}_{t-1}) \\ (\mathbf{h}_t^B, \mathbf{c}_t^B) &= \texttt{LSTM}(\mathbf{e}_t, \mathbf{h}_{t+1}, \mathbf{c}_{t+1}) \\ \mathbf{o}_t &= \texttt{softmax}(\mathbf{W}_o[\mathbf{h}_t^F; \mathbf{h}_t^B] + \mathbf{b}_o) \end{aligned} \tag{5.4}$$

Notation is identical to the one used for the unidirectional LSTM, $\mathbf{g}_t$ describes the one-hot encoding of vehicle positions. During training, we reset the internal state of the backward LSTM every T_B steps. During inference, in each timestep t sequences of length T_B are processed: Input data $\mathbf{g}_{t'}$ with $t' \leq t$ are derived from observed vehicle positions, steps $\mathbf{g}_{t''}$ with $t'' > t$ from predictions given by the IDM. We use $T_B = 10$s. Selection of T_B potentially has a high influence on the algorithm's performance. We recommend setting T_B to as large values as possible, for which a reliable prediction can be expected.

5.4 Evaluation

For evaluation we use the previously introduced NGSIM dataset (compare Section 3.3.2).

SVM Baselines For a competitive comparison we consider the SVM model from Nie et al. [86] (*SVM*). The only difference in application is the used dataset, which is bigger and not as restricted in our case. For the action-based dataset we analogously to Nie et al. pick one negative sample from T_N as well as one positive sample from T_P for each occurring lane change. In order to not sample nearly identical data frames once as positive and once as negative, we require a minimum time gap of 1s between each such two samples (which corresponds to the *Ignore* time period for the LSTM models). For the automatic dataset we split the available data into tracks, with one track containing all data frames involving a specific car. Then we sample a certain number of positive and negative samples from each track. As input features for the SVM model d_{PV}, d_{RV}, d_{PFV}, d_{PLV}, v_{PV}, v_{RV}, v_{PFV} and v_{PLV} are used, i.e. distances and relative velocities to the respective vehicles. To analyze the effect of a predictive component, we extend the conventional SVM with such a method. In particular, we concatenate each input frame with frames 5 respectively 10s in the future. These are obtained using real future trajectories, serving as an upper bound of reachable performance (*SVM**). A Gaussian radial basis function is used in both SVM models, best hyperparameters are found by grid-search.

LSTM Models We compare the unidirectional LSTM model (*LSTM*) and the bidirectional LSTM variant (*Bi-LSTM*). To again analyze the influence of the prediction component and compare against theoretically reachable performance, we further deploy *Bi-LSTM**, for which the IDM prediction component is replaced by real future trajectories. A single layer of size 128 is used, and weight regularization with weighting factor 0.001 employed.

To obtain a non-deep-learning baseline, we further analyze the application of solely the IDM model for automatically labelled samples: The IDM is used for predicting future vehicle positions for the next three seconds, and a situation deemed suitable

when these do not fall below 1s, analogously to how automatic labels are defined (*IDM*).

Table 5.1 lists results of these experiments. We observe, that already the unidirectional LSTM variant outperforms both SVM variants for both labelling schemes. This proves, that LSTMs are better suited for understanding temporal problems as encountered here, and also that SVMs struggle to find temporal correlations when corresponding inputs are given (*SVM**). We further observe an improve in performance when employing the bidirectional LSTM variant with real future data (*Bi-LSTM**). When replacing real future trajectories with predicted ones, a small deterioration in performance is detected. This is to be expected, as prediction models, in particular our used IDM, are not perfect models of the future. An interesting course for future experiments would be to use the IDM also during training, to condition our models on expected errors. Ultimately, we decided against it, as this increases training time significantly. Using action-based labels, *Bi-LSTM* performs comparably to *LSTM*. On automatically labelled data though, performance increases, indicating the usefulness of bidirectional LSTMs, even in online applications. Possible reasons for this outcome are the length of available labelled data tracks when using an automatic labelling scheme, as well as the absence of questionable labels, as arising in the action-based scheme. *IDM* by itself performs worst, showing that indeed we need learned, data-driven approaches to solve this complex problem, and that it is the combination of deep learning models and prediction component that generate the good results of *Bi-LSTM*, and not the IDM alone.

Figure 5.3 visualizes three scenarios and their resulting predictions and ground truth labels when using the automatic labelling scheme. Scenario a) depicts a situation in which a large gap is present in the target lane. This is correctly predicted by all three algorithms, which all predict suitability of a lane change. In scenario b), a vehicle is approaching quickly from behind in the target lane, making a lane change unsafe. Models *LSTM* and *Bi-LSTM* correctly recognize this evolution, while *SVM* fails to do so, classifying the situation as suited for a lane change. In scenario c), also *LSTM* fails. While at $t = 0$ the situation looks suited for a lane change at first glance, 2 seconds later the ego vehicle will have reached its preceding vehicle on the target lane, and having done a lane change would most likely have resulted in a crash. The IDM very accurately predicts this happening, as shown in Figure 5.4, and seemingly this strong knowledge helps *Bi-LSTM* in outputting the right prediction.

Table 5.1: Results of all analyzed algorithms for both labelling schemes.

	IDM	SVM	SVM*	LSTM	Bi-LSTM*	Bi-LSTM
Action-Based	-	77.24%	78.62%	88.76%	92.59%	88.19%
Automatic	61.10%	80.70%	57.90%	83.08%	88.49%	87.03%

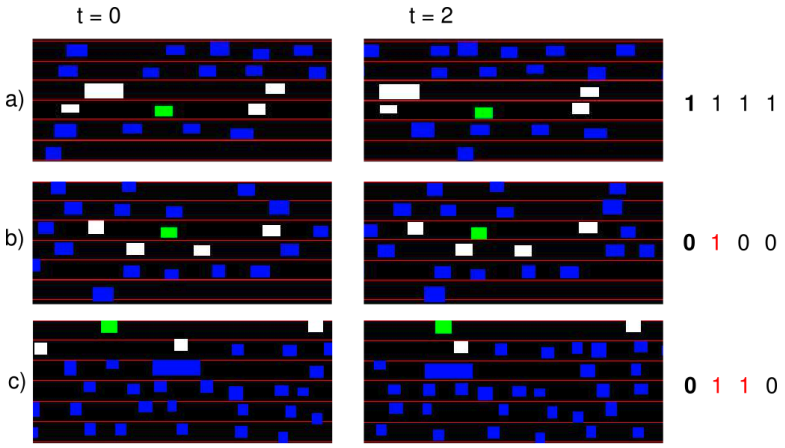


Figure 5.3: Visualizations of three different scenarios when using the automatic labelling scheme. The ego vehicle is drawn in green, the relevant neighbouring vehicles in white, thus marking the target lane. The prediction at timestep $t = 0$ is reported on the right: The ground truth label and the predicted labels from *SVM*, *LSTM* and *Bi-LSTM* are shown in this order. "1" describes a situation suited for a lane change, a "0" the opposite. For a better understanding, the resulting scene after 2 seconds is depicted in the right image (figure from [3]).

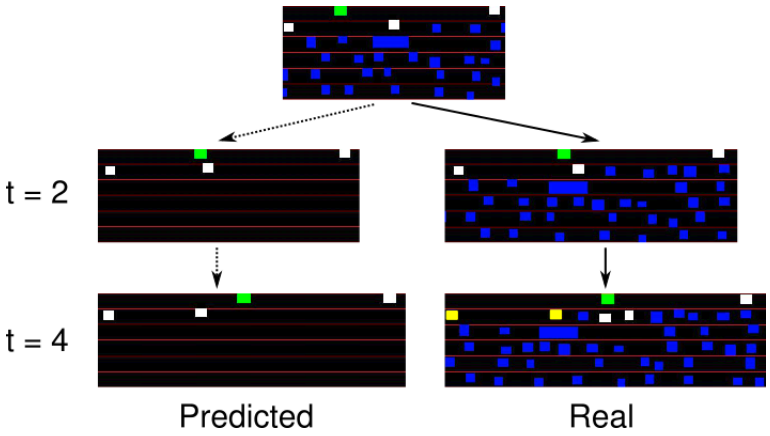


Figure 5.4: A closer analysis of situation c) from Fig. 5.3 to examine the prediction quality of the IDM: The predicted and actual scene is depicted 2 and 4 seconds after the initial image. One can observe a rather accurate prediction from the IDM. Note that the ego vehicle will have overtaken different vehicles, thus resulting in different neighbouring vehicles (white) - the "old" neighbouring vehicles being predicted are depicted in yellow (figure from [3]).

Figure 5.5 shows the temporal development of an exemplary scene, along with the corresponding predictions of *SVM* and *Bi-LSTM*. The scene begins with a rather congested traffic situation, and is unsuited for a lane change to the left. This is

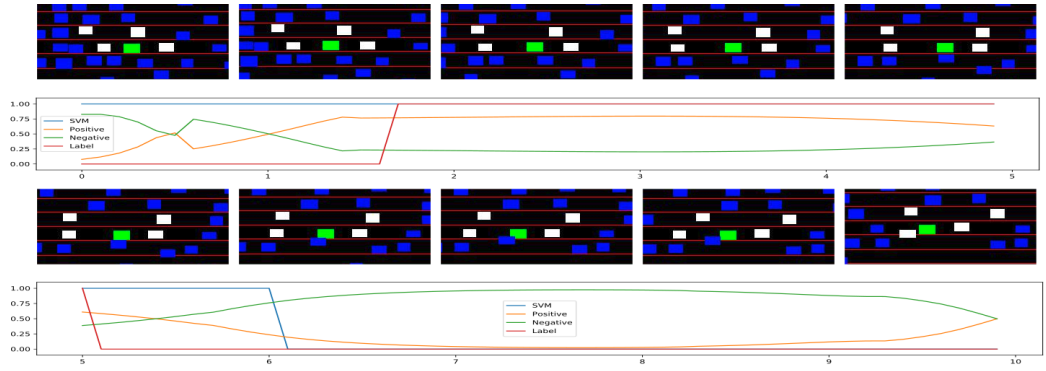


Figure 5.5: Visualization of the temporal development of a scene. Each image is recorded 1 second apart. The diagram below shows the correct label of each frame as well as prediction (probability for $\mathbf{o}_t = 1$) from *SVM* and *Bi-LSTM* (figure from [3]).

correctly predicted by *Bi-LSTM*, but *SVM* wrongly outputs suitability. As the cars move on, a suitable gap for a lane change forms, which is anticipated by *Bi-LSTM*, although somewhat early. Eventually, a vehicle is approaching quickly from behind, closing the gap. This is first recognized by *Bi-LSTM*, and around 1s later by *SVM*.

5.5 Conclusion

The ability to plan one's future actions is what defines an autonomous, intelligent agent. Data-driven approaches, such as RL and IL, show great promise but exhibit problematic characteristics for practical applications, such as difficulties to verify their behaviour. Still, deep learning most likely will be needed to solve the complex problem of fully autonomous driving. We therefore introduced a supporting deep learning layer to analyze traffic situations and support other planning algorithms in their decision-making.

In particular, we aimed at analyzing situations w.r.t. their suitability for lane changes, outputting binary decisions to indicate possible suitability. Therefor we employed an LSTM network, which is trained in supervised fashion on actual driving data as well as automatically labelled samples. We further extended this to a bidirectional LSTM, querying the IDM for explicit future state predictions, thus allowing the usage of bidirectional networks. Results indicate a good performance and an accurate scene classification.

In the future we would like to experiment with different prediction components as alternatives to the IDM. Good results should also be achieved by using learned approaches, such as training another LSTM network on predicting future developments of traffic scenes. This is an interesting direction of research for itself. Additionally, it would be helpful to improve the quality of assigned labels. Currently, labels are either derived from actual driving behaviour or obtained automatically by analyzing

future safety distances. When using actual driving behaviour, we use executed lane changes as positive samples. Negative samples are harder to find though. Using automatically generated labels yields large labelled datasets with good quality in many situations, but might be too passive and non-interactive compared to human behaviour. We examine possible improvements upon this in the next chapter.

6 Combining Prediction and Planning

In the previous chapters we addressed the tasks prediction and planning, particularly regarding lane change manoeuvres, separately. However, these are highly correlated: When predicting future lane changes of other traffic participants, this implicitly contains reasoning about drivers' intentions and feasibility of such manoeuvres - two topics, which are also essential when planning lane change manoeuvres for the ego vehicle.

In this chapter we thus study combinations of both tasks and our proposed models, showing interesting connections and improve model performances in some use cases. In particular, our contributions are:

- We combine our proposed lane change prediction approach from Chapter 3 and our situation assessment model from Chapter 5 via joint components, and train both tasks jointly on the same dataset.
- We aim at extracting intention and feasibility information from the lane change prediction model, and use this knowledge to improve labelling quality of proposed labelling methods for the situation assessment task from Chapter 5.

6.1 Introduction

Predicting and planning manoeuvres are inherently correlated: When predicting future driving manoeuvres of other traffic participants, good prediction algorithms should consider intentions of drivers (deciding whether there exist incentives for executing such manoeuvre) as well as practical limitations (is executing such a manoeuvre safe and feasible). These exact same steps also need to be considered when planning such manoeuvres for the ego vehicle. Further, incorporating assumptions and predictions of future driving states greatly helps planning algorithms, as planning is not a static problem, but needs to account for and react to dynamically changing situations in the future. Therefore, many well-working planning algorithms implicitly or explicitly consider prediction subtasks, and learn both tasks

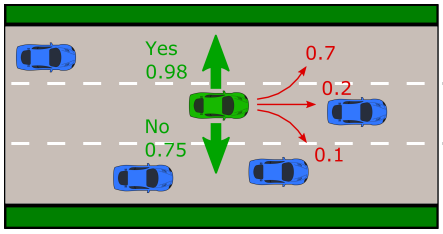


Figure 6.1: Visualization of the connection between the tasks planning and predicting lane changes. In green, outputs of the model assessing situations are shown: A lane change to the left is deemed feasible for the ego car with high probability, as the target gap in that lane is sufficient. A lane change to the right is rejected, due to other vehicles blocking this lane. Conversely, we can treat the green vehicle as target car in a lane change prediction task: The red arrows indicate predicted probabilities of the three manoeuvre classes *Left*, *Follow* and *Right*. One can observe the similarities between these tasks, which mostly differ in input structure and expected output. In particular, a good prediction model should consider surrounding vehicles as well, and not predict lane changes in unsuited situations (to the left in this example) (image from [2]).

jointly [37, 27]. Such joint learning of different, but related tasks, has proven to be good practise in general, and is done frequently to improve performance (consider e.g. jointly learning semantic segmentation and depth estimation [92]). When training one model for different tasks, this has several advantages: A shared backbone architecture is formed, which potentially better learns to capture and understand patterns in the data - since the model is forced to reason jointly about different tasks - before separate specialization layers output final results for all tasks. Additionally, with each task the number of labels (ideally) grows: Each label of every task constitutes a possibility for the model to learn from ground truth.

Here, we first propose a model to jointly predict lane changes and assess situations w.r.t. their suitability for lane changes. For this, we combine the prediction model from Chapter 3 with the situation assessment model from Chapter 5 over a joint component, which is responsible for analyzing and evaluating gap distances. Figure 6.1 visualizes the connections between these two tasks.

Next, we aim at improving the labelling quality for the situation assessment task from Chapter 5. Our lane change prediction model first is used to predict driving intent to avoid false negative labels generated by the automatic labelling scheme. Then, we employ the prediction model for analyzing feasible safety distances in order to avoid false negative labels in the automatic labelling scheme.

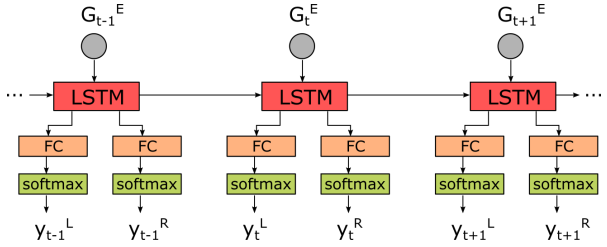


Figure 6.2: Depiction of the modified situation assessment model: Instead of embedding actual distances, temporal distances are used, i.e. feature group G^E. Features for both target lanes are processed by a single LSTM cell, and two independent fully connected layers (FC) followed by a **softmax** function return resulting predictions for both lanes simultaneously (image from [2]).

6.2 Combined Approaches for Predicting Lane Changes and Assessing Situations

We first describe the proposed joint model and then address the issue of labelling quality.

6.2.1 Joint Training

Goal of this section is combining our lane change prediction model from Chapter 3 and our situation assessment model for lane changes from Chapter 5. To better be able to use the latter, we use a slightly modified model, which is depicted in Figure 6.2 and expressed formally as follows:

$$\begin{aligned}(\mathbf{h}_t, \tilde{\mathbf{c}}_t) &= \texttt{LSTM}(G_t^E, \mathbf{h}_{t-1}, \tilde{\mathbf{c}}_{t-1}) \\ \mathbf{y}_t^L &= \texttt{softmax}(\mathbf{W_L}\mathbf{h}_t + \mathbf{b_L}) \\ \mathbf{y}_t^R &= \texttt{softmax}(\mathbf{W_R}\mathbf{h}_t + \mathbf{b_R})\end{aligned} \tag{6.1}$$

We remove the dependency of the model's input and prediction to the target lane. Now, all neighbouring vehicles (PV, RV, PFV_L, PFV_R, PLV_L, PLV_R) are used as input, and a prediction formed for both possible target lanes simultaneously, i.e. yielding two outputs indicating whether a lane change to the left and right is possible. In addition, we remove the one-hot encoding and subsequent embedding of vehicle distances, and simply use temporal distances dt_X to describe available gaps. Thus, input to this model exactly is feature group G_t^E (compare Section 3.3.1).

The lane change prediction model *LSTM-E* from Section 3.3.2 consists of three LSTM cells to process the feature groups G_t^Z, G_t^E and G_t^M. Purpose of the cell responsible for G_t^E is analyzing distances to surrounding vehicles and evaluating suitability of a lane change. It is thus nearby to combine models via this cell: We treat both tasks as a single, multi-task problem using a joint model, namely a combination of our lane change prediction model and the adapted situation assessment model, and joining these by using the same LSTM cell to process G_t^E. When using

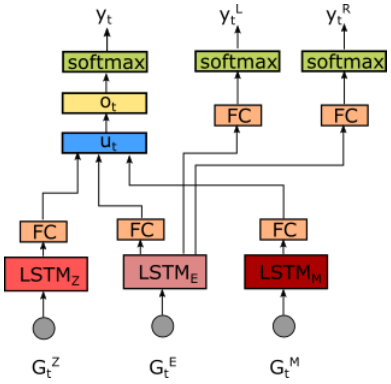


Figure 6.3: Visualization of our proposed architecture for jointly solving the tasks assessing situations and predicting lane changes: The architecture consists of the same models used for the single tasks, except cell $LSTM_E$ is now shared between both (figure from [2]).

the automatic labelling scheme for the situation assessment task, all frames of the dataset are labelled for both tasks, and we are able to train them jointly and simultaneously, reweighing loss functions to give both tasks similar importance. Figure 6.2 shows the resulting model. As stated before, such multi-task learning often improves performance when considering related tasks [37, 27, 26, 93].

6.2.2 Label Pruning

Defining labels is a crucial task for any supervised machine learning problem, and this is no different for the introduced situation assessment problem. However, defining proper labels here is challenging in some situations. Thus, in this section we set out to improve the labelling quality of this task by using our lane change prediction model.

Positive samples in the action-based labelling scheme are given by executed lane change manoeuvres, and are thus near optimal. Finding negative labels is a harder task: We follow existing works and label a fixed time period before a lane change as not suited for lane changes, following the assumption that before the execution of a manoeuvre drivers want to change lanes, but cannot. We further improve this approach by filtering out situations with no observable change from such assumed waiting periods to execution periods. Still, it is hard to decide whether drivers really do want to change lanes and cannot, or simply have no intention of doing so yet. We thus propose to further filter resulting labels by making use of our lane change prediction model: Part of its decision process includes determining drivers' intentions, making it a suitable candidate for intended task. Specifically, we compare predictions outputted from this model with labels generated from the action-based labelling scheme, removing negative labels if the corresponding prediction is *Follow*.

Conversely, labels generated by the automatic labelling scheme might be too "strict" and "passive": Core of such labelling scheme is the evaluation of future

safety distances, labelling situations as unsuited for lane changes when these fall below fixed thresholds. While this is useful in ruling out definitely unsuited situations, in the real world drivers often interact with each other to negotiate for gaps and exhibit a much more dynamic decision-making. We could try improving this behaviour by using our same proposed situation assessment model, possibly trained on action-based labels. However, we would like to avoid such recurrence, and instead again resort to our lane change prediction model: Good lane change prediction models need to consider intent as well as feasibility of a lane change, and thus should only predict a lane change in feasible situations. Therefore, negative labels are changed to positive ones, when prediction for a lane change exceeds a certain threshold for the corresponding frame.

6.3 Evaluation

6.3.1 Joint Training

In this section we describe our findings about jointly training the prediction and situation assessment task. Table 6.1 compares performance of the joint LSTM to that of a single-task lane change prediction model. For a fair comparison we compare against *E-LSTM*, as this is the model used in the joint architecture. Note though that such a joint model could, e.g., easily be extended by attention, as well. On a first glance, results of both models are comparable. However, *Joint-LSTM* solves two tasks simultaneously, using fewer parameters than two specialized models and exhibiting a faster inference time. Further, we observe a lower *Frequency* for *Joint-LSTM*, indicating, that incorporating knowledge and learned structures from the situation assessment task does indeed help in analyzing suitable lane change gaps, preventing false predictions when neighbouring lanes are blocked. This is confirmed in Table 6.2, *Joint-LSTM* performs better than *E-LSTM* in both scenarios *Left-/ Right-Blocked* (for an introduction of these scenarios compare Section 3.4.3.2).

Table 6.3 compares results of the joint architecture as well as single-task architecture for the situation assessment task. First note that a slightly higher accuracy is reported here for the standard LSTM model compared to Chapter 5. This is due to model differences stated in Section 6.2. For this task, joint training does not result in performance increase, but in fact we observe a worsening. Note again, that now two tasks are solved simultaneously with a combined model: The fact, that performance is still acceptable, proves the close relationship of tasks and the

Table 6.1: Comparison of single- and multi-task models for predicting lane changes on the NGSIM dataset.

Algorithm	Acc LC	Acc F	Miss LC	De LC	Ov LC	Pr LC	Re LC	Fr F	Pr F	Re F	Rank
E-LSTM	**0.689**	0.975	**0.006**	0.731	**0.596**	0.074	**0.696**	3.38	**0.958**	**0.93**	**14**
Joint-LSTM	0.684	**0.979**	0.008	**0.721**	0.582	**0.084**	0.692	**3.047**	0.95	**0.93**	16

Table 6.2: Summarized comparison of single- and multi-task models for predicting lane changes on the NGSIM dataset, split by scenario (L-/R-PV and L-/R-B denote the scenarios *Left-/Right-PV* and *Left-/Right-Blocked*. For each model, *Rank* is listed for each scenario.

Algorithm	Left	Follow	Right	L-PV	R-PV	L-B	R-B	L1
E-LSTM	11	7	**6**	9	**6**	7	7	7
Joint-LSTM	**6**	**4**	11	**7**	10	**5**	**5**	**4**

Table 6.3: Summarized comparison of single- and multi-task models for analyzing situations on the NGSIM dataset. For each, mean accuracy is specified.

	Automatic Labelling
LSTM	**0.894**
Joint-LSM	0.791

existence of common substructures. However, the decrease in performance indicates a hierarchical relationship between the two examined tasks: Assessing situations w.r.t. their suitability for lane changes is a helping subtask. Analyzing gaps and safety distances helps understanding when lane changes are feasible and when such manoeuvres are likely. Conversely, feedback from the more general prediction task is not applicable much in assessing situations.

6.3.2 Label Pruning

In this section we aim at improving labelling quality. As stated before, action-based labelling offers good labels during execution of lane change manoeuvres (positive labels), but struggles to define negative labels. Automatic labelling is able to filter out definitely unsuited situations, but struggles with finding positive labels due to static safety distances. Thus, these properties are somewhat contrary. As labelling quality is hard to define and measure, here we report the overlap of both labelling schemes and try enlarging this: When such overlap grows, both labelling schemes improve upon their weaknesses, and we believe this leads to a better generalization overall. In particular, we measure the fraction of matching labels and list these separately for positive and negative labels (w.r.t. the action-based labelling scheme). Column "Original" of Table 6.4 shows the initial overlap: Nearly 70% of negative labels match, while less than 30% of positive ones do. This means, that only 30% of frames labelled positively in the action-based labelling scheme are labelled accordingly in the automatic labelling scheme, indicating the passivity of this scheme, as well as unwillingness of humans to adhere to static safety distances.

Table 6.4: Resulting label correspondence of action-based and automatic labelling scheme on the NGSIM dataset, listed in %.

Label	Original	$t_1 = 0.9$	$t_1 = 0.95$	$t_1 = 0.99$
Positive	28.8	89.5	87.9	80.1
Negative	67.3	55.0	58.0	62.7

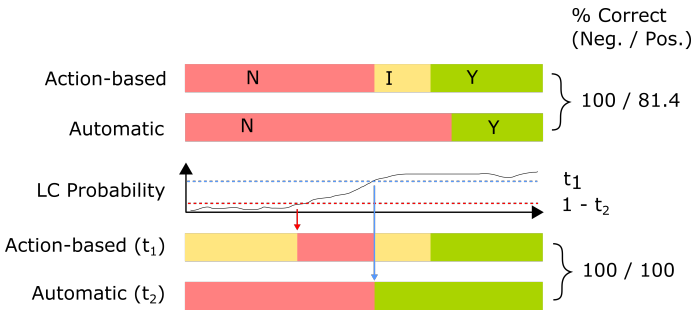


Figure 6.4: Depiction of the mentioned re-labelling procedures: The upper two blocks indicate original resulting labels of the action-based and automatic labelling scheme for a lane change sequence. In row 3, the predicted probability of such manoeuvre is depicted, and the thresholds t_1 and $1-t_2$ marked. When the predicted probability of a lane change exceeds t_1, automatic labels are changed to *Yes*. Conversely, when the predicted probability of *Follow* exceeds t_2 (which equals that the predicted probability for a lane change is less than $1-t_2$), action-based labels are changed from *No* to *Ignore*. On the right, the fraction of frames with matching label, w.r.t. the action-based labelling, is depicted (figure from [2]).

To improve labelling quality, we employ model *LSTM* from Chapter 3, using it "as-is". Labelling lane changes earlier as such to improve anticipation qualities did not yield any feasible results. We then

- change negative automatic labels to positive, when the predicted probability of a lane change exceeds t_1
- change negative action-based labels to *Ignore*, when the predicted probability of *Follow* exceeds t_2 (or, expressed differently, the probability of a lane change falls below $1-t_2$).

Figure 6.4 depicts the resulting changes stemming from these actions.

As is to be expected, decreasing t_1 increases the number of frames labelled as positive in the automatic labelling scheme, increasing the percentage of matching positive frames. However, simultaneously the number of negative frames decreases, decreasing the corresponding percentage. Still, the increase of positive frames outweighs the decrease in negative ones. Table 6.4 shows the effects of different t_1.

Consider $t_1 = 0.99$: We observe a decrease of around 7%, but an increase of roughly 180%. Still, one has to prioritize such tradeoff regarding potential applications and importances. Nevertheless, we find our proposed method to be useful in improving the quality of labels, in particular for generating more realistic, dynamic labels using the automatic labelling scheme.

Experiments with changing t_2 were not successful though: Changing t_2 yields identical values or slight decreases in matching negative labels. Possible explanations are difficulties of recognizing drivers' intents of following lanes, as our prediction model more is conditioned on detecting the opposite, as well as good performance of our filtering mechanisms: We already filter out implausibly looking labels by comparing differences between negative and positive frames, which solves a similar task as intended by this step.

6.4 Conclusion

Planning and prediction are inherently related tasks, and analyzing and exploiting this connection is an interesting research direction. In this chapter we specifically examined correlations between predicting lane changes and analyzing situations regarding their suitability for lane changes. These tasks were separately introduced in Chapter 3 and 5, respectively.

One line of research we followed in this chapter was introduction of a joint model: We combined previously proposed models for predicting lane changes and analyzing situations via a joint component responsible for measuring distances to neighbouring vehicles. This effectively combines both tasks into a multi-task problem, which is trained jointly. We show advantages of such reformulation of the problem, specifically a better understanding of for lane changes unsuited situations, as shown by better prediction performances. Further, we point out the hierarchical relation of the two tasks, resulting in a performance gain only for the prediction task, but not for the situation assessment one. Additionally we examined possible techniques for improving the labelling quality of the situation assessment task. We tried employing our lane change prediction model for filtering out false negative automatic labels as well as false negative action-based labels. Approach one was successful, the other was not.

In the future, we would like to deepen research and understanding about the connections of planning and prediction. Good planning algorithms should always account for future situations by either implicit or explicit prediction. Further, one could think about improving prediction algorithms by incorporating planning for all agents in a scene. Furthermore, employing prediction models for improving labelling quality of the situation assessment task seems like a fruitful area of further study. In particular, one could think of explicitly conditioning models on learning intentions, specifically for *Follow* manoeuvres, as they are most relevant in the second, unsuccessful re-labeling approach.

Part IV

Transfer Learning

7 Knowledge Transfer

Transfer learning describes the process of transferring knowledge and trained models from one domain to another - a skill mastered exceptionally well by humans: When faced with new environments or tasks, humans use their innate and learned understanding of the world to quickly master new challenges, often by just adhering to rudimentary explanations or few demonstrations. As of today, machines mainly lack such capabilities: Most machine learning models are trained for a specific task and domain, and by successful training reduce the model's bias on that task and domain, but increase it on most others. Finding ways of transferring knowledge without having to completely train models from scratch every time thus is an interesting line of research. This holds especially for the problem of autonomous driving. Here, many domains are possible: Sensor setups and software versions might change, vehicles need to navigate safely through all weather conditions, and autonomous vehicles should be available all over the world, ranging from rural areas to densely populated cities. Especially in the field of autonomous driving, safety and verifiability also play a crucial role: When transferring knowledge and models, ideally we would like to transfer safety guarantees, as well.

Therefore, we here propose a general framework to address shifting domains. As first step a transformation matrix to convert between samples from different domains is learned, and the model then fine-tuned on the new domain. Calculation of such an explicit transformation matrix here is beneficial, as this yields interpretable and verifiable transformations, and allows the induction of prior domain knowledge.

In particular, our contributions are as follows:

- We propose a general framework for domain adaptation, making use of a novel correspondence loss for learning inter-domain transformations.

- We apply our framework to a diverse set of problems, such as image classification and lane change prediction (Chapter 3), and compare against several other baseline methods.

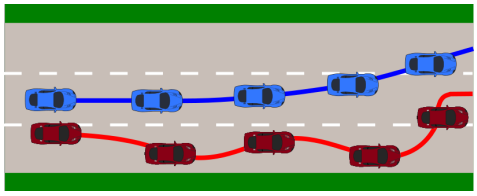


Figure 7.1: Side-by-side comparison of two lane changes in different domains. While the blue car executes its lane change smoothly, the red one exhibits a noisy driving style, most likely causing many false predictions in models not exposed to this (image from [6]).

7.1 Introduction

Particularly in the field of autonomous driving, vehicles and models need to be able to function in multiple domains and be able to adjust to possible changes. Such domain shifts could e.g. be induced by changes in sensor setups or software versions: A LiDAR sensor might be replaced by one of a different manufacturer, or a combination of Radar and camera sensors might be deemed sufficient even. In any case, layers above the perception layer should not be influenced by such design choice, a reliable and accurate detection of objects obviously still is required. Another reason for variety in domains arises from environmental conditions: Autonomous vehicles need to navigate safely through all weather conditions, such as sunny weather, rain and snow. Yet another difference stems from geographical properties and country-specific driving styles: Rural areas and cities all over the world look different, further traffic rules differ as well as driving styles (see Figure 7.1 for a visualization). Training and storing full models for each possible domain hardly seems feasible. In addition, this discards many similarities between domains and possibly worsens performance. Ideally, we would like to only learn domain-specific differences - and possibly even transfer proven safety guarantees for existing algorithms. Furthermore, often a skewed data distribution will be present, as automobile manufacturers predominantly test close to their offices. This completely discards the possibility of training full deep models for each domain, as data is not sufficient.

Many works in the field of domain adaptation and transfer learning focus on image tasks, using e.g. generative models to fantasize samples from different domains [64]. Although these yield fascinating results, such methods are non-interpretable and non-ideal for data different to images. Here, we propose a general framework for domain adaptation, specifically suited for numerical, high-level features, and designed to yield interpretable transformations. A *Converter* is prepended before the actual neural network, aim of which is calculating a transformation matrix, with which features are transformed to more resemble the original domain. This component can be pre-trained with existing domain knowledge (we could, for example, know to which new position a camera was shifted), and then further be refined by using *corresponding samples* from both domains (consider the problem of predicting lane changes: we can easily identify lane changes in two domains, and overlap these

to get corresponding pairs usable for training). Eventually, the resulting model is fine-tuned in the new domain to further improve performance.

General goal of this chapter is developing ways for transferring previously established models, particularly our lane change prediction model from Chapter 3, into new domains. For this, we analyze a transfer of lane changes from the fleet dataset to the NGSIM dataset, as introduced in Section 3.4.1. Nevertheless, to show general applicability of our proposed framework, we also examine the problem of image classification using the MNIST dataset [94].

7.2 Related Work

We begin this section with a brief overview over related techniques, and then turn our focus towards methods more related to ours. In existing literature, the terms "transfer learning" and "domain adaptation" are sometimes used interchangeably and inconsistently. Here, we follow the notation of [22], denoting with transfer learning the complex pool of methods for transferring knowledge between domains and tasks, and with domain adaptation a subbranch: Domain adaptation denotes the subfield concerned with modifying probability distributions of data between domains to make them more similar. As our methods make use of a learned transformation, it can be mostly counted in this field.

One of the simplest but nevertheless successful transfer learning methods is fine-tuning of existing networks. This method can be applied when aiming to deploy models in different domains or for different tasks than they were trained for. For this, most layers of the pre-trained network are frozen, while only layers close to the output are re-trained on the new domain or task. This often outperforms training the full model from scratch, due to the encoding of useful prior knowledge in the network's weights and restricting the space of free parameters, lessening the need for data, even for different tasks [95, 96]. Re-using internally stored knowledge is a fundamental principle of knowledge distillation [97], particularly self-knowledge distillation [87]. In this, outputs of the `softmax` layer are used as soft targets during training, thereby providing the model with more information, such as relations and similarities between classes. Generative models, such as GANs, are another commonly used method for addressing transfer learning problems. Using these data resembling the desired distribution can be generated, which can for example be used for the creation of synthetic training samples in new domains and modifying input properties, such as style, to better resemble another domain. These methods are applied frequently and with great success in fields concerning image inputs [98, 99, 100]. Other interesting methods, which can be counted towards the field of transfer learning, are the concepts of zero-, one- and few-shot learning. These aim at recognizing and classifying new inputs based on few examples, and thus effectively need to deal with arising domain shift and few annotated samples [25, 101]. Multi-task learning specializes on solving different tasks jointly, often using a common base architecture, and thus enabling information exchange between tasks [102].

The previously mentioned concept of knowledge distillation was extended to transfer learning applications, as well, and serves as baseline in Section 7.4.3. It was originally introduced by Hinton et al., fundamental idea is using a (possibly) smaller student network which is trained on the output of a teacher network [97]. Yim et al. extend this concept with a possible focus on transfer learning, particularly considering applications for network compression [103]. Their contribution consists of matching feature activations over different layers and domains, which is commonly used in other works, as well [104, 105]. They compare their approach to fine-tuning the last layers of the full model, nearly achieving similar performance. As fine-tuning also is one of our baselines and we compare favourably, a relative comparison can be deduced.

Domain adaptation is commonly employed in the area of image processing, with sample applications being style transfer and creating synthetic training samples in new domains [99, 100]. Zhu et al. make use of two separate GANs for translating images from domain A to domain B and back, achieving fascinating results without requiring corresponding samples [64]. Generator 1 learns a mapping from domain A to B, while generator 2 learns a mapping from domain B to A. The corresponding discriminators are trained to distinguish generated samples from actual samples of that domain. By concatenating these networks both ways and employing a cycle consistency loss between reconstructed sample in the original domain and original sample, the model implicitly learns to generate corresponding images, and to only transfer styles and similar, but keep structure and content. In the field of autonomous driving, we often have (loosely) coupled training samples, such as lane changes recorded in different domains. Thus, our method makes use of this, and we calculate an explicit transformation matrix $\mathbf{T}$. Using such explicit transformation simplifies understanding and interpretability of our model, and further is closer aligned to the general goal, namely transforming samples between domains instead of hallucinating plausible ones: When, e.g., converting a lane change from domain A to B, we do not just want any lane change of domain B as output, but the "exact" correspondence in the opposite domain. Additionally, as $\mathbf{T}$ nearly always is invertible, we obtain the cyclic conversion without additional parameters or inverse networks. Isola et al. use similar principles as [64], but only employ a single GAN, training it with exact correspondence pairs (x, y) to learn a mapping from domain A to B [106]. Somewhat similar to us, Li et al. apply linear transformation matrices for transferring image styles [107]. In comparison, we use a different loss: Whereas they calculate style losses at different layers in the network, we introduce a correspondence loss in the actual domains. This, again, simplifies understanding and verification of learned transformations, and allows usage independent of the number of layers, for example in RNNs. Further, their work is only focused on style transfer, while we develop a transfer learning framework. Although CNNs are undisputed state-of-the-art methods for basically all image-related tasks, surprisingly, they show little understanding of spatial information, and can e.g. be fooled to wrongly detect objects in shuffled images, or fail to recognize transformed images. Problem one arises from the commonly used max-pooling layers: These guarantee

desired spatial invariance (it is irrelevant where in the image we detect an object), but simultaneously discard all information of spatial structures. Due to this, a CNN still recognizes a shuffled collection of, e.g., two eyes, a nose and a mouth as a face, although the image merely contains all characteristic features of a face, but not the desired object itself. To combat this, Sabour et al. introduce capsule networks, which encode spatial information through different capsules and requiring images to be consistent with these [108]. On the other hand, performance of CNNs can suffer when images are transformed geometrically, e.g. rotated or distorted. Therefore, Jaderberg et al. introduce Spatial Transformer Networks (STNs). These are prepended before other layers, with the purpose of normalizing inputs. In particular, STNs calculate geometric transformations with which they transform corresponding inputs, s.t. e.g. relevant objects are always resized to a particular size and rotated to a certain angle. Thus, their goal is improving classification performance and not transferring models to new domains; further, they lack the correspondence loss used by us and instead only use a classification loss. Still, their idea is similar, and one can think of our method as a way of extending STNs to numerical data. We show the benefits of our loss in Section 7.4.3, and extend the concept to time series data.

Transformation matrices are also calculated by Duan et al. and used to project data of different domains into a common subspace, which is subsequently processed by SVMs [109]. Paaßen et al. also combine a common optimization method with explicit transformation matrices $\mathbf{H}$, although their approach can be generalized to other models [110]. In comparison to their global transformation matrix $\mathbf{H}$, we calculate a point-wise transformation $\mathbf{T}_x$, which allows greater flexibility. Additionally, similar to [111] such transformation is only considered implicitly by analyzing influences to existing loss functions. We compare against this method in Section 7.4.3 and refer to this for more details. Aswolinskyi et al. extend this concept to unsupervised applications [112]. A predictive model to generate artificial data for domain B is learned, and with this a transformation between domains learned in unsupervised fashion. Sun et al. introduce, in their own words, a "frustratingly simple domain adaptation" method, dubbed CORAL [23]. This extends the common standardization preprocessing step to include calculations of second-order statistics, in order to minimize differences in data distributions between domains.

7.3 A Framework for Domain Adaptation

We begin this section by introducing our definition of sample "correspondence" and then describe our proposed domain adaptation framework.

7.3.1 Corresponding Samples

For calculating transformation matrices between domains, we require - to a certain degree - corresponding samples of different domains. In the work of Zhu et al., no aligned samples of both domains are needed, cyclic conversions are learned by training discriminators to distinguish real from generated samples, and randomly

drawn samples are used for training [64]. Long et al. fine-tune classification models for the task of semantic segmentation [113]. One can now interpret domain A as the original image space, and domain B as corresponding semantic maps. This describes another extreme of correspondence, in contrast to the previously mentioned principle of no correspondence: Here, a (nearly) perfect (depending on the labelling quality) mapping f between every image x and corresponding semantic map x' exists, s.t. $f(x) = x'$.

For our use cases, we do not have nor require such an exact mapping, but instead relax this to $f(x) \approx x'$: Instead, for each sample x in the original domain we assume the existence of n approximately corresponding samples x' in the new domain, and thus obtain n pairs $(x, x'_1), \ldots, (x, x'_n)$. By training our model with such pairs, we expect it to learn an average mapping, in hopes that this provides a good fit. Consider the example of mapping images to semantic maps: Our relaxation equals pairing images not with semantic maps of exactly those images, but instead with n semantic maps of slightly different scenes, e.g. differing in the number of parked cars on the street.

Here, we conduct experiments on image data (i.e. the MNIST dataset) and sequential data (i.e. predicting lane changes). We compare our three introduced degrees of correspondences (no correspondence, approximate correspondence, exact correspondence) using these examples, and further explain our used approximate correspondence mappings: For the image classification task, domain A consists of original MNIST images, while domain B consists of images rotated by R. For training the proposed model in [64], random, independent samples from domain A and B are drawn, and with help of the cycle consistency loss a mapping between domains is learned. In an exact correspondence setting, training pairs would consist of original image x and its exact correspondence x' in domain B, i.e. the result of applying R to x. In our setting requiring approximate correspondences, this relaxes to pairing an image from domain A with a rotated image from domain B s.t. their labels match - i.e, the images still show the same number, but otherwise can differ in appearance. For aligning lane changes of different domains, we also conduct such alignment based on label. In particular, sequences containing only *Follow* frames are paired with sequences of identical label. Sequences containing lane changes are aligned s.t. the exact moment of lane change matches as closely as possible. Further extensions are possible, considering also other factors, such as searching for equally crowded scenes. Based on experience regarding good performance as well as available data in the wild, for all experiments we use $n = 5$, i.e. five correspondence pairs.

We would like to point out another interesting relation of our proposed definition of correspondence, namely to triplet loss [114]. Triplet loss is e.g. employed in few-shot learning: The general idea is not to directly learn classification boundaries, but instead embeddings of input samples into latent spaces, with aims of placing related samples nearby in such space. This way, new classes can still be classified by comparing to a single base representation of such class, or existing objects and classes re-identified accurately. Triplet loss is one method of training such networks.

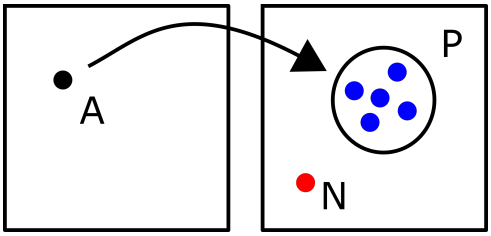


Figure 7.2: Comparison of our used correspondence mapping f to triplet loss. When using triplet loss, an anchor point A is paired with a positive sample P and a negative one (N). In our interpretation, f, and thus the resulting model, is a generative way of converting A to another domain, ideally resulting in the mean of given correspondence points (image from [6]).

Formally, for each point a (also called anchor point) a positive sample p and negative sample n are used to define the loss $L = max(0, d(a,p) - d(a,n) + m)$, in which m is a choosable minimal desired margin. This forces latent distances to be small for a and p, while it penalizes distances smaller than m for a and n. Our used correspondence pairs $(x, x'_1), \ldots, (x, x'_n)$ can be understood as pairs of anchor point and positive samples, while no negative points are given. With this interpretation, models trained with triplet loss can be seen as discriminative models for identifying related data samples, while our framework offers a generative method for generating such positive samples. This connection is depicted in Figure 7.2.

7.3.2 Model

Our proposed framework is suited for application in the following application scenario: There exists a source domain A and target domain B, and we intend to solve the same task on both domains. Data for A is sufficient, and we have trained a (complex) model $\mathbf{M}$ on A for the desired task. Data for domain B is rare: We do have labelled examples, but less than for domain A. In this setting, as previously mentioned, it is nor desired nor effective to train full, complex models on domain B: The amount of available data is too little compared to the number of tunable model parameters, resulting in bad generalization performance (as also observed by [103]), and it is not feasible to train and store full models for each possible domain B. Instead, it is desirable to leverage as much prior and stored knowledge from domain A as possible. Therefore, we here assume existence of a trained model on domain A, and explore ways of adapting this to a new domain. To simulate the sparsity of data and further analyze relation of transfer performance and amount of data available, we artificially limit domain B to contain at most b samples for varying b.

Core of proposed framework is usage of a network we name *Converter* ($\mathbf{C}$). When applying $\mathbf{M}$ in domain B, $\mathbf{C}$ is prepended before $\mathbf{M}$ with aims of transforming input data to appear more similar to data from A, thus making the task of $\mathbf{M}$ easier. $\mathbf{C}$ can be pre-trained with prior knowledge of domain differences, and then is further refined

by making use of the established correspondence fairs. Eventually, the combination of $\mathbf{C}$ and $\mathbf{M}$ is fine-tuned on domain B to further improve performance.

Let us introduce this formally: Assume samples of domains A and B have dimensionality n. Let $x \in \mathbb{R}^d$, $d \in \mathbb{N}$, be a sample of domain B, and let $\mathbf{x}$ be its homogeneous representation, i.e. $\mathbf{x} = \begin{pmatrix} x & 1 \end{pmatrix}^\top$. Then we define $\mathbf{C}$ to be the mapping:

$$\mathbf{C} : B \to \mathbb{R}^{(d+1)\times(d+1)},\ x \mapsto \mathbf{T}_x \tag{7.1}$$

s.t.

$$\mathbf{T}_x \mathbf{x} = \mathbf{x}' \tag{7.2}$$

Here, $\mathbf{x}'$ denotes a corresponding point of x in domain A (i.e. (x, x') is one of the established corresponding samples). Thus, goal of $\mathbf{C}$ is transforming samples of domain B into corresponding samples of domain A. Denote with $\mathbf{L}$ the last l layers of $\mathbf{M}$ w.r.t. to the output. In total, training of our proposed framework consists of three steps, each of which is optional (i.e., pre-training could be left out or no corresponding samples used):

1. **Pre-training**: Induce prior knowledge into the model by pre-training $\mathbf{C}$ with expected transformation matrices $\tilde{\mathbf{T}}_x$. For this, we train $\mathbf{C}$ with samples $x \in B$ while minimizing the element-wise L2 loss $\mathfrak{L}_P(x)$ between generated transformation matrix and expected one:

   $$\mathfrak{L}_P(x) = |\mathbf{C}(x) - \tilde{\mathbf{T}}_x|_F \tag{7.3}$$

   The element-wise L2 loss between two matrices equals the Frobenius norm in matrix space.

2. **Correspondence training**: Train $\mathbf{C}$ with established correspondence pairs $(x, x'_1), \ldots (x, x'_n)$ for $x \in B$. As loss function we are free to use any loss functions applicable in the domain spaces, such as the L2-loss between converted and actual samples:

   $$\mathfrak{L}_C(x) = \frac{1}{n} \sum_{i=1}^{n} |\mathbf{C}(x)\mathbf{x} - \mathbf{x}'_\mathbf{i}|_2 \tag{7.4}$$

   Here, $\mathbf{x}'_\mathbf{i}$ again is the homogeneous version of x'_i.

3. **Fine-tuning**: Fine-tune the combined model on domain B. For this, we introduce two train modes: In mode 0, we only retrain $\mathbf{L}$ based on the original loss used for the respective task (e.g. cross-entropy for classification). In mode 1, we retrain $\mathbf{L}$ and $\mathbf{C}$ by accumulating task loss and correspondence loss $\mathfrak{L}_C$.

Step 1 enables incorporation of existing domain knowledge into the calculation of $\mathbf{T}$. Consider e.g. the scenarios of changing from Radar to LiDAR sensors or using cameras whose position on the vehicle was modified. For both, we could devise transformations to map from one domain to another (e.g. estimate a transformation matrix from original to new camera positions). When such information is not

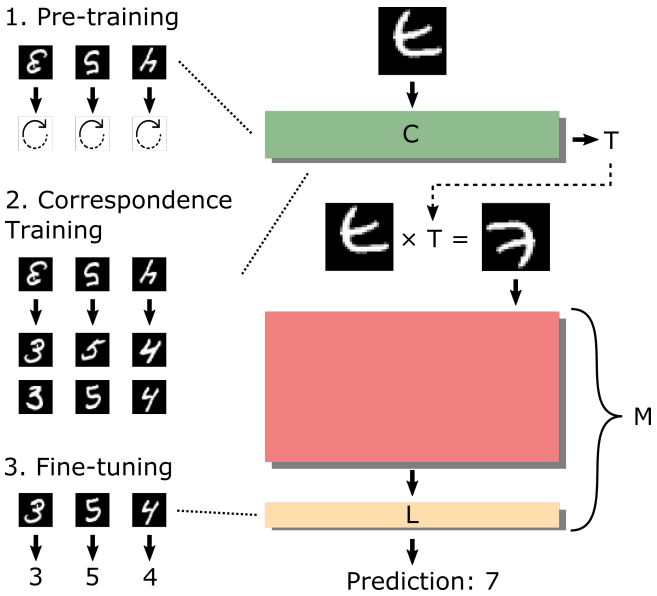


Figure 7.3: Visualization of our proposed framework, showing a sample application of adapting a model trained on standard MNIST images to rotated images. Steps 1 and 2 concern training the Converter **C**: First, we pre-train it with an (expected) rotation, i.e. requiring the outputted transformation matrix **T** to equal a rotation matrix. Then, **C** is trained using correspondence pairs for each sample (here, $n = 2$). In Step 3, **L** is fine-tuned on the new dataset, i.e. weights of the last layers of **M** are adjusted while the complex part drawn in red is frozen (image from [6]).

available, either because it is unknown or too complex to calculate manually, it is recommend to use $\tilde{\mathbf{T}} = \mathbf{I}^{d+1}$ (i.e. an appropriately dimensioned identity matrix), similar to [111]. This way, performance of **M** combined with **C** is at least as good as that of **M** alone, with the potential of improvement by choosing suitable transformations. In step 2, we make use of the loose correspondences between domains to improve and learn such a mapping in data-driven fashion. Each pair provides the model with information of possible transformations. However, these are, as mentioned, no perfect correspondences, but by providing n correspondence points for each point from B the model learns to approximate a good, average mapping. By now, hopefully the resulting sample after this step $\mathbf{x}'$, i.e. $\mathbf{T}_x\mathbf{x}$, resembles data from domain A, and **M** already exhibits good performance. Still, calculated transformations will be noisy and not perfect. Therefore, in step 3 we fine-tune the resulting model on the new domain to further improve performance and allow it to learn responses to domain shift. An overview over the model is displayed in Figure 7.3.

In general, **C** and obviously **L** posses less parameters than the full model **M**. This is important, as now training **C** and **L** is feasible with limited data, and so is storing individual components for each domain (while **M** only needs to be trained and stored once). Introduction of **C** makes the formulation of the domain adaptation

task explicit and independent of the model itself, a technique often yielding good results empirically.

7.3.3 Simplifying Assumptions

$\mathbf{T}$ represents a general transformation matrix in the domain space, and thus can model arbitrary transformations expressible in homogeneous coordinates. However, it is possible to restrict the space of possible transformations, thus making the model less expressive, but allowing easier learning. This is especially recommended if one knows possible transformations, e.g. can restrict possible transformations to rotations. In line with this, in the following we describe constituted assumptions and simplifications when dealing with images and sequential data. Note that our framework can be applied without them, working in the full vector spaces of images and sequences, respectively, but that these assumptions help in reducing complexity and improve ease of training. In addition, further restricting the space of possible transformations - as described before - is of course possible, as well.

7.3.3.1 Image Data

Let x be an image of width w and height h, thus $x \in \mathbb{R}^{w \times h}$. As image transformations are often of geometric nature (e.g. rotations, rectifying projections, ...), we employ such global image transformations instead of transforming pixels individually. Thus, instead of operating in the full image space $\mathbb{R}^{w \times h}$, we restrict examined transformations to the space $\mathbb{R}^{3 \times 3}$, and using this represent homogeneous 2D-transformations which are applied to all points. For calculating and applying such transformations, we utilize the module introduced in STNs.

7.3.3.2 Sequential Data

In a similar manner, we reduce transformations of full sequences to frame-based transformations. Assume, used sequences are of length l and feature size f, thus $x \in \mathbb{R}^{f \times l}$. Instead of considering transformations in this space, we transform each frame individually, s.t. $\mathbf{T} \in \mathbb{R}^{(f+1) \times (f+1)}$. To still model and acknowledge the temporal context, $\mathbf{C}$ is an LSTM network. This network processes full input sequences, and outputs a sequence of transformation matrices. Using our standard notation for LSTM cells and denoting with $\mathbf{h}_t$ the respective hidden state at timestep t, the calculation of the corresponding transformation matrix $\mathbf{T}$ is given by

$$\mathbf{T}_x^t = \texttt{softmax}(\mathbf{W} \cdot \mathbf{h}_t + \mathbf{b}). \tag{7.5}$$

7.4 Evaluation

In this section we introduce used datasets, baseline methods and results of experiments comparing these against our proposed framework.

7.4.1 Datasets

As stated, we showcase usage of our framework on image and sequential data. For the latter, goal is enabling model transfer between domains for our lane change prediction models from Chapter 3. We introduce and motivate this by first analyzing simulated lane changes.

MNIST MNIST is a collection of hand-written digits between 0 and 9, stored as black and white images of size 28×28 [94]. This original dataset constitutes domain A. Domain B consists of images rotated by 180°. As mentioned in Section 7.3.1, corresponding pairs of samples from domain A and B are formed according to labels, i.e. images of A are aligned with random images of B with equal label.

Toy Sequences We believe, that simple, simulated datasets offer a good first insight into the mode of operation of models and clear visualizations of how these perform. Thus, before addressing the intended problem of predicting "real" lane changes, we propose examination of a toy dataset: This consists of synthetically generated lane changes, only represented by the vehicle's distance to the lane's left lane boundary (0 denoting a position at the left boundary, 1 a position at the right boundary). Domain A consists of such lane changes in "clean" fashion, meaning drivers smoothly and without oscillations execute lane change manoeuvres and otherwise follow their intended lanes. Domain B contains perturbations of these trajectories in the form of added noise to the lateral positions. Such domain differences model a real application scenario we are interested in, namely changing domains from a dataset consisting of smooth trajectories to more noisy ones (one can think of different country-, driver- and road-specific driving styles). The objective of this problem is identical to the lane change prediction problem described in the next section: Each frame is given one of the labels *Left*, *Follow* or *Right*, which needs to be predicted by the applied models. A time period of 3s before a lane change is labelled with the corresponding label, while all other frames are labelled *Follow*.

We expect more false predictions when applying a model trained on domain A to domain B, due to the higher amount of noise and oscillations in trajectories. Indeed, this is the case, as Section 7.4.3 proves. Note that domain gaps are larger when working with raw sensor data, e.g. in original image or point cloud spaces. Still, our results indicate a significant drop in performance when changing domains using this abstract feature representation - indicating that transfer learning is of need also in these scenarios.

Lane Change Prediction For the problem of predicting lane changes we closely follow our work from Chapter 3. Domain A is the fleet dataset, while domain B is the NGSIM dataset - both are introduced in Section 3.4.1. We observe several differences between these datasets, starting with the location of recording: BMW's fleet dataset was predominantly recorded in Germany, while NGSIM is recorded in the United States. Further, used sensor setups to capture scenes are different: While

static cameras record traffic and detect vehicles on fixed highway segments, the fleet dataset was recorded from moving vehicles using a combination of different sensors. Although these differences are, as mentioned, diminished due to the used high-level representation, still domain-specific characteristics arise. Additionally, data in the NGSIM dataset was mainly recorded during rush-hour and in segments with on- and off-ramps, resulting in different driving behaviour than observed in the fleet dataset, which was recorded over different times and highway segments. This again leads to the assumption, that models trained on A react too sensitively to data from B.

7.4.2 Metrics

For the experiments on the MNIST dataset, we use prediction accuracy as metric.

For evaluating (simulated) lane changes, we rely on the metrics introduced in Section 3.4.2, and for simplicity and a better overview restrict our focus to *Frequency*, *Miss* and *Delay*. In our opinion these describe most relevant properties of lane change prediction models. We further accumulate these to allow quick comparisons between models: Model $\mathbf{M}$ trained on domain A and tested on B serves as baseline. For all analyzed models, we measure and average percental differences regarding these three metrics compared to the baseline. While doing so, we weigh metric *Frequency* by the percental amount of frames labelled *Follow*, and *Miss* and *Delay* by the amount of lane change frames, accordingly. This score calculation acknowledges the amount of time drivers experience the respective manoeuvres for, and additionally notes the high importance of *Frequency* on felt driving comfort.

7.4.3 Results

We first introduce implemented baseline methods against which we compare our framework, before showing results of all experiments. While showing results for problems separately, we additionally mention custom modifications of the introduced baseline methods, and particularly explain the used base model $\mathbf{M}$, which is to be adopted to domain B.

7.4.3.1 Baseline Methods

- *Fine-tuning*: We employ model $\mathbf{M}$, which was trained on A, and fine-tune layers $\mathbf{L}$ on B. As [103] also compare against this and fare slightly worse, a relative comparison can be deduced.

- *CORAL* [23]: We follow the methodology of the CORAL framework, standardize data and additionally also minimize covariance distances between data from domains A and B. This input data is again fed to model $\mathbf{M}$, and we also fine-tune layers $\mathbf{L}$.

- *pix2pix* [106]: We use the pix2pix framework as representative of an implicit domain adaption method, in contrast to our explicit one: A GAN is used to

transform data from domain A to B, which directly outputs data samples from domain B, instead of outputting transformation matrices.

- *Imp*: To further analyze the influences and differences of explicit vs. implicit domain adaptation models, this baseline equals our proposed framework but directly generates samples in the corresponding domain, instead of making use of a transformation matrix. In particular, outputs of $\mathbf{C}$ are now directly samples in domain B, instead of transformation matrices. Thus, training step 1 of our framework is not possible and consequently left out.

- *Mode 2*: We further compare against [110] and STNs [111], and would like to point out how these models can be represented by means of our framework: Both make use of a transformation matrix ([110] originally use a global one, we extend this to allowing sample-specific ones), and also fine-tune pre-trained models by modifying $\mathbf{C}$ and $\mathbf{L}$. Differences to our method are the abundance of training steps 1 and 2 (pre-training and further refining the Converter), and the lack of correspondence loss in general. Otherwise, they are similar to our framework with train mode 1, except for training only the original task (classification) loss is considered. We modify our framework accordingly, and denote the resulting method with *Mode 2*. In addition, we add step 1 with identical pre-training targets for a fairer comparison.

- *Mode 0*: To further investigate the usefulness of our proposed correspondence loss, we introduce baseline *Mode 0*: This is our framework without training step 2, trained with train mode 0, thus totally excluding corresponding samples and the correspondence loss. Such correspondence loss is one of our contributions and one essential difference to [110] and [111]. Both baselines *Mode 0* and *Mode 2*, employing no such loss, thus help answer the question, whether such a loss is beneficial, or similar results can be achieved by training / fine-tuning a model equipped with a converter and training it in usual ways.

For further clarification, we repeat / rephrase essential differences between our full proposed framework and baselines *Mode 0* and *2*: Our full framework consists of the three steps pre-training, correspondence training and fine-tuning. During fine-tuning, we distinguish between train mode 0 and 1: With train mode 0, only $\mathbf{L}$ is retrained, employing solely the original task loss. When using train mode 1, weights of both $\mathbf{L}$ and $\mathbf{C}$ are modified, making use of original task and correspondence loss. The introduced baselines *Mode* 0 and *2* lack the correspondence training step - for the respective applications, no existence or usage of any correspondence pairs is assumed. *Mode 2* further differs from our proposed framework by only considering the original task loss, albeit retraining $\mathbf{L}$ and $\mathbf{C}$. This method thus exactly mirrors training of e.g. STNs. *Mode 0*, is similar, but just $\mathbf{L}$ is retrained.

7.4.3.2 Rotated MNIST

Used base model **M** for this task is a simple CNN consisting of 3 convolution and max-pooling layers, followed by a fully connected layer for classification. Layer **L** equals this last layer. **C** consists of 2 convolution layers followed by one fully connected layer.

Table 7.1 shows quantitative results on the dataset, ordered by b, the maximal amount of available data in B. Naturally, with growing b performance of all models increases. As general baselines, the performance of testing model **M**, which was trained on A, on B is reported (A on B), as well as training all of **M** on the full training set of domain B (to give an understanding of theoretically achievable performance - B on B). The drastic drop in performance when applying a model trained on A to B points out the need for transfer learning techniques. $T1$ and $T2$ denote the application of our framework with pre-training **T** to equal an identity matrix and 180° rotation matrix, respectively.

We use train mode 0, train mode 1 does not yield any improvements. This is due to the fact, that aligning images via pixel-losses is difficult, which in turn also implies that here training step 2, and the correspondence loss in general, is of not much use. We thus omit results for *Mode 0* and *2*, as these analyze the influences

Table 7.1: Results on the MNIST dataset.

b	Model	Accuracy
100 (0.14)	*Fine-tune*	0.738
	CORAL	0.5
	pix2pix	0.375
	Imp	0.313
	Ours - $T1$	0.766
	Ours - $T2$	**0.912**
1000 (1.43)	Fine-tune	0.908
	CORAL	0.852
	pix2pix	0.898
	Imp	0.711
	Ours - $T1$	0.901
	Ours - $T2$	**0.947**
2000 (2.86)	Fine-tune	**0.972**
	CORAL	0.935
	pix2pix	0.859
	Imp	0.846
	Ours - $T1$	0.966
	Ours - $T2$	0.969
70000 (100)	A on B	0.153
	B on B	0.980

of said loss. Note again the similarity between STNs and our proposed framework, particularly for images, as the same transformation component is used, but also for general applications, when train mode 1 is used. As train mode 1 here does not yield any improvements in performance, this shows that also STNs in this scenario do not perform better.

Still, our proposed variant $T1$ performs very comparably to fine-tuning. On top of that, employing prior knowledge, as done in $T2$, significantly helps improve performance. In particular for small b, very high classification accuracies are reached. The original pix2pix framework is used as another baseline: Although it excels in applying local transformations, such as converting styles and changing pixel intensities, results indicate an inability to understand and model global transformations, such as rotating the full image. Same can be observed for the likewise implicit domain adaptation method *Imp*. This shows the need for frameworks like STNs or ours, and why using explicit transformation matrices can be beneficial: Such transformations can be initialized with meaningful values (or the identity, at last, to not worsen performance), and yield clear results, instead of hallucinating samples, which might be, e.g., blurry and decrease performance. As all methods, *CORAL* becomes better with growing b, but also does not reach performance of fine-tuning or our models.

7.4.3.3 Toy Sequence

For predicting lane changes on synthetic data, model **M** is a single LSTM cell of size 32 followed by a fully connected layer for classification, which equals layer **L**. Solely used input feature is distance to the lane's center line. **C** is an LSTM network of size 16, and we use train mode 1. As this problem mainly serves as preparation for lane change prediction on real data, we only compare our models against fine-tuning, and additionally list similar baselines as before (A on B and B on B). Quantitative results are shown in Table 7.2. We find a previously stated assumption confirmed: When switching from "clean" domain A to "noisy" domain B, *Frequency* nearly doubles. As before, $T1$ denotes our framework with using an identity matrix as pre-training target in step 1. In $T2$, during lane following periods $\begin{pmatrix} 0 & 0 \\ 0.5 & 1 \end{pmatrix}$ serves as target matrix, and an identity matrix during lane change manoeuvres. This forces a smoothing and projection towards the lane's center, when the model's belief is lane-following, but still allows accurate predictions during lane changes. In fact, such encoding of prior knowledge drastically reduces the number of false positive predictions (*Frequency*). The reported *Frequency* is even lower than for the full model trained on B - which we will discuss in the next section. Depending on b, fine-tuning and $T1$ are close together: The used dataset is too small and simplistic to learn meaningful transformations without guided supervision - for different outcomes we again refer to the next section. Figure 7.4 depicts a sample lane change, plotting also the Converter's output and the resulting prediction. One can nicely observe the smoothing effect of $T2$ and check the quality of transformed trajectories visually, as well as examine the resulting smoother prediction.

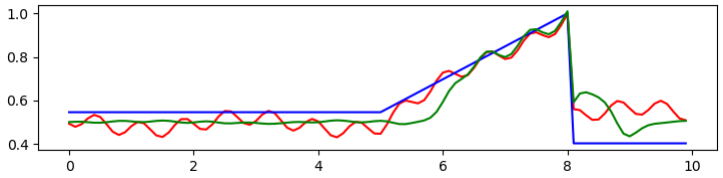


Figure 7.4: Depiction of a simulated lane change to the right. The distance to the lane's left lane boundary is plotted on the y-axis, time in seconds on the x-axis. The "noisy" lane change from domain B is shown in red, and a corresponding one from domain A in blue (for sake of simplicity, just one of the n corresponding ones is shown). The transformed sample after application of the Converter (i.e., the lane change from domain B multiplied by the generated transformation matrix $\mathbf{T}$) is drawn in green (with $T2$), yielding a very plausible converted lane change (image from [6]).

7.4.3.4 Lane Change Prediction

Our used base model $\mathbf{M}$ for predicting lane changes is the model *LSTM* introduced in Section 3.3.2. $\mathbf{L}$ consists of the last fully connected layer for classification. $\mathbf{C}$ is modelled by another LSTM with hidden size 32. However, we modify the set of used input features for solving this problem, and in particular only consider the two features m and v, distance to the lane's center line and lateral velocity. Employing more features in general improves performance of models in their original domains (compare Chapter 3), but complicates knowledge transfer, as more features allow for greater differences in domains - fine-tuning is particularly effected by this. Other transfer techniques perform worse initially, but better eventually - after retraining, the additional information is helpful. The used pre-training targets of step 1 are

Table 7.2: Results of the toy sequence problem. Smaller values for *Frequency*, *Delay* and *Miss* are better, larger ones for *Score*.

b	Model	Frequency	Delay	Miss	Score
100	*Fine-tune*	3.196	0.945	0.121	0.186
(1)	$T1$	2.849	**0.897**	**0.112**	**0.275**
	$T2$	**2.724**	1.077	0.153	0.262
500	*Fine-tune*	3.456	0.888	**0.113**	0.137
(5)	$T1$	3.156	**0.878**	0.117	0.203
	$T2$	**1.410**	1.140	0.140	**0.565**
2000	*Fine-tune*	2.940	**0.903**	0.106	0.257
(20)	$T1$	3.871	0.931	0.120	0.034
	$T2$	**1.300**	0.992	**0.104**	**0.625**
10000	A on B	4.085	0.851	0.108	-
(100)	B on B	2.129	0.542	0.062	-

Table 7.3: Results of the lane change prediction problem. Smaller values for *Frequency*, *Delay* and *Miss* are better, larger ones for *Score*.

b	Model	Frequency	Delay	Miss	Score
100	Fine-tune	7.344	0.612	0.008	0.352
	CORAL	7.996	**0.601**	0.011	0.291
	pix2pix	5.399	0.837	0.039	0.439
	Imp	4.363	0.837	0.006	0.592
(1.8)	Mode 0 - $T1$	7.48	0.637	0.008	0.339
	Mode 0 - $T2$	5.477	0.738	0.008	0.502
	Mode 2 - $T1$	5.835	0.699	**0.005**	0.48
	Mode 2 - $T2$	5.243	0.833	0.011	0.51
	Ours - $T1$	4.573	0.797	0.006	0.578
	Ours - $T2$	**3.373**	0.956	**0.005**	**0.672**
500	Fine-tune	7.744	**0.544**	0.01	0.319
	CORAL	8.336	0.547	0.011	0.265
	pix2pix	6.003	0.842	0.071	0.321
	Imp	4.476	0.835	0.010	0.576
(9.1)	Mode 0 - T1	7.93	0.559	0.01	0.302
	Mode 0 - $T2$	5.138	0.704	0.006	0.536
	Mode 2 - $T1$	5.635	0.668	0.003	0.502
	Mode 2 - $T2$	5.535	0.658	0.003	0.511
	Ours - $T1$	4.848	0.759	**0.002**	0.565
	Ours - $T2$	**3.524**	0.887	**0.002**	**0.669**
1000	Fine-tune	6.551	0.620	0.005	0.424
	CORAL	6.933	**0.608**	0.010	0.383
	pix2pix	2.339	1.157	0.144	0.460
	Imp	3.803	0.800	0.006	0.641
(18.2)	Mode 0 - $T1$	6.629	0.617	0.005	0.418
	Mode 0 - $T2$	5.223	0.706	0.005	0.531
	Mode 2 - $T1$	5.058	0.686	**0.002**	0.552
	Mode 2 - $T2$	4.976	0.702	0.003	0.556
	Ours - $T1$	3.594	0.901	0.01	0.646
	Ours - $T2$	**3.241**	0.888	0.003	**0.691**
5500	A on B	11.672	0.347	0.010	-
(100)	B on B	4.732	0.698	0.005	-

similar to the ones used in the previous section, m and v are transformed to 0.5 and 0 respectively, i.e. neutral positions during *Follow* periods.

Table 7.3 shows quantitative results for all analyzed methods. For this problem, both our proposed method in both variants $T1$ and $T2$ outperforms all others (using train mode 1), both in terms of *Frequency* and total score. Similar to results in the previous section, *Frequency* of the resulting transferred model is even lower than

that of the original model trained purely on B. We explain this by the fact, that we train the original model using cross-entropy, which yields a well-rounded solution (and possibly a local optimum, although this is independent of using cross-entropy or not), not focusing on particular higher-level metrics. Inducing prior knowledge in the form of forcing a smoothing of trajectories then favours the metric *Frequency*. This highlights another exciting application of our method: Using it, in a subsequent training step we can adapt models to exhibit certain desired characteristics, such as reducing the number of false positives or rewarding early predictions, depending on the application.

We here yield a full comparison of all models, e.g. also examining *Mode 0* and *2* with variants $T1$ and $T2$. For variant $T1$, we observe a larger performance difference between our full framework and the other ablation studies (*Mode 0* and *2*). This makes sense, as all models profit from given domain knowledge ($T2$), and with less information available using the selected correspondence pairs and the connected correspondence loss becomes more valuable.

The CORAL framework yields acceptable results, but performs slightly worse than fine-tuning alone. Possible reasons are the relatively small size of the covariance matrix, and that interactions in complex sequences are non-linear and parametrized by more than second-order statistics. Our implicit model *Imp* yields better results. This essentially consists of just the generator of pix2pix, and uses a loss in the actual output space instead of an adversarial loss as training target. This, for once, shows, that such domain adaptation can be done implicitly, as well, but also, that we do not loose expressiveness by using a model-based approach employing transformation matrices. In addition, we would like to remind the reader of the advantages of such a scheme as described in the introduction, and the possibility of pre-training $\mathbf{T}$ with domain knowledge: This ($T2$), especially for smaller b, yields best results by far.

Figure 7.5 plots two aligned lane changes of both domains, showing m and v, as well as corresponding predictions and ground truth. For a comparison, prediction results of fine-tuning are depicted, as well.

7.5 Conclusion

Transfer learning is becoming more and more important: Nowadays, we can train complex, powerful models for many different tasks, often outperforming humans. Still, these models mostly lack an innate ability available to humans: They are trained only for specific purposes, and are not "intelligent" enough to react to changing domains and tasks, needing to be completely re-modelled or re-trained for these. To decrease the effort of doing so, improve performance in new domains or tasks, and also slowly move towards the concept of artificial general intelligence (AGI), transfer learning is a valuable tool and important direction of further research.

In this chapter we proposed a general framework for transfer learning, or domain adaptation, to be precise. For this, we employed a novel correspondence loss, aligning samples from both analyzed domains to learn an explicit transformation matrix.

In addition to allowing insights into what models are calculating and verification of such transformations, we can also pre-train this mapping making use of prior knowledge. We applied our proposed framework to image data and sequential problems, considering digit classification on the MNIST dataset and predicting lane changes. We observed superior performance of our model, especially when prior knowledge is incorporated and when considering sequential problems such as lane changes. While predicting lane changes, application of our framework leads to a a visible smoothing of trajectories, significantly reducing the number of false positive predictions (*Frequency*).

In the future, we would like to see our framework applied for various other tasks, such as image classification on harder, larger datasets, or speech-to-text problems. It would also be interesting to further analyze adversarial approaches for generating trajectories, and in particular apply such models to adapt domains containing of numerical, non-image data.

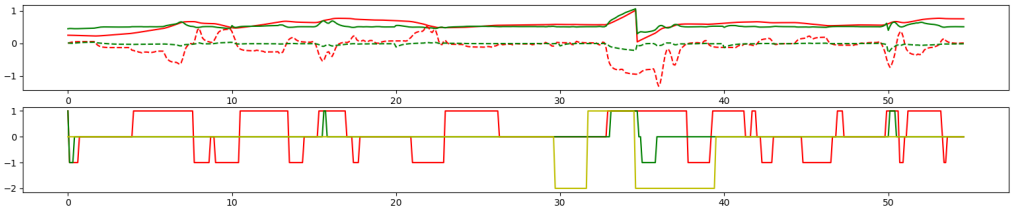


Figure 7.5: Depiction of a lane change to the left. In both plots, time in seconds is plotted on the x-axis. In the top plot, m, once in raw form from domain B (red), and once after being processed by the Converter (green). Similar values are plotted for v, which are drawn using dashed lines. In the bottom plot, corresponding ground truth labels are drawn in yellow, the predictions of fine-tuning in red and the output of our model ($T2$) in green. Here, 1 / -1 denote lane changes to the left / right, 0 *Follow* and -2 *Ignore* labels. As described in Chapter 3, the latter are inserted between *Follow* and lane change labels, and after execution of such manoeuvres, to give models time to reset. Results of applying the Converter are strongly visible, smoothing out fluctuations and scaling down extreme values of the input features, especially during *Follow* periods. Our proposed model outperforms fine-tuning, exhibiting much less false positive predictions and yielding near identical lane change predictions (figure from [6]).

Part V

Conclusion and Outlook

8 Conclusion

In this section we summarize our work, highlight its advantages and limitations, and finally point out interesting directions for future research.

8.1 Summary

A commonly envisioned software architecture for autonomous vehicles consists of the five basic layers perception, fusion, prediction, planning and control. In this work we addressed the building blocks prediction and planning, mostly employing LSTM networks to exploit the sequential nature of examined problems.

Building on existing perception and fusion methods, which generate an environmental understanding based on different sensors, such as cameras, Radar and LiDAR, we started in the field of prediction: The output of such perception / fusion layers commonly is a static representation of a traffic scene, which is not sufficient for the problem of autonomous driving. Virtually all scenes exhibit dynamic properties, such as moving agents and probabilistic events, which need to be modelled in order to allow planning of safe paths for the ego vehicle. In particular, we attended the problem of predicting discrete manoeuvres, namely lane change events. For this, we proposed a series of LSTM networks, among others combining different LSTM cells for different modalities of the problem and attention mechanisms to improve performance and equip models with ways of explaining their decisions. The latter point is especially crucial for safety-critical applications, such as autonomous driving. Our proposed models yield good results, performing better than existing methods. Previously, more classical, frame-based methods, such as Random Forests were applied for this kind of problem [44]. These neglect the temporal component of such problem, in which our RNNs excel. We could, particularly, reduce the number of false positive predictions - which have a great negative influence on felt driving comfort - due to the recurrent nature of our proposed models, which implicitly can be understood as variants of low-pass filters. Additionally, we proposed novel evaluation metrics to directly measure what drivers and passengers experience inside

vehicles, which is an improvement over traditionally used metrics from information retrieval, such as Precision and Recall.

For general machine learning problems, but particularly in the field of autonomous driving, ambiguity and uncertainty is often present, as e.g. vehicles have multiple possibilities of continuing through a scene and intention prediction inherently is ambiguous. Therefore, we extended the MHP framework [8] to different recurrent models, allowing the prediction of multiple hypotheses simultaneously. Our contribution is a generally applicable framework, which can easily be applied for existing models and problems, yielding good results regarding the understanding and capturing of ambiguity. In particular, we examined the application to the previously mentioned problem of predicting lane changes, but also to more general tasks, such as predicting trajectories or text generation.

Equipped with this functionality, we moved towards the area of planning: As full, data-driven driven planning methods exhibit several downsides (such as compounding errors when using Behavioral Cloning), we mitigated this issue by introducing a novel supporting layer, which can be trained in classical supervised fashion, and allows the inclusion of deep learning models into any existing planner, by providing an interface for query answering. In particular, this layer is able to answer whether lane change manoeuvres in the current situation are feasible and safe. We again employed LSTM networks for this, which perform better than existing methods, such as SVMs [86]. Furthermore, we showcased the usage of bidirectional LSTM networks by integrating a planning component, further improving performance.

We then analyzed other interesting connections between the tasks prediction and planning: We combined the aforementioned situation assessment model with our lane change prediction models, training them jointly, proving the connections between these tasks and improving performance for one. Vice versa, we employed prediction models to improve labelling quality of the analyzed planning task.

In recent years, transfer learning has received more research attention and has become more and more important: It is concerned with making models robust to domain shift and re-using existing knowledge for new domains and tasks. This is especially important for the problem of autonomous driving, as here we are faced with a multitude of possible domains: one can think of varying sensor setups, environmental conditions as well as country-specific driving styles and rules. Therefore, we proposed a framework for domain adaptation, which is a subfield of transfer learning concerned with transforming samples between domains. To allow better understandability and verifiability of models, as well as allow inclusion of existing prior domain knowledge, our model contains a component responsible for generating explicit transformation matrices, with which we transform samples between domains. This method yields better results than existing ones for our applications, particularly predicting lane changes and analysis of non-image, sequential data.

8.2 Limitations and Future Work

While we successfully introduced various LSTM networks for core problems needed for solving the problem of fully autonomous driving, future research needs to put more effort into optimizing them for efficient inference in the car. Hardware in vehicles, especially one which is needed for the inference of deep neural networks, is costly and occupied by many tasks already. Next to optimizing networks w.r.t. size and inference times, other interesting and promising fields of research exist, such as network compression and the development of specialized, customized hardware platforms.

Throughout our work, we employed RNNs due to their excellent understanding of complex, temporal dependencies, as present in many fields of autonomous driving. However, addressing the issue of resource consumption, exploring alternative methods is a worthwhile field of future research: Avoiding recurrent connections, but still allowing learning of long-term dependencies, applying 1D-(dilated) convolutions or feed-forward models combined with attention mechanisms yields promising results in different fields.

We already explored interesting connections between different tasks, such as predicting and planning for lane changes. We encourage further studies in this field, moving away from solving tasks separately to complex, joint architectures, making use of all available data and similarities between tasks. More traditional stack-based architectures and full end-to-end learning models certainly both exhibit their own distinct characteristics as well as advantages and disadvantages, and it will be interesting to see which method prevails in the end. We might see a stronger blurring between fields as the levels of automation progress, and would like to see our research grow with it, combining more and more tasks and replacing more components with data-driven models.

Safety and verification obviously is a central part of any autonomous system. With proposed metrics and models (i.e. using attention mechanisms), we made promising contributions to this field. Still, it needs to be further discussed and analyzed, what "safe" in this context means, and especially how we can guarantee and verify such behaviour.

8.3 Epilogue

Although this incredibly exciting journey of finishing a dissertation now comes to an end, there is still a long way to go until fully autonomous vehicles will be a common sight on our streets. We believe our research contributed fruitful ideas to this field, paving the way for future work, especially using data-driven methods and exploiting the sequential nature of many problems. It will be (as always) exciting to see what the future holds, and which methods and techniques eventually will enable us to develop fully autonomous vehicles.

Bibliography

[1] C. Olah. Understanding lstm networks. https://colah.github.io/posts/2015-08-Understanding-LSTMs/. Accessed: 2020-05-06.

[2] O. Scheel, N. S. Nagaraja, L. Schwarz, N. Navab, and F. Tombari. Recurrent models for planning and predicting lane changes. 2020.

[3] O. Scheel, L. Schwarz, N. Navab, and F. Tombari. Situation assessment for planning lane changes: Combining recurrent models and prediction. *Int. Conf. on Robotics and Automation (ICRA)*, 2018.

[4] O. Scheel, N. S. Nagaraja, L. Schwarz, N. Navab, and F. Tombari. Attention-based lane change prediction. *Int. Conf. on Robotics and Automation (ICRA)*, 2019.

[5] A. Berlati, O. Scheel, L. D. Stefano, and F. Tombari. Ambiguity in sequential data: Predicting uncertain futures with recurrent models. *Robotics and Automation Letters (RA-L)*, 2020.

[6] O. Scheel, L. Schwarz, N. Navab, and F. Tombari. Explicit domain adaptation with loosely coupled samples. *arXiv preprint arXiv:2004.11995*, 2020.

[7] A short history of mercedes-benz autonomous driving technology. https://www.autoevolution.com/news/a-short-history-of-mercedes-benz-autonomous-driving-technology-68148.html. Accessed: 2020-02-10.

[8] C. Rupprecht, I. Laina, R. DiPietro, M. Baust, F. Tombari, N. Navab, and G. D Hager. Learning in an uncertain world: Representing ambiguity through multiple hypotheses. *Int. Conf. on Computer Vision (ICCV)*, 2017.

[9] F. Rosenblatt. The perceptron, a perceiving and recognizing automaton project para. 1957.

[10] M. Minsky and S. A Papert. Perceptrons: An introduction to computational geometry. 1969.

[11] A. Krizhevsky, I. Sutskever, and G. E Hinton. Imagenet classification with deep convolutional neural networks. *Int. Conf. on Neural Information Processing Systems (NeurIPS)*, 2012.

[12] Y. LeCun, P. Haffner, L. Bottou, and Y. Bengio. Object recognition with gradient-based learning. *Shape, Contour and Grouping in Computer Vision*, 1999.

[13] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*, 1986.

[14] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. *Advances in Neural Information Processing Systems 27*, 2014.

[15] F. A Gers, J. Schmidhuber, and F. Cummins. Learning to forget: Continual prediction with lstm. *Neural Computation*, 1999.

[16] K. Cho, B. van Merriënboer, C. Gulcehre, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. 2014.

[17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Int. Conf. on Neural Information Processing Systems (NeurIPS)*, 2017.

[18] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[19] C. M. Bishop. Mixture density networks. 1994.

[20] A. Guzmán-Rivera, D. Batra, and P. Kohli. Multiple choice learning: Learning to produce multiple structured outputs. *Int. Conf. on Neural Information Processing Systems (NeurIPS)*, 2012.

[21] S. Lee, S. Purushwalkam, M. Cogswell, V. Ranjan, D. J. Crandall, and D. Batra. Stochastic multiple choice learning for training diverse deep ensembles. *Int. Conf. on Neural Information Processing Systems (NeurIPS)*, 2016.

[22] S. J. Pan and Q. Yang. A survey on transfer learning. *Transactions on Knowledge and Data Engineering*, 2010.

[23] B. Sun, J. Feng, and K. Saenko. Return of frustratingly easy domain adaptation. *AAAI Conference on Artificial Intelligence*, 2016.

[24] J. Hoffman, E. Tzeng, T. Park, J. Zhu, P. Isola, K. Saenko, A. A Efros, and T. Darrell. Cycada: Cycle-consistent adversarial domain adaptation. *arXiv preprint arXiv:1711.03213*, 2017.

[25] G. Koch, R. Zemel, and R. Salakhutdinov. Siamese neural networks for one-shot image recognition. *ICML deep learning workshop*, 2015.

[26] A. R Zamir, A. Sax, W. Shen, L. J Guibas, J. Malik, and S. Savarese. Taskonomy: Disentangling task transfer learning. *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[27] M. Bansal, A. Krizhevsky, and A. Ogale. Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst. *arXiv preprint arXiv:1812.03079*, 2018.

[28] World Health Organization (WHO). Global status report on road safety 2018. december 2018. `https://www.who.int/violence_injury_prevention/road_safety_status/2018/en/`. Accessed: 2020-02-10.

[29] D. A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. *Int. Conf. on Neural Information Processing Systems (NeurIPS)*, 1989.

[30] 2getthere. Operations contract of driverless parkshuttle extended with 2 years. `https://www.2getthere.eu/driverless-parkshuttle/`. Accessed: 2020-05-05.

[31] AARP The Magazine. The driverless car is (almost) here. `https://www.aarp.org/home-family/personal-technology/info-2014/google-self-driving-car.html`. Accessed: 2020-05-05.

[32] M. Ardelt, C. Coester, and N. Kaempchen. Highly automated driving on freeways in real traffic using a probabilistic framework. *Transactions on Intelligent Transportation Systems*, 2012.

[33] K. Wiggers. Waymo's autonomous cars have driven 20 million miles on public roads. `https://venturebeat.com/2020/01/06/waymos-autonomous-cars-have-driven-20-million-miles-on-public-roads/`. Accessed: 2020-05-05.

[34] S. Crowe. Researchers back tesla's non-lidar approach to self-driving cars. `https://www.therobotreport.com/researchers-back-teslas-non-lidar-approach-to-self-driving-cars/`, 2019.

[35] M. Bojarski, D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[36] A. Kendall, J. Hawke, D. Janz, P. Mazur, D. Reda, J. Allen, V. Lam, A. Bewley, and A. Shah. Learning to drive in a day. *Int. Conf. on Robotics and Automation (ICRA)*, 2019.

[37] W. Zeng, W. Luo, S. Suo, A. Sadat, B. Yang, S. Casas, and R. Urtasun. End-to-end interpretable neural motion planner. *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[38] M. Treiber, A. Hennecke, and D. Helbing. Congested traffic states in empirical observations and microscopic simulations. *Phys. Rev. E*, 2000.

[39] A. Jain, A. Singh, H. S. Koppula, S. Soh, and A. Saxena. Brain4cars: Car that knows before you do via sensory-fusion deep learning architecture. *Int. Conf. on Robotics and Automation (ICRA)*, 2016.

[40] A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei, and S. Savarese. Social lstm: Human trajectory prediction in crowded spaces. *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[41] N. Lee, W. Choi, P. Vernaza, C. Choy, P. Torr, and M. Chandraker. Desire: Distant future prediction in dynamic scenes with interacting agents. *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[42] A. Gupta, J. Johnson, F. Li, S. Savarese, and A. Alahi. Social gan: Socially acceptable trajectories with generative adversarial networks. *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[43] A. Sadeghian, V. Kosaraju, A. Sadeghian, N. Hirose, and S. Savarese. Sophie: An attentive GAN for predicting paths compliant to social and physical constraints. *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[44] J. Schlechtriemen, F. Wirthmueller, A. Wedel, G. Breuel, and K. Kuhnert. When will it change the lane? a probabilistic regression approach for rarely occurring events. *Intelligent Vehicles Symposium (IV)*, 2015.

[45] H. Woo, Y. Ji, H. Kono, Y. Tamura, Y. Kuroda, T. Sugano, Y. Yamamoto, A. Yamashita, and H. Asama. Dynamic potential-model-based feature for lane change prediction. *Int. Conf. on Systems, Man, and Cybernetics (SMC)*, 2016.

[46] G. Weidl, A. L Madsen, V. Tereshchenko, W. Zhang, S. Wang, and D. Kasper. Situation awareness and early recognition of traffic maneuvers. *EUROSIM Congress on Modelling and Simulation*, 2016.

[47] J. Schlechtriemen, A. Wedel, J. Hillenbrand, G. Breuel, and K. Kuhnert. A lane change detection approach using feature ranking with maximized predictive power. *Intelligent Vehicles Symposium (IV)*, 2014.

[48] D. Frossard, E. Kee, and R. Urtasun. Deepsignals: Predicting intent of drivers through visual signals. *Int. Conf. on Robotics and Automation (ICRA)*, 2019.

[49] S. Patel, B. Griffin, K. Kusano, and J. J. Corso. Predicting future lane changes of other highway vehicles using rnn-based deep models. *Int. Conf. on Intelligent Robots and Systems (IROS)*, 2018.

[50] J. Zeisler, F. Schönert, M. Johne, and V. Haltakov. Vision based lane change detection using true flow features. *Int. Conf. on Intelligent Transportation Systems (ITSC)*, 2017.

[51] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *Int. Conf. on Learning Representations (ICLR)*, 2015.

[52] V. Mnih, N. Heess, A. Graves, and K. Kavukcuoglu. Recurrent models of visual attention. *Int. Conf. on Neural Information Processing Systems (NeurIPS)*, 2014.

[53] T. Luong, H. Pham, and C. D. Manning. Effective approaches to attention-based neural machine translation. *Empirical Methods in Natural Language Processing (EMNLP)*, 2015.

[54] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

[55] Ngsim project. https://ops.fhwa.dot.gov/trafficanalysistools/ngsim.htm.

[56] R. Krajewski, J. Bock, L. Kloeker, and L. Eckstein. The highd dataset: A drone dataset of naturalistic vehicle trajectories on german highways for validation of highly automated driving systems. *Int. Conf. on Intelligent Transportation Systems (ITSC)*, 2018.

[57] D. Rezende and S. Mohamed. Variational inference with normalizing flows. *arXiv preprint arXiv:1505.05770*, 2015.

[58] A. Van den Oord, N. Kalchbrenner, L. Espeholt, O. Vinyals, A. Graves, et al. Conditional image generation with pixelcnn decoders. *Int. Conf. on Neural Information Processing Systems (NeurIPS)*, 2016.

[59] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[60] D. Silver, 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. Mastering the game of Go with deep neural networks and tree search. *Nature*, 2016.

[61] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *Int. Conf. on Learning Representations (ICLR)*, 2014.

[62] K. Gregor, I. Danihelka, A. Graves, D. Rezende, and D. Wierstra. Draw: A recurrent neural network for image generation. *Inf. Conf. on Machine Learning (ICML)*, 2015.

[63] U. Jain, Z. Zhang, and A. Schwing. Creativity: Generating diverse questions using variational autoencoders. *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[64] J. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. *Int. Conf. on Computer Vision (ICCV)*, 2017.

[65] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein gan. 2017.

[66] Y. Gong, Y. Jia, T. Leung, A. Toshev, and S. Ioffe. Deep convolutional ranking for multilabel image annotation. *Int. Conf. on Learning Representations (ICLR)*, 2014.

[67] J. Wang, Y. Yang, J. Mao, Z. Huang, C. Huang, and W. Xu. Cnn-rnn: A unified framework for multi-label image classification. *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[68] Y. Liu, R. Yu, S. Zheng, E. Zhan, and Y. Yue. NAOMI: non-autoregressive multiresolution sequence imputation. *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[69] Z. Shi, X. Chen, X. Qiu, and X. Huang. Towards diverse text generation with inverse reinforcement learning. *arXiv preprint arXiv:1804.11258*, 2018.

[70] Y. Shao, S. Gouws, D. Britz, A. Goldie, B. Strope, and R. Kurzweil. Generating high-quality and informative conversation responses with sequence-to-sequence models. *arXiv preprint arXiv:1701.03185*, 2017.

[71] L. Bazzani, H. Larochelle, and L. Torresani. Recurrent mixture density network for spatiotemporal visual attention. *arXiv preprint arXiv:1603.08199*, 2016.

[72] A. Kalyan, S. Lee, A. Kannan, and D. Batra. Learn from your neighbor: Learning multi-modal mappings from sparse annotations. *Int. Conf. on Machine Learning (ICML)*, 2018.

[73] B. Gao, C. Xing, C. Xie, J. Wu, and X. Geng. Deep label distribution learning with label ambiguity. *IEEE Transactions on Image Processing*, 2017.

[74] X. Geng and Y. Xia. Head pose estimation based on multivariate label distribution. *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[75] N. Rhinehart, K. Kitani, and P. Vernaza. R2p2: A reparameterized pushforward policy for diverse, precise generative path forecasting. *European Conference on Computer Vision (ECCV)*, 2018.

[76] A. Shahroudy, J. Liu, T. Ng, and G. Wang. Ntu rgb+d: A large scale dataset for 3d human activity analysis. *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[77] A. Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[78] P. Schydlo, M. Raković, , and J. Santos-Victor. Anticipation in human-robot cooperation: A recurrent neural network approach for multiple action sequences prediction. *Int. Conf. on Robotics and Automation (ICRA)*, 2018.

[79] A. Robicquet, A. Sadeghian, A. Alahi, and S. Savarese. Learning social etiquette: Human trajectory understanding in crowded scenes. *European Conf. on Computer Vision (ECCV)*, 2016.

[80] M. Marcus, G. Kim, M. Marcinkiewicz, R. MacIntyre, A. Bies, M. Ferguson, K. Katz, and B. Schasberger. The penn treebank: Annotating predicate argument structure. *Proceedings of the Workshop on Human Language Technology*, 1994.

[81] A. Sadeghian, V. Kosaraju, A. Gupta, S. Savarese, and A. Alahi. Trajnet: Towards a benchmark for human trajectory prediction. *arXiv preprint*, 2018.

[82] P. Brown, V. Pietra, R. Mercer, S. Pietra, and J. Lai. An estimate of an upper bound for the entropy of english. *Computational Linguistics*, 1992.

[83] K. Papineni, S. Roukos, T. Ward, and W. Zhu. Bleu: a method for automatic evaluation of machine translation. *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, 2002.

[84] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[85] S. Ulbrich and M. Maurer. Situation assessment in tactical lane change behavior planning for automated vehicles. *Int. Conf. on Intelligent Transportation Systems (ITSC)*, 2015.

[86] J. Nie, J. Zhang, X. Wan, W. Ding, and . Ran. Modeling of decision-making behavior for discretionary lane-changing execution. *Int. Conf. on Intelligent Transportation Systems (ITSC)*, 2016.

[87] S. Hahn and H. Choi. Self-knowledge distillation in natural language processing. *Int. Conf. on Recent Advances in Natural Language Processing (RANLP 2019)*, 2019.

[88] S. G. Jeong, J. Kim, S. Kim, and J. Min. End-to-end learning of image based lane-change decision. *Intelligent Vehicles Symposium (IV)*, 2017.

[89] D. Isele, R. Rahimi, A. Cosgun, K. Subramanian, and K. Fujimura. Navigating occluded intersections with autonomous vehicles using deep reinforcement learning. *Int. Conf. on Robotics and Automation (ICRA)*, 2018.

[90] E. Balal, R. Cheu, and T. Sarkodie-Gyan. A binary decision model for discretionary lane changing move based on fuzzy inference system. *Transportation Research Part C: Emerging Technologies*, 2016.

[91] Y. Dou, F. Yan, and D. Feng. Lane changing prediction at highway lane drops using support vector machine and artificial neural network classifiers. *Int. Conf. on Advanced Intelligent Mechatronics (AIM)*, 2016.

[92] A. Mousavian, H. Pirsiavash, and J. Košecká. Joint semantic segmentation and depth estimation with deep convolutional networks. *2016 Fourth International Conference on 3D Vision (3DV)*, 2016.

[93] S. Ruder. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098*, 2017.

[94] Y. LeCun and C. Cortes. MNIST handwritten digit database. 2010.

[95] H. Noh, P. H. Seo, and B. Han. Image question answering using convolutional neural network with dynamic parameter prediction. *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[96] S. Antol, A. Agrawal, J. Lu, M. Mitchell, D. Batra, C Lawrence Zitnick, and D. Parikh. Vqa: Visual question answering. *Int. Conf. on Computer Vision (ICCV)*, 2015.

[97] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *Advances in Neural Information Processing Systems (NIPS)*, 2014.

[98] M. Liu and O. Tuzel. Coupled generative adversarial networks. *Advances in Neural Information Processing Systems (NIPS)*, 2016.

[99] K. Bousmalis, N. Silberman, D. Dohan, D. Erhan, and D. Krishnan. Unsupervised pixel-level domain adaptation with generative adversarial networks. *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[100] X. Ouyang, Y. Cheng, Y. Jiang, C. Li, and P. Zhou. Pedestrian-synthesis-gan: Generating pedestrian data in real scene and beyond. *arXiv preprint arXiv:1804.02047*, 2018.

[101] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *Int. Conf. on Machine Learning (ICML)*, 2017.

[102] M. Long and J. Wang. Learning multiple tasks with deep relationship networks. *arXiv preprint arXiv:1506.02117*, 2015.

[103] J. Yim, D. Joo, J. Bae, and J. Kim. A gift from knowledge distillation: Fast optimization, network minimization and transfer learning. *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[104] Z. Luo, Y. Zou, J. Hoffman, and L. Fei-Fei. Label efficient learning of transferable representations across domains and tasks. *Int. Conf. on Neural Information Processing Systems (NIPS)*, 2017.

[105] A. Rozantsev, M. Salzmann, and P. Fua. Beyond sharing weights for deep domain adaptation. *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2016.

[106] P. Isola, J. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[107] X. Li, S. Liu, J. Kautz, and M. Yang. Learning linear transformations for fast arbitrary style transfer. *arXiv preprint arXiv:1808.04537*, 2018.

[108] S. Sabour, N. Frosst, and G. E. Hinton. Dynamic routing between capsules. *Int. Conf. on Neural Information Processing Systems (NeurIPS)*, 2017.

[109] L. Duan, D. Xu, and I. W. Tsang. Learning with augmented features for heterogeneous domain adaptation. *Int. Conf. on Machine Learning (ICML)*, 2012.

[110] B. Paaßen, A. med. Schulz, and B. Hammer. Linear supervised transfer learning for generalized matrix lvq. *Workshop New Challenges in Neural Computation*, 2016.

[111] M. Jaderberg, K. Simonyan, A. Zisserman, et al. Spatial transformer networks. *Advances in neural information processing systems (NIPS)*, 2015.

[112] W. Aswolinskiy and B. Hammer. Unsupervised transfer learning for time series via self-predictive modelling-first results. *Workshop on New Challenges in Neural Computation (NC2)*, 2017.

[113] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[114] F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2015.