RWTH Aachen University
Faculty of Mechanical Engineering

Institute for Automotive Engineering
Univ.-Prof. Dr.-Ing. Lutz Eckstein

Seminar Thesis

Surrounding Object Trajectory Prediction with Recurrent Neural Networks

submitted by:

Till Beemelmanns, matriculation number: 32 06 02

supervised by:

Julian Bock, M.Sc.

Aachen, 14 December 2017

Contents

1 Introduction 5
1.1 Content and Structure 5
2 State of the Art 6
2.1 Prediction of Surround Object Trajectories 6
2.2 Related models for Sequence Prediction 7
3 Artificial Neural Networks 8
3.1 Background 8
3.2 Feedforward Neural Networks 9
3.3 Recurrent Neural Networks 10
3.4 Long Short Term Memory Networks 10
3.5 Prediction Network Architecture 11
3.6 Sequence to Sequence Model 13
4 Approach and Performance Evaluation 15
4.1 Approach 15
4.2 Definition of Performance Evaluation 15
5 Model Design 17
5.1 Dataset 17
5.2 Data Pre-processing 17
5.3 Model Details 19
5.4 Training & Evaluation 20
5.5 Learning Strategies 20
5.6 Stability Analysis 21
6 Results and Evaluation 22
6.1 Initial Hyperparameter Parameter Search 22
6.2 Fine Tuning Results 22
6.3 Prediction Stability 26
7 Conclusion and Outlook 28
8 Literature 29

9 Appendix 33
9.1 Attachment B: Neural network activation functions 33
9.2 Attachment C: Loss functions 34
9.3 Attachment D: Dataset details 35
9.3.1 Dataset Visualisation 35

1 Introduction

For automated driving functions, the prediction of surrounding objects in dynamically changing scenarios is particular relevant. ADAS constantly perform risk assessments of the current situation in order to drive safely and avoid collisions [KAN17]. Further, the future position and movement of agents enables the possibility of an energy optimal driving strategy. Hence, such a driving system needs a robust module that is able to anticipate what happens in its direct environment and react to this situation accordingly. However, the prediction of objects movement pattern is a difficult task. Road users could be classified as cars, pedestrians, cyclists or trucks. All of these objects have different movement characteristics and the individual behavior depends on various influencing factors that can not be determined a priori [KIM17].

Since information storage and transfer has become more cheaper and faster. The vast increasing amount of data which is generated is more and more used for big data analytics. Parallel to this development, GPU performance has exponentially increased in the last years enabling large sized artificial neural network computations [MUJ16]. Thus modern data-driven approaches in the automotive software became feasible. Test drives with vehicles that capture information about object's positions, movement dynamics and other specific data of road users gathered over a long period of time could enable predictions of future situations with the help of machine learning algorithms. One of the main advantages of these deep learning strategies is the fact that it is not necessary to model each handcrafted specific situation, but rather let the model learn to respond appropriately.

Therefore, there exists the need to develop a framework that is able to predict surrounding object movements at various traffic scenarios with the help of a large scaled driving dataset and predictive machine learning algorithms. Consequently, in this thesis the prediction of future trajectories of road users with the help of machine learning will be investigated.

1.1 Content and Structure

This thesis is subdivided into six parts. Section 2 gives a brief summary about different classes of prediction models that deal with movement anticipation and sequence learning strategies. A state of the art review about Artificial Neural Networks can be found in section 3. Section 4 gives the research objective and the approach of this work. Model design and implementation details of the machine learning methodology are presented in section 5. The model prediction quality is discussed and compared with other models in section 6. Finally, section 7 summaries this work and gives an outlook for future research.

2 State of the Art

In the following chapter, a brief review about existing techniques for the prediction of surround objects, followed by a presentation and discussion of machine learning techniques for sequence prediction tasks. Section 2.1 deals with different general approaches for road user motion modelling. In the following section 2.2, different approaches that are not directly intended to predict object movements, but which are capable to model sequential predictions with machine learning algorithms in a different context, are discussed.

2.1 Prediction of Surround Object Trajectories

Since automated driving robots and advanced driver assistance systems (ADAS) will play an increasingly important role in our daily lives, anticipating surrounding agent's future movements is important to improve road safety and trajectory planning. Hereby, the road users can be classified as vehicles, cyclists and pedestrians which have all different movement characteristics. Furthermore, the movement behaviour of the agents which are controlled by humans are often uncertain and they depend from many individual influencing factors. Thus, it is a challenging task to design a single model that is able to forecast future movements of these diverse agents. In the following, an overview of literature that deals with the prediction of surround objects is given.

A basic path anticipation approach was presented by Ammoun et al. [AMM09]. The prediction model is based on a simple linear Kalman filter which uses the acceleration and velocity of the tracked surround object in order to estimate it's future state. Furthermore, the model considers sensor errors in the measurement with the effect that the final prediction is a series of uncertainty ellipses. On this basis a real time collision risk assessment between vehicles could be implemented.

Kang et al. [KAN17] proposed in 2017 a object vehicle path's prediction algorithm. The algorithm is based on an polynomial extrapolation of the object's velocity and position fused together with a movement model that also considers the current motion of the observing vehicle. The input data stream of the surround vehicles was obtained by a real-time sensor data fusion. In various experiments and simulations it was shown that a reasonable performance could be obtained when the weighting factor of the object's velocity is high.

An activity classification approach was proposed by Khosroshahi et al. [KHO16]. The researchers classified manoeuvres of surrounding vehicles at intersections. Hereby, twelve different classes were utilized which describe the movement behaviour of the observed road users. A deep LSTM architecture was deployed in order to learn a labelled dataset and to anticipate the movement based on an observed movement pattern. It was shown that the model was able to correctly predict the movement direction of the vehicles in most cases.

A different machine learning approach based on LSTM architectures was elaborated by Kim et al. [KIM17]. The authors of this article used a occupancy grid map to assign future vehicle

positions in a probabilistic fashion. In order to predict the movement of a surrounding vehicle, the object's last coordinates were fed into a recurrent neural net and the LSTM computed the probability of the occupancy for each finite grid element. It was shown that this methodology has a higher accuracy in comparison to existing filter-based methods. These results were obtained on a data collection from highway driving.

2.2 Related models for Sequence Prediction

A search in the relevant literature yielded that sequence learning approaches have been implemented successfully in various interdisciplinary research fields. Several papers that were recently publicised examined the capabilities of recurrent neural networks such as Long Short-term Memory (LSTM) [HOC97] and Gated Recurrent Units (GRU) for sequence prediction tasks. Very successful applications of these RNN can be found for machine translation [BAH14] [CHO14] [SUT14], caption generation [VIN14], chatbots [VIN15] and image classification [KRI12]. In many cases, modern RNN architectures outperformed traditional approaches and this shows that RNN are suitable for modelling complex long range sequential dependencies. In the following, an overview about recent literature is given that deals in a general manner with recurrent neural networks which are trained on *positional* sequential datasets in order to make future predictions.

Graves [GRA13] used Long Short-term Memory recurrent neural networks for several sequence generating tasks with complex long-ranged time dependencies. In particular, he trained the LSTM network with handwriting sequences based on tracked pen-tip trajectories. Graves was then able to show, that the trained net is capable of generating handwriting samples and in another setting he could compute the probability distribution of future pen tip locations. To the best of the authors knowledge, this approach is the first attempt of training X-Y positional data with a RNN-LSTM and this idea inspired other researchers to train RNN with tracking datasets, as for instance Alahi et al. [ALA16].

A sensor fusion architecture for car driver manoeuvres prediction was presented in [JAI15] by Jain et al. The researchers gathered a huge labelled dataset consisting of different sensory streams such as vehicle dynamics, surrounding information and two-dimensional trajectories from tracked landmarks points of the drivers face. These temporal sequences were concatenated together and were used to train an end-to-end deep learning network that computes the probabilities of the drivers future manoeuvres such as *right turn* or *left line change*. This classifier was compared with several other baseline models and the authors of this publication reported that the proposed LSTM architecture in combination with the facial positional data caused an increase of the performance and lead to state-of-the-art results.

3 Artificial Neural Networks

This chapter provides an overview over artificial neural networks with a particular focus on recurrent neural networks for sequence learning. In the first subsection of this chapter the fundamental background of neural nets is described. Section 3.2 introduces simple feedforward neural networks and basic concepts for training. Section 3.3 gives a detailed description of recurrent neural networks and their application in sequence learning. Finally, in section 3.5 the introduced recurrent networks are used to construct prediction network architectures that are able to model complex sequence tasks.

3.1 Background

Artificial Neural Networks (ANNs) are a class of nature-inspired computational systems. An ANN consist of a large collection of *artificial neurons* that are connected with each other through weighted directed edges that imitate the *synapses* and *neurons* of a biological brain [BAS00]. The weights of the edges represent the strength of the synapses between the neurons [BAS00]. The computational complexity and the memory storage of a single neuron is limited. However, the connection of hundred thousand of artificial neurons arranged in a network compound can achieve remarkable artificial intelligence like performance. The computation of an ANN is triggered when an input signal is send to one or several nodes that spread their output to their connected neighbour nodes, which again forward their signal throughout the whole network. At the initial state of an ANN all weights are randomly chosen and is known as an untrained network [RUM86]. This situation is comparable to a brain of a human new-born in which the majority of the neurons are grown but the connecting synapses are not yet entirely formed. Typically, ANNs are used to approximate unknown complex functions and are trained with the help of very large datasets that are observations of a particular input and their corresponding output of these unknown models. Thereby, the weights of the network are modified in such a way that the artificial network model fits best to the observations. This methodology is also known as *supervised learning* [GRA12].

An artificial neuron, usually described as node or unit, computes its output value by applying an *activation function* to the weighted sum of its input values [BAS00]. Each input $x_{j'}$ is hereby linked with its specific weight $w_{jj'}$. Note that the index notation jj' denotes the weight which receives the edge from node j' and emerges into the unit j. The weighted sum is then added to a bias term b_j and the whole sum is put through the activation function f_j of the unit (cf. figure 3-1),

$$a_j = \sum_{j'} w_{jj'} x_{j'} + b_j \qquad \text{Eq. 3-1}$$

$$\nu_j = f_j\left(a_j\right). \qquad \text{Eq. 3-2}$$

The activation function f_j or *transfer function* is often a $\mathrm{sigmoid}$, $\tanh$ or ReLu function (cf. Attachment 9.1).

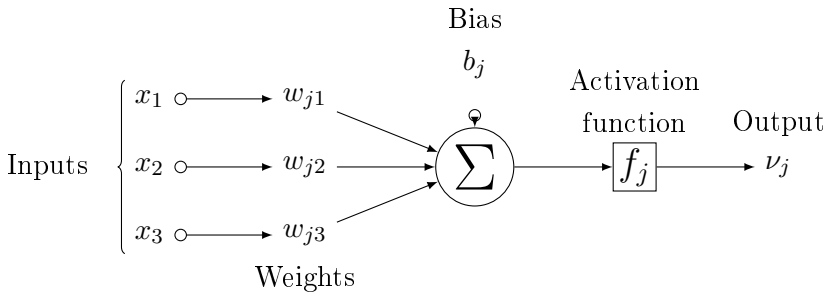


Fig. 3-1: The output of the artificial neuron j is a weighted sum of its inputs which is put through an activation function. Figure adapted from [GER03b].

3.2 Feedforward Neural Networks

One of the simplest way for arranging artificial neural neurons are *Feedforward Neural Networks* (FNNs). In such a network system, the neurons are ordered in vertical, straight, forward layers where each unit of one layer is connected to every node in the following layer. This implies that cyclic connections or recurrent edges, that would create a loop inside a graph, are not allowed. Hence, the information flow inside the graph is directed only in one direction. In contrast, *recurrent neural nets* have a feedback loops or also recurrent edges, resulting in a cyclic graph (cf. Section 3.3). Figure 3-2 visualises a deep fully connected feedforward net with six input nodes that are organised in the *input layer*. When the input $x = (X_1, \ldots, x_n)$ is passed to the net, the input layer propagates its results to the nodes of the *hidden layer*, which then in turn transfer their activation to the *output layer*, respectively the output units. FNN are used for regression and classification task and are usually trained with the supervised learning methodology.

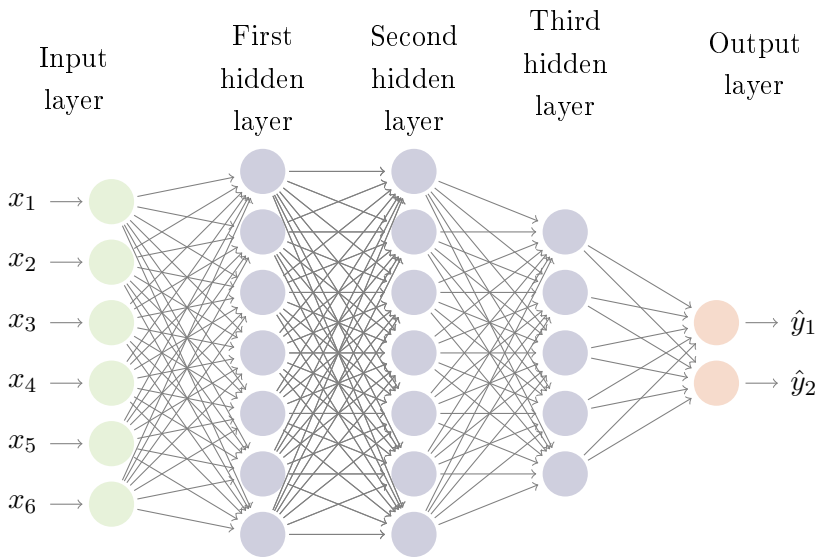


Fig. 3-2: Fully connected deep feedforward neural net with three hidden layers and two output units. Note that there is no connection between units from the same layer.

3.3 Recurrent Neural Networks

Recurrent neural networks (RNNs) are a subclass of Deep Neural Networks (DNNs). RNNs have the unique feature that edges transfer data from the previous time step to the current state. These loops, called recurrent edges, allow the information to persist over time and enable complex, time-dependent sequence modelling. In contrast to FNNs RNNs do not require a fixed dimensionality of the input and the output. This makes RNNs a suitable choice for tackling tasks with differing sequence lengths.

Before we start, a few terms and their mathematical notations are introduced. Since we consider discrete sequences that are sampled at certain *time steps* indexed by t, we define the input sequence,

$$X = (x_1, x_2, \dots x_T) \qquad \text{Eq. 3-3}$$

and the corresponding output sequence,

$$Y = (y_1, y_2, \dots y_T) \qquad \text{Eq. 3-4}$$

where each data point x_t,y_t is real vector with fixed dimensionality

$$x_t \in \mathbb{R}^n, \;\; y_t \in \mathbb{R}^n. \qquad \text{Eq. 3-5}$$

For simplicity, the vectors of the input and the output have the same dimensionality and the sequences X and Y have equal length. In fact RNNs do not need a fixed sequence length as they can deal with different input and output dimensions. When a RNN computes predicted output values, the sequence is denoted in the following by $\hat{Y}$ and $\hat{y}_t$ respectively.

3.4 Long Short Term Memory Networks

Long Short Term Memory networks also called LSTMs where introduced by Hochreiter & Schmidhuber [HOC97] in the late 90's and are known for their good performance in storing and remembering information for a long period of time outperforming standard RNNs. Their recent success was caused by the increasing price-performance ratio of GPUs and the advances of deep learning algorithms such as optimisation techniques and parallel computing enabling large-scale learning [LIP15]. In recent literature, LSTMs reached state-of-the-art performance in various sequence learning tasks. Up to date, the application of LSTMs in speech processing and natural language translation appears to be the most favoured research area [SUT14] [GRA13] [CHO14] [BAH14]. Further applications are chatbots [VIN15], image caption generation [VIN14], handwriting prediction [GRA13] and human motion prediction [ALA16].

In the literature many different variations of the original cell structure exist, such as the popular *Peephole* variant [GER03a], the Gated Recurrent Unit (*GRU*) or the Depth Gated LSTM [YAO15]. Since the former variant is used in the later implementation (cf. Section 5.3) and is considered in the literature as the state of the art LSTM structure [GRA13], a detailed overview is now given.

Figures 3-3 and 3-4 depict the LSTM memory cell architecture by Gers and Schmidhuber [GER03a] which extends a basic LSTM structure with the so called *Peephole Connections*.

Both diagrams represent the same cell structure. While figure 3-3 has cell-centred ordered structure, the second figure explicitly passes the cell state from the previous timestep C_{t-1} as an input to the current cell instance. The additional Peephole Connections have the effect that each gate layer is allowed to inspect the current cell state. Gers et al. report that this new architecture improves the accuracy of timing tasks that require the accurate measurement or generation of timed events leading to the modern LSTM cell network structure,

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}C_{t-1} + b_i) \quad \text{Eq. 3-6}$$

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}C_{t-1} + b_f) \quad \text{Eq. 3-7}$$

$$\tilde{C}_t = \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \quad \text{Eq. 3-8}$$

$$C_t = f_t \otimes C_{t-1} + i_t \otimes \tilde{C}_t \quad \text{Eq. 3-9}$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}C_t + b_o) \quad \text{Eq. 3-10}$$

$$h_t = o_t \otimes \tanh(C_t) \quad \text{Eq. 3-11}$$

where

- $x \otimes y$: denotes the element-wise product
- Candidate Internal state $\tilde{C}_t$: The candidate internal state is computed at the input node and represents possible future states of the memory.
- Forget gating vector f_t: The forget gate controls the information
- Input gating vector i_t: The gate i_t was introduced to protect the memory contents from currently unwanted input.
- Internal state C_t: The core of the cell is denoted with C_t.
- Output gating vector o_t: Before the new computed internal state C_t is forwarded as an output of the recurrent cell, it is run through a $\tanh$ activation and it is multiplied by the output gate o_t.

3.5 Prediction Network Architecture

There exists various architectures that arrange RNN cells in different shape and with differing information flows that are designed for their corresponding task. Figure 3-5 visualises different applications of sequence learning tasks which are common in literature. The subfigure on the top left represents a sequence generation task with a given single data point. This could be for example a image caption generation task like in [VIN14]. In the opposite case on the top right, a sequence is classified by one output vector. An example application for that would be classification of movie reviews into different moods [TIM14]. A *sequence-to-sequence* mapping which has been used for machine translation [CHO14] is depicted on the bottom left. Predictions for every new time step in real time applications are realise by an architecture visualised on the bottom right.

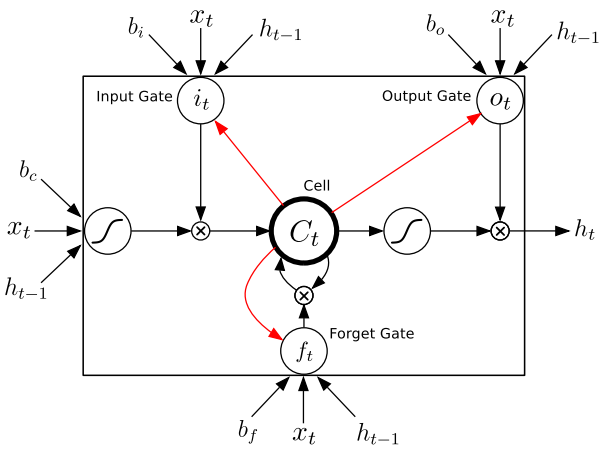


Fig. 3-3: Long Short Term Memory Cell with Peephole connections (red arrows) [GRA13], cell-state centred representation.

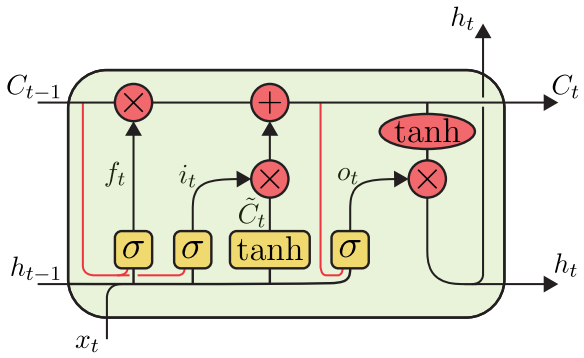


Fig. 3-4: Long Short Term Memory Cell with Peephole connections (red lines), explicit representation similar to Olah [OLA15].

In the following, such a prediction architecture is introduced, which is later used in chapter 5 for the trajectory prediction task. This design can be classified as an *encoder-decoder* structure and it was originally designed for natural language processing tasks. To the best of the authors' knowledge, the first encoder-decoder architecture for sequence-to-sequence mapping was developed by Sutskever et al. [SUT14]. The main idea of this architecture is to *encode* the input sequence to an internal representation or so-called *context vector* with fixed length. This context vector can be regarded as a summary of the input and it is fed into the *decoder* in order to generate the sequential output.
We introduce now an abbreviation for a Long-Short-Term Memory cell,

$$\mathrm{LSTM}(X) = (C_{t-1}, h_{t-1}, x_t) \quad \forall t \qquad \text{Eq. 3-12}$$

where X is the observation sequence with length $N_{obs} \in \mathbb{N}^+$

$$X = (x_1, x_2, \dots x_{N_{obs}}). \qquad \text{Eq. 3-13}$$

Further the predicted output sequence has now the length $N_{pred} \in \mathbb{N}^+$,

$$\hat{Y} = (\hat{y}_1, \hat{y}_2, \dots, \hat{y}_{N_{pred}}). \qquad \text{Eq. 3-14}$$

The the internal representation of the input sequence or the context vector is denoted by V with fixed dimensionality.

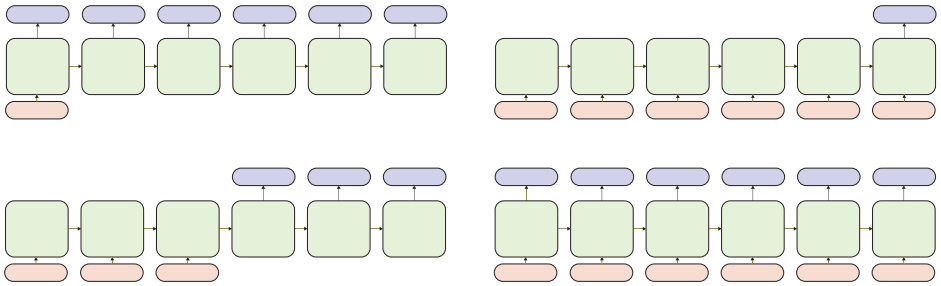


Fig. 3-5: Sequence mapping scheme based on figure from Karpathy [KAR15] with input represented as red rectangles, recurrent cells in green and output in blue.

3.6 Sequence to Sequence Model

This model was described by Sutskever et al. [SUT14] and represents the first encoder-decoder LSTM architecture with an additional context vector. It was originally designed for a language translation task and it achieved compelling results in several benchmark methods [PAP02].

The mechanism of this model can be explained by the following. Usually, the encoder and the decoder are trained to maximise the conditional distribution $p(y_1, \ldots, y_{N_{pred}} | x_1, \ldots, x_{N_{pred}})$ over the input and output sequences where N_{pred} may be different from N_{obs}. The net computes this probability by reading the input X element by element in the encoder. After reaching the `<EOS>` token, the fixed dimensional representation V is given by the last hidden state of the LSTM ($h_{N_{obs}}$ and $C_{N_{obs}}$). The decoder consists of another LSTM with an initial state which is set to the summary V and it is conditioned to generate the output sequence $\hat{Y}$ based on the previous prediction $\hat{y}_{t-1}$ and the previous hidden states, respectively V. A pseudo formulation of this architecture is given for the encoder by

$$\begin{aligned} X &= (x_1, ..., x_{N_{obs}}) \\ V &= \mathrm{LSTM}(X) \end{aligned} \qquad \text{Eq. 3-15}$$

and for the decoder by,

$$\begin{aligned} \hat{y}_1 &= \mathrm{LSTM}(C_0, h_0, V) \\ \hat{y}_t &= \mathrm{LSTM}(C_{t-1}, h_{t-1}, \hat{y}_{t-1}) \;\; \forall\, t = 2, \ldots, N_{pred}. \end{aligned} \qquad \text{Eq. 3-16}$$

In addition to that figure 3-6 visualises this scheme explicitly. The network is trained as usual. The true output y_t is fed to the decoder and the error propagated through the whole architecture in order to fit the the input.

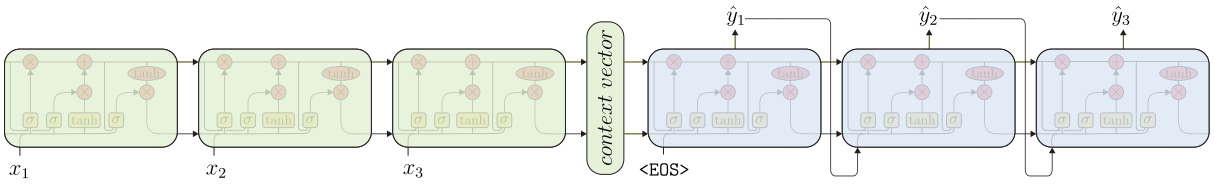


Fig. 3-6: Sequence to sequence architecture by [SUT14] with $N_{obs} = N_{pred} = 3$. In green the encoding LSTM and in blue the decoder. The prediction of the decoder $\hat{y}_t$ at each time step t becomes the input for the decoder at the next time step. Figure adapted from Lipton et al. [LIP15].

4 Approach and Performance Evaluation

In this chapter, the general approach of this thesis is presented in 4.1. Subsequently, it is necessary to define performance criteria of a prediction model in order to compare different models or model configurations with each other. These definitions will be explained in section 4.2.

4.1 Approach

The analysis of the relevant existing literature for prediction models shows that it is desirable to investigate the potentials of recurrent neural networks for general object trajectory prediction. Sequence to sequence designs combined with LSTM cells performed remarkably well in various research problems, in which complex time dependant sequential data was trained and generated. For example, natural language processing tasks or handwriting recognition were successfully modelled with this special machine learning approach.

Consequently, the approach of this thesis consists of the application of a LSTM architecture on an object trajectory prediction setting. Hereby, an encoder-decoder architecture should be used in order to process a certain observation of an object and consequently predict the most probable future trajectory. By this means, a sequence to sequence mapping will be deployed. The foundation of this deep learning approach should be a dataset that contains a set of trajectories that were captured from a vehicle's ego perspective. In addition to observations of the ego vehicle, it is also possible to use the ego motion as a model input. The yaw rate and the velocity of the test vehicle directly influence the relative position and relative velocity of its surrounding objects. As shown in [KIM17], fusing ego motion with relative spatial information is a common modelling approach. Hence, different end-to-end learning strategies can be formulated. E.g. it is possible to consider spatial coordinates of an object with ego motion fusion or use a relative history mapped onto spatial coordinates without ego motion. A search in the relevant literature yielded that there does not exist any comparison between such approaches. Hence, there is a need to investigate different learning strategies with different mapping techniques.

Based on the obtained results of the LSTM model and the learning strategies, it is necessary to compare and evaluate the prediction quality in each case. Therefore, another goal in this thesis is to analyse, whether the forecasting reliability and stability is comparable to a baseline model and whether the model's predictions are acceptable for a safety-critical application.

4.2 Definition of Performance Evaluation

In order to determine the prediction quality of a model or a special model configuration, it is essential to define performance criteria. To persevere consistency with relevant literature, some of the metrics used by [ALA16] will be also used in this work:

- **Mean Squared Displacement** denoted by $\mathcal{L}_{MSD}$
  Average of the squared distances between all estimated coordinates and the true coordi-

nates of all test trajectories

- **Mean Final Displacement** denoted by $\mathcal{L}_{MFD}$
  Average of the distance between predicted final position and the true final position of all test trajectories

- **Mean Displacement** denoted by $\mathcal{L}_{MD}$
  Average of the distances between all estimated coordinates and the true coordinates of all test trajectories

Implementation details of these metrics can be found in the attachment 9.2.

5 Model Design

This chapter gives a detailed description of the implementation of the approach of this seminar thesis. Section 5.1 specifies the datasets on which the deep learning approach will be applied. Consequently, section 5.2 displays a pre-processing pipeline that was applied on every dataset to ensure processability with sequence to sequence models. An in-depth description of the model design and the deployed LSTM cell is given in 5.3. Details about the training procedure and the evaluation of the model quality can be found in section 5.4. In order to measure the robustness of a trajectory prediction model, a stability analysis method is proposed in section 5.6.

5.1 Dataset

In this thesis, three datasets from the KITTI Dataset [GEI13] are used which were all captured with an identical setup. A visualisation and statistics for each of the datasets can be found in Attachment 9.3. The datasets contain the relative positions and velocities of the tracket objects which are classified as cars, trucks, cyclists and pedestrians. These information are sampled at $10Hz$. In addition to that, movement information of the ego vehicle is also stored in the datasets. The following three road scenarios were choosen in order to provide a realistic driving scenario for the evaluation of the prediction models:

- **City**: Urban scenario with cars, pedestrians and cyclists. Captured dataset has a total length of 7 minutes and 41 seconds.
- **Residential**: Slow traffic flow with cars and only few pedestrians. Captured dataset has a total length of 14 minutes and 51 seconds.
- **Road**: Fast traffic flow with cars and some few trucks. Captured dataset has a total length of 3 minutes and 44 seconds.
- **All**: All previous datasets merged together. Captured dataset has a total length of 26 minutes and 16 seconds.

5.2 Data Pre-processing

Almost every deep learning approach makes use of intense data pre-processing in order to make the data usable for neural networks. This also applies to our setting. We start from the premise that a given dataset consists of a set of uncorrelated single trajectories with different length, but with small measurement errors. Thereby, each trajectory consists of several data points sampled with a fixed sampling frequency f_s measured in $[Hz]$. One data point denotes the position of the object in a relative coordinate system at time t. In addition to that, the ego motion of the observing vehicle is capured in this specific time step. Thus, the trajectory Y_i can be written as a four-dimensional matrix where the first column denotes the x-coordinates and the second column consists of the y-coordinates measured in $[m]$. The third column contains

the current velocity in $[m/s]$ and the fouth column contains the yaw rate of the vehicle:

$$Y_i = \begin{pmatrix} x_i^1 & y_i^1 & \nu_i^1 & \phi_i^1 \\ x_i^2 & y_i^2 & \nu_i^2 & \phi_i^2 \\ \vdots & \vdots & \vdots & \vdots \\ x_i^{N_i-1} & y_i^{N_i-1} & \nu_i^{N_i-1} & \phi_i^{N_i-1} \\ x_i^{N_i} & y_i^{N_i} & \nu_i^{N_i} & \phi_i^{N_i} \end{pmatrix} \in \mathbb{R}^{N_i \times 4}. \quad \text{Eq. 5-1}$$

where $i = 1, \ldots, K$ with K the total amount of trajectories in the dataset and N_i the total number of sample points of trajectory Y_i.

In the following the basic preprocessing steps are presented, that are applied to every trajectory of a dataset in order to obtain a new dataset that is suitable for supervised training.

1. **Smoothing**
   Usually the live capture measurement, especially those measured through a laser scanner, are very noisy. Hence, a one-dimensional Gaussian filter from the SciPy package [JON15] is applied to the xy-coordinates in order to smooth the trajectories.

2. **Feature Generation**
   In addition to the absolute position of the objects, an relative description of the positions is applied. Therefore, the preprocessing class implements a polar-coordinate system that computes the radius r and angular heading α of the tracket road user. Hereby r can be obtained by Pythagoras,

   $$r_i^n = \sqrt{(x_i^n - x_i^{n-1})^2 + (y_i^n - y_i^{n-1})^2} \quad \text{Eq. 5-2}$$

   and the α can be computed by

   $$\alpha_i^n = \text{arctan}(y_i^n - y_i^{n-1}, x_i^n - x_i^{n-1}) - \text{arctan}(y_i^{n-1} - y_i^{n-2}, x_i^{n-1} - x_i^{n-2}). \quad \text{Eq. 5-3}$$

   The implementation of the feature generation and the equations 5-2 and 5-3 are taken from [WIS17].

3. **Normalisation**
   Data normalisation is an essential step of the pre-processing for machine learning. Two different scaling method are implemented in the preprocessing class. The first method uses the maximal and minimal bounds of the dataset. Hereby, all data points of the dataset are linearly scaled from the original domain to the new region $[-1, 1]^2$. Another method scales the dataset in such a way that this has a mean of zero and a standart deviantion of one. These scaling methods ensure that the activation functions in the neural net are fully addressed and thus a faster convergence of the training procedure is archived.

4. **Sliding Window**
   Sliding Window is a technique that subsamples a single trajectory with two constant sized

moving intervals into several subparts. The first interval is intended to generate the *observation* of the object's movement. This sequence is also known as the *history* of the object. The observation window is then incrementally moved, simultaneously with the prediction window, over the next coordinate pairs until the prediction window reaches the last point of the trajectory.

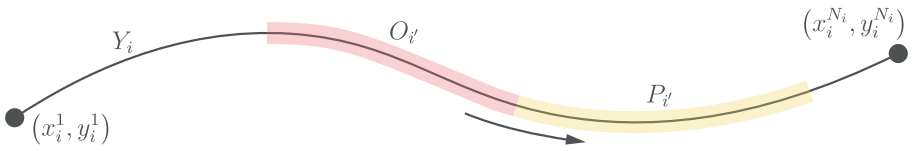


Fig. 5-1: Sliding Window applied to trajectory Y_i. In red the window, the observations are taken into account while the prediction trajectories are captured by the yellow window. That applies for $1 \leq i' \leq N_i - (N_{obs} + N_{pred}) + 1$.

Hence, from one single trajectory Y_i with length $N_i \geq N_{obs} + N_{pred}$, we obtain several observation and prediction trajectories that are denoted in the following by $O_{i'}$ and $P_{i'}$. This Sliding Window method can be applied with arbitrary lengths that are defined in the configuration file, but in the standard case $N_{obs} = 8$ and $N_{pred} = 12$ like in [ALA16] are used. Note that a trajectory Y_i is removed from the training dataset if $N_i < N_{obs} + N_{pred}$. That means, the sequence is too small to be processed by the sliding window.

To summarise the preprocessing, the following matrix is obtained. It describes one trajectory of a single object and the ego motion of the observing vehicle at this specific time interval.

$$Y_i = \begin{pmatrix} x_i^1 & y_i^1 & r_i^1 & \alpha_i^1 & \nu_i^1 & \phi_i^1 \\ x_i^2 & y_i^2 & r_i^2 & \alpha_i^2 & \nu_i^2 & \phi_i^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_i^{N_i-1} & y_i^{N_i-1} & r_i^{N_i-1} & \alpha_i^{N_i-1} & \nu_i^{N_i-1} & \phi_i^{N_i-1} \\ x_i^{N_i} & y_i^{N_i} & r_i^{N_i} & \alpha_i^{N_i} & \nu_i^{N_i} & \phi_i^{N_i} \end{pmatrix} \in \mathbb{R}^{N_i \times 6} \qquad \text{Eq. 5-4}$$

5.3 Model Details

The actual implementation of the used LSTM is done in the Python module *Recurrent Shop* [RAH16a] which is accessed by the seq2seq package [RAH16b]. The implemented LSTM cell is the peephole LSTM variant [GER03a] with a $\mathrm{hard\ sigmoid}$ activation function (cf. attachment 9.1). The seq2seq package deploys an encoder and decorder architecture which is jointly trained using an input and output sequence. In the studied case, the input sequence is the preprocessed observation trajectory O_i, whereas the target sequence is defined as the corresponding prediction sequence P_i. Hence, the arising general optimisation problem can be written as

$$p_w(P|O) = \underset{p_w(P,O)}{argmin} \{\mathcal{L}(\hat{P}, P)\} \qquad \text{Eq. 5-5}$$

where $\hat{P} = f(O; w)$ is the model's prediction and w are the weights of the RNN.

5.4 Training & Evaluation

As we perform supervised learning with the presented approach, it is crucial to train and to evaluate the model on different datasets. Therefore, the model evaluation is done using cross validation. By this means, the dataset is split with a fixed integer $n = 5$ into equally sized parts. A predefined count $c = 4$ denotes the number of iterations that is performed on this split dataset. In each iteration four of this splits are jointly used to train the model, whereas the remaining split is used as the test split. In the next iteration the train and test splits are rotated in order to compute a homogeneous performance criteria considering the whole dataset. When the training of one iteration has finished, the observation trajectories of the test split are taken as an input to the model $f(\cdot, w)$. Consequently, the output of the RNN are the corresponding predictions $\hat{P}_i$:

$$\hat{P}_i = f(O_i, w) \quad \forall i \in \text{test-split} \qquad \text{Eq. 5-6}$$

Subsequently, the predictions are measured against the true trajectories P by computing the introduced performance evaluation criteria $\mathcal{L}(\hat{P}, P)$. Then, the final model performance concerning the whole dataset is computed by averaging over all results of the iterations.

In addition to the cross validation method, all samples in the test split are randomly shuffled, in order to archive a possible faster convergence of the training [BEN12].

5.5 Learning Strategies

The whole data preprocessing offers several approaches in order to learn the trajectory prediction task. On the one hand there are the spatial x-y coordinates of the object and on the other hand it is possible to additionally use the relative $R\alpha$ data stream of one single object. Further, it is possible to consider the ego motion of the observing vehicle as a model input like in [KIM17]. As there are no comparable approaches in the relevant literature, it is not possible to give an a-prior performance estimation of the following learning strategies. Hence the best strategy has to be determined by trial and error.

- $XY \rightarrow XY$: Learning of local paths.
- $XYR\alpha \rightarrow R\alpha$: Learning of local movement patterns.
- $R\alpha \rightarrow R\alpha$: Learning of a global movement model.
- $XYV \rightarrow XY$: Learning of local paths jointly with the ego velocity.
- $XYV\phi \rightarrow XY$: Learning of local paths jointly with the ego motion.
- $XYR\alpha V\phi \rightarrow R\alpha$: Learning of global movement patterns jointly with the ego motion.

Where the notation $feature \rightarrow label$ represents the sequence mapping that is learned by the neural network.

The approaches that use XY as a feature need the mean squared displacement (cf. Eq. 9-3) as a *loss function*. As soon as the $R\alpha$ is used it is needed to adapt the loss function in order to evaluate adequately the relative position information (cf. Eq. 9-6). Further, the $R\alpha$ output is post-processed in such a way that the corresponding trajectory is reconstructed.

5.6 Stability Analysis

In section 4.1 it was stated that a prediction model must be able to cope with uncertainties in the measurements without evoking instabilities in the prediction. Thus, in order to investigate the stability of a RNN prediction model, a basic uncertainty analysis method is explained.

If a trained model $f(\cdot, w)$ and a test observation trajectory O_i is considered, it is the task to determine the prediction stability of this single observation. Hence, artificial measurement uncertainties are introduced by adding noise to the input. To archive this, each single element of the trajectory O_i is added by a random number, drawn from a Gaussian distribution with zero mean $\mu = 0$ and with a fixed standard deviation of $\Sigma = 0.1\ [m]$. The resulting matrix is thus the perturbed trajectory $O_i^{\sim}$. This procedure is repeatedly done for $v = 2048$ times with the effect that a new noisy dataset is generated. It is denoted by $O^{\sim} \in \mathbb{R}^{V \times N_{obs} \times 2}$. Consequently, the perturbed trajectories are pre-processed and finally used to perform the prediction task obtaining a set of model predictions

$$\hat{P}^{\sim} = f\left(O^{\sim}, w\right). \qquad \text{Eq. 5-7}$$

The result is in turn a stochastic distribution that is visualised by a heat map. By this means, the stability of the model can be manually evaluated and interpreted (cf. section 6.3).

6 Results and Evaluation

This chapter gives a detailed discussion and evaluation of the presented approach. In the first section 6.1, an initial hyper-parameter search is performed and discussed. The performance of the six different sequence to sequence architectures is compared in section 6.2. The prediction stability of the presented approach is visualised and assessed in section 6.3.

6.1 Initial Hyperparameter Parameter Search

An initial rough hyperparameter search was performed with the tool Hyperopt [BER13b] in order to obtain an idea how the parameters could be chosen. This tool applies the TPE (Tree-structured Parzen Estimator) search algorithm that was especially designed for optimising ANN's hyper-parameters [BER13a]. The optimisation target was to minimise the loss $\mathcal{L}_{MSD}$ of a cross validation sample. Hereby, 50 optimisation iterations were executed for each dataset and each learning strategy. A wide set of hyperparameters including dropout, batch size, scaling strategy and loss function was used. Table 6-1 gives obtained optimal fine parameter space for the fine tuning optimisation, that is described in the following section (cf. section 6.2).

Learning Strategy	Depth d	Hidden Dimensions h	Learning Rate η	Scaling Method	Loss Function	Epochs e
$XY \rightarrow XY$	1, 2	64, 128	0,5 - 1,5	bounds	MSD	1-300
$XYR\alpha \rightarrow R\alpha$	1, 2	64, 128	0,05 - 0,5	mean standard deviation	MSDOC	1-300
$R\alpha \rightarrow R\alpha$	1, 2	64, 128	0,05 - 0,5	mean standard deviation	MSDOC	1-300
$XYV \rightarrow XY$	1, 2	64, 128	0,5 - 1,5	bounds	MSD	1-300
$XYV\phi \rightarrow XY$	1, 2	64, 128	0,5 - 1,5	bounds	MSD	1-300
$XYR\alpha V\phi \rightarrow R\alpha$	1, 2	64, 128	0,05 - 0,5	mean standard deviation	MSDOC	1-300

Fig. 6-1: An initial parameter search led to this rought parameter space for each learning strategy. This parameter space is used for the fine hyperoptimisation.

6.2 Fine Tuning Results

In order to assess the prediction quality of the sequence to sequence model a simple baseline model was implemented and applied on the datasets. An out-of-the-box Kalman filter with a constant velocity movement model was deployed. The observation trajectory is hereby utilised to determine the state of the tracked object. Subsequently, the state at the last data point of the observation is used to extrapolate the trajectory for N_{obs} time steps.

The qualitative model comparison is performed by computing the performance evaluation criteria of each model and for all datasets. Hereby, the results of the proposed learning strategies are obtained by the fine hyper-parameter optimisation. The uncertainty parameters in the Kalman filter where also determined through an optimisation. Both, baseline and sequence to sequence architectures obtained identical preprocessed trajectories. This should ensure comparability among them.

The data in Figure 6-2 indicate, that the sequence-to-sequences approaches outperforms the baseline model in most cases. In particular, the performance of the $XYR\alpha \rightarrow R\alpha$ and the $R\alpha \rightarrow R\alpha$ learning strategy is significantly better than all other learning strategies. Considering

the ego motion in the input of the approaches does not lead to an overall improvement of the prediction quality. Apparently, the neural net is not able to successfully process these additional information. Thus, a sensor fusion or a movement model is not achieved with these approaches. Moreover, it is quite plausible that the learning strategy $XY \rightarrow XY$ performs poor on the *Road* dataset. This can be explained by the fact that the *Road* dataset is the smallest of all datasets (cf. section 9-2) which leads to lack of regional training data. This applies also for the $XYV \rightarrow XY$ and $XYV\phi \rightarrow XY$ mappings.

The $R\alpha \rightarrow R\alpha$ approach seems to be most effective on the datasets *Residential*, *Road* and *City*. It can be concluded that this approach does not need much training data for a very good prediction quality. Thus, this simple approach learns a precise global movement model just based on the yaw rates and the distances between time steps. But if we consider the much more dense dataset *All*, the $XYR\alpha \rightarrow R\alpha$ strategy is slightly more accurate. Combining positional data with the $R\alpha$ vector as a model input seems to be beneficial on a dense dataset. Here, further research with much larger datasets has to be performed, since the *All* dataset captured only 26 minutes with 1233 independent trajectories. Much larger datasets could be deployed in this case.

Figures 6-3 and 6-4 visualise the prediction results of the Kalman Filter and the six learning strategies. In addition, the real future trajectory (green) and the corresponding observation (blue) is shown. These figures indicate, that the RNN is able to approximate the ground truth much more accurately than the baseline model especially considering winding trajectories. However, some predictions are slightly error prone. In Figure 6-4 it is clearly visible that all strategies based on a local mapping (mapping to XY) have a worse accuracy than strategies that map to $R\alpha$. It becomes evident that an abnormal movement pattern is better generalized by an $R\alpha$ learning. On the other hand, a XY mapping tends to minimise the local error and thus some kind of mainstream prediction is achieved with these strategies. Note that the underlying training data is shown cyan-coloured in the background. All XY based strategies are attracted by the areas with a high density (dark cyan-coloured background).

Metric	Dataset	Kalman filter	$XY \rightarrow XY$	$XYR\alpha \rightarrow R\alpha$	$R\alpha \rightarrow R\alpha$	$XYV \rightarrow XY$	$XYV\phi \rightarrow XY$	$XYR\alpha V\phi \rightarrow R\alpha$
$\mathcal{L}_{MSD}$	All	1.053	0.673	**0.463**	0.483	0.659	0.798	0.469
	Residential	1.001	0.711	0.512	**0.502**	1.124	1.647	0.553
	Road	1.656	2.971	0.841	**0.798**	18.187	22.876	2.299
	City	1.003	0.868	0.461	**0.422**	0.913	2.825	0.503
$\mathcal{L}_{MD}$	All	0.652	0.482	**0.372**	0.488	0.488	0.719	0.379
	Residential	0.688	0.501	0.393	**0.387**	0.642	0.719	0.410
	Road	0.728	0.962	0.438	**0.402**	2.600	3.105	0.804
	City	0.618	0.537	0.373	**0.344**	0.583	0.879	0.403
$\mathcal{L}_{MFD}$	All	1.296	0.832	0.833	0.936	0.936	1.283	**0.829**
	Residential	1.379	0.972	**0.875**	0.880	1.181	1.281	0.885
	Road	1.397	1.700	0.916	**0.880**	4.377	5.290	1.554
	City	1.225	1.037	0.843	**0.785**	1.039	1.171	0.873

Fig. 6-2: Quantitative results of all prediction models on all dataset. The performance for each model is measured with the proposed model evaluation criteria.

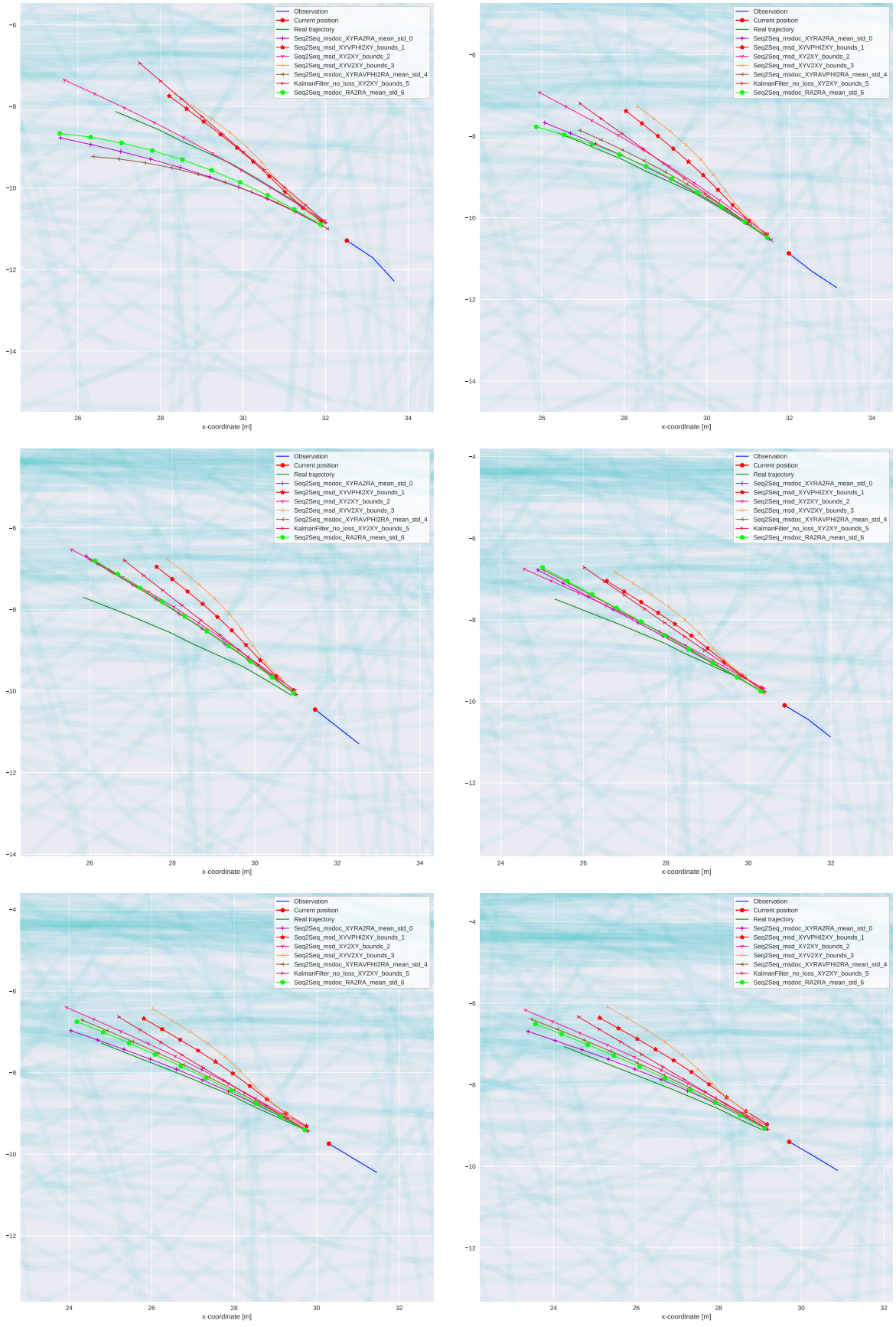


Fig. 6-3: Different learning strategies in comparison with the Kalman Filter on dataset *All* using different time steps.

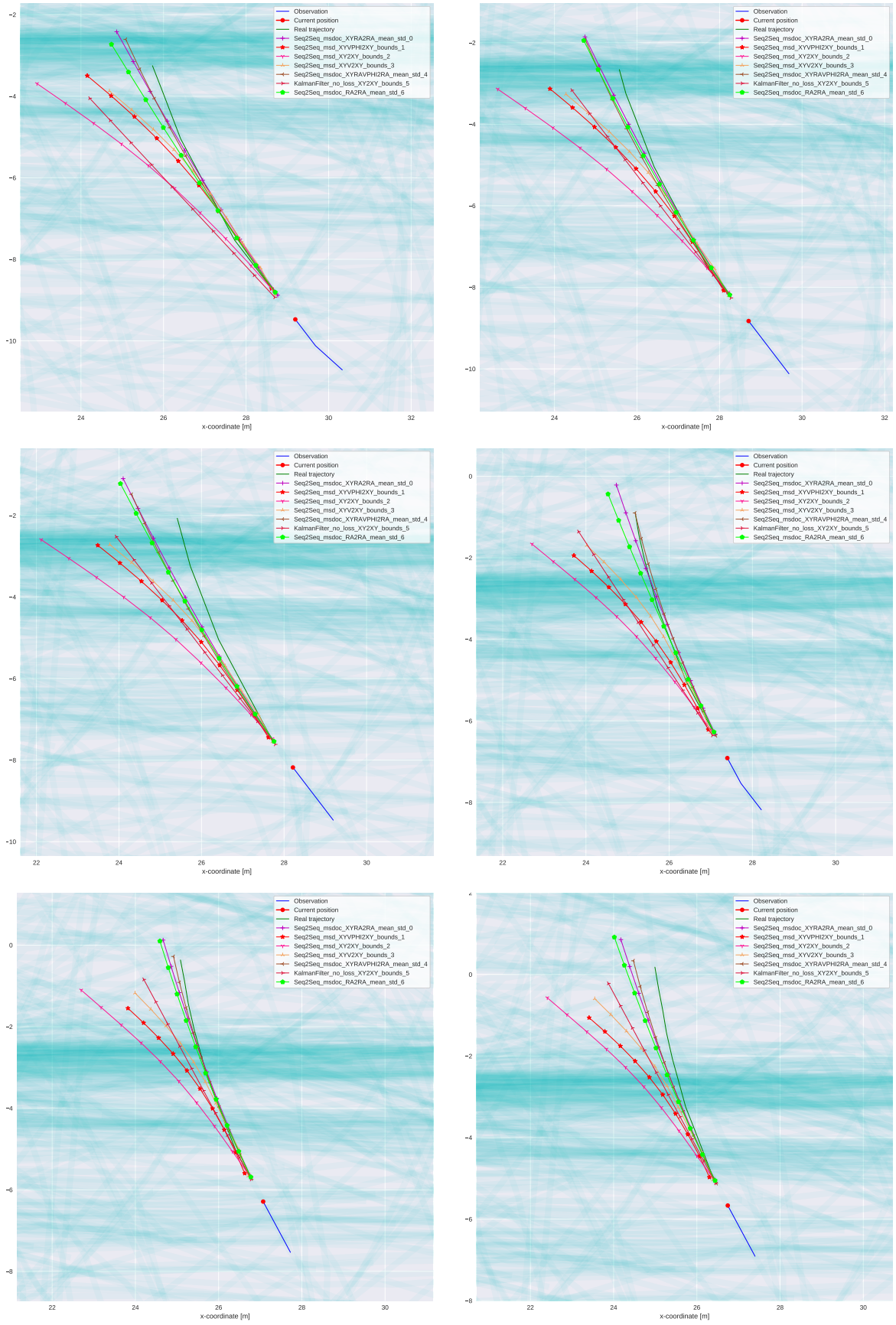


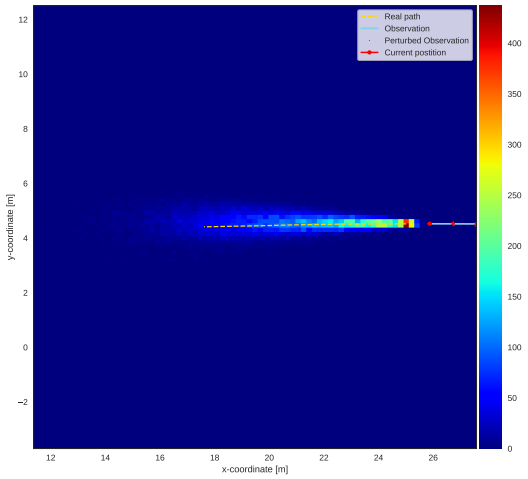
Fig. 6-4: Different learning strategies in comparison with the Kalman Filter on dataset *All* using different time steps.

6.3 Prediction Stability

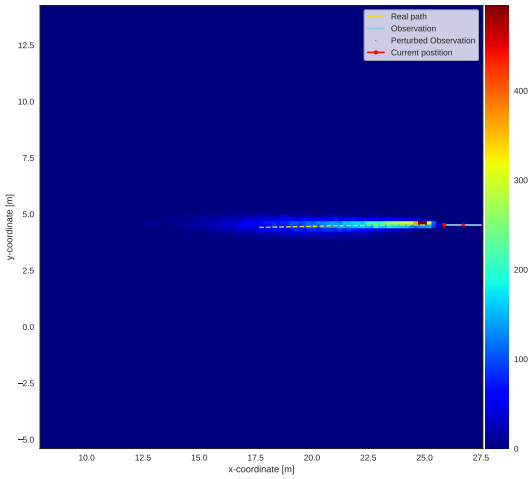
The prediction stability analysis is performed by using the presented uncertainty analysis in section 5.6. Each learning strategy was investigated with the best fitting parameter set. The resulting stability heat maps were manually examined. It was generally observed that the model prediction did show some small instabilities. Perturbation in the input of the model leads to a uncertainty in the output that is visualised in the following as a heat map. These heat maps have often a conical shape. A few chosen examples of the stability visualisations are depicted in Figure 6-5. Every different learning strategy obtained the identical input trajectories. The *All* dataset was used to train these models.

In all these cases the distribution is located in the close proximity of the ground truth. Only small variations can be observed. The strategies that map to $R\alpha$ tend to have a slight bigger conical shape compared to a XY mapping (cf. Figures 6-5i, 6-5iv and 6-5vi). This can be explained by the fact that the $R\alpha$ strategy reconstructs the future trajectory by extrapolating from the last known position. Hence, this approach is much more sensitive to small perturbations and errors propagate more likely compared to the other approach.

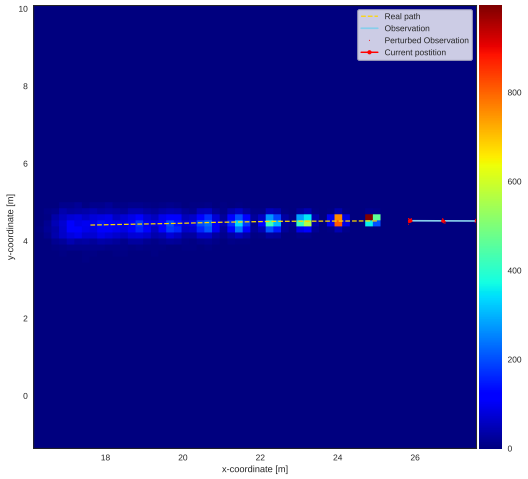
It is important to stress, that it cannot be excluded that certain model inputs could cause an unstable model response. Especially, regions with a very thin underlying training data might cause negative effects when using a local-based XY mapping. Thus, a more statistically valid method has to be elaborated and applied in future investigations in order to examine the reliability the presented prediction models.



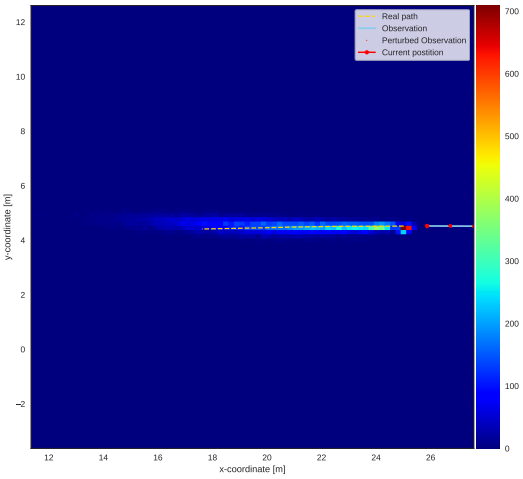
(i) $XYR\alpha \rightarrow R\alpha$



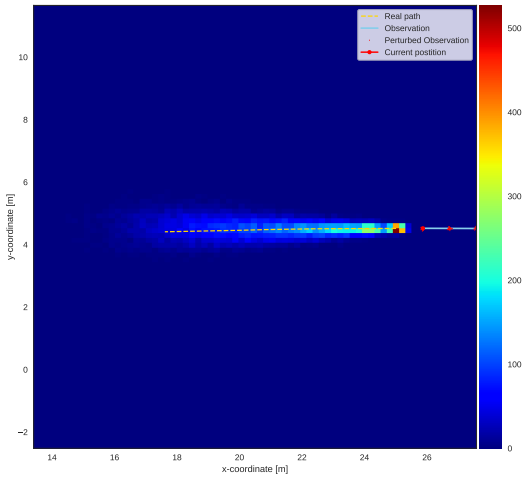
(ii) $XYV\phi \rightarrow XY$



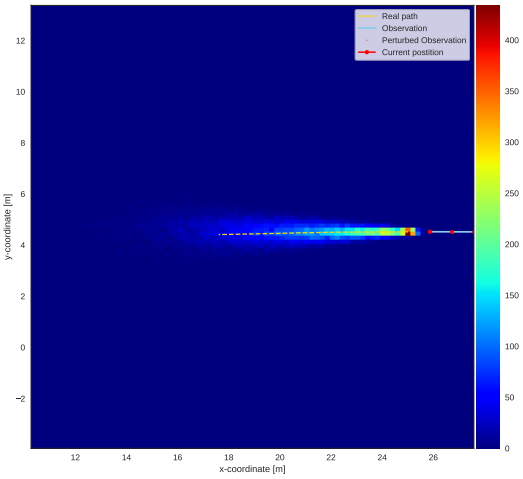
(iii) $XY \rightarrow XY$



(iv) $XYV \rightarrow XY$



(v) $XYR\alpha V\phi \rightarrow R\alpha$



(vi) $R\alpha \rightarrow R\alpha$

Fig. 6-5: Model prediction stability on dataset *All* and different learning strategies.

7 Conclusion and Outlook

This thesis presented an approach for a surrounding object trajectory prediction with recurrent neural networks at different road scenarios. State of the art Long Short-Term Memory designs where deployed in order to learn preprocessed trajectories in an end-to-end fashion. Six different learning strategies were tested and compared on four publicly available datasets. In all cases the neural network was able to predict future individual movements with a good accuracy for a given history trajectory. The overall prediction quality of the specific model configurations were hereby measured by several performance evaluation criteria. On this basis, it was qualitatively shown that the proposed methods outperformed a basic Kalman Filter.

In order to find the best hyper-parameters of the RNN, a systematic optimization has been carried out. In this respect, the effect of training parameters, loss functions and scaling strategies in combination with different learning strategies were assessed and the final model parameters were chosen in such a way that the prediction accuracy was maximized. It turned out that $R\alpha$ based strategies need the MSDOC loss function and a very small learning rate in comparison to the XY based strategies. The highest prediction quality was obtained with the $XYR\alpha \rightarrow R\alpha$ and the $R\alpha \rightarrow R\alpha$ approach. This behaviour was observed on all datasets. Further, the $XYR\alpha \rightarrow R\alpha$ mapping showed a slight better behaviour on a dense dataset in contrast to the $R\alpha \rightarrow R\alpha$ learning strategy. Here, further analyses with bigger datasets has to be performed.

Contradictory to a first assumption, adding additional information of the ego vehicle to the model input does not lead to an improvement of the prediction quality. The RNN is not able to process the velocity or the yaw rate of the observing vehicle and associate this information with the relative movement of the surrounding objects.

Further analyses demonstrated that uncertainties in the input of such model leads to small instabilities in the prediction. This behaviour was observed on manually evaluated stability visualizations. However, it would be conceivable that abnormal model input would cause inconsistent predictions. Thus, future investigations should include a statistically valid approach that determines the overall model stability.

Although interesting results have been achieved in this thesis, it has to be admitted that the presented approach are not quite appropriate for a collision detection system since the dataset does not contain any collisions. Therefore, it is a future task to validate the results of this thesis by using simulated trajectory datasets that contain additional collisions with the ego vehicle. Moreover, the presented approach does not consider the orientation of the objects. Considering the orientation of a vehicle in a prediction model could lead to a higher forecasting quality.

8 Literature

[ALA16] ALAHI, A.; GOEL, K.; RAMANATHAN, V.; ROBICQUET, A.; FEI-FEI, L.; SAVARESE, S.,
Social LSTM: Human Trajectory Prediction in Crowded Spaces
The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2016

[AMM09] AMMOUN, S.; NASHASHIBI, F.,
Real time trajectory prediction for collision risk estimation between vehicles
2009 IEEE 5th International Conference on Intelligent Computer Communication and Processing, Aug. 2009, pp. 417–422

[BAH14] BAHDANAU, D.; CHO, K.; BENGIO, Y.,
Neural Machine Translation by Jointly Learning to Align and Translate
CoRR abs/1409.0473 (2014)

[BAS00] BASHEER, I.; HAJMEER, M.,
Artificial neural networks: fundamentals, computing, design, and application
Journal of microbiological methods 43.1 (2000), pp. 3–31

[BEN12] BENGIO, Y.,
Practical recommendations for gradient-based training of deep architectures
CoRR abs/1206.5533 (2012)

[BER13a] BERGSTRA, J.; YAMINS, D.; COX, D. D.,
Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures
Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28, ICML'13, Atlanta, GA, USA: JMLR.org, 2013, pages

[BER13b] BERGSTRA, J.; YAMINS, D.; COX, D. D.,
Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures
Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013, 2013, pp. 115–123

[CHO14] CHO, K.; MERRIENBOER, B. van; GÜLÇEHRE, Ç.; BOUGARES, F.; SCHWENK, H.; BENGIO, Y.,
Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation
CoRR abs/1406.1078 (2014)

[GEI13] GEIGER, A.; LENZ, P.; STILLER, C.; URTASUN, R.,
Vision meets Robotics: The KITTI Dataset
International Journal of Robotics Research (IJRR) (2013)

[GER03a] GERS, F. A.; SCHRAUDOLPH, N. N.; SCHMIDHUBER, J.,
Learning Precise Timing with LSTM Recurrent Networks
J. Mach. Learn. Res. 3 (2003), pp. 115–143

[GER03b] GERSHENSON, C.,
Artificial Neural Networks for Beginners
CoRR cs.NE/0308031 (2003)

[GRA12] GRAVES, A.,
Supervised Sequence Labelling with Recurrent Neural Networks
Vol. 385, Studies in Computational Intelligence, Springer, 2012

[GRA13] GRAVES, A.,
Generating Sequences With Recurrent Neural Networks
CoRR abs/1308.0850 (2013)

[HOC97] HOCHREITER, S.; SCHMIDHUBER, J.,
Long Short-Term Memory
Neural Comput. 9.8 (1997), pp. 1735–1780

[JAI15] JAIN, A.; SINGH, A.; KOPPULA, H. S.; SOH, S.; SAXENA, A.,
Recurrent Neural Networks for Driver Activity Anticipation via Sensory Fusion Architecture
CoRR abs/1509.05016 (2015)

[JON15] JONES, E.; OLIPHANT, T.; PETERSON, P., et al.,
SciPy: Open source scientific tools for Python
http://www.scipy.org/, accessed 09-March-2017, 2015

[KAN17] KANG, C. M.; JEON, S. J.; LEE, S. H.; CHUNG, C. C.,
Parametric trajectory prediction of surrounding vehicles
2017 IEEE International Conference on Vehicular Electronics and Safety (ICVES), June 2017, pp. 26–31

[KAR15] KARPATHY, A.,
The Unreasonable Effectiveness of Recurrent Neural Networks
http://karpathy.github.io/2015/05/21/rnn-effectiveness/, accessed 20-December-2016, 2015

[KHO16] KHOSROSHAHI, A.; OHN-BAR, E.; TRIVEDI, M. M.,
Surround vehicles trajectory analysis with recurrent neural networks
Intelligent Transportation Systems (ITSC), 2016 IEEE 19th International Conference on, IEEE, 2016, pp. 2267–2272

[KIM17] KIM, B.; KANG, C. M.; LEE, S.; CHAE, H.; KIM, J.; CHUNG, C. C.; CHOI, J. W.,
Probabilistic Vehicle Trajectory Prediction over Occupancy Grid Map via Recurrent Neural Network
CoRR abs/1704.07049 (2017), arXiv: 1704.07049

[KRI12] KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E.,
Imagenet classification with deep convolutional neural networks
Advances in neural information processing systems, 2012, pp. 1097–1105

[LIP15] LIPTON, Z. C.,

A Critical Review of Recurrent Neural Networks for Sequence Learning
CoRR abs/1506.00019 (2015)

[MUJ16] MUJTABA, H.,
NVIDIA Pascal GPU Analysis
http://wccftech.com/nvidia-pascal-gpu-analysis/, accessed 05-March-2017, 2016

[OLA15] OLAH, C.,
Understanding LSTM Networks
http://colah.github.io/posts/2015-08-Understanding-LSTMs/, accessed 20-December-2016, 2015

[PAP02] PAPINENI, K.; ROUKOS, S.; WARD, T.; ZHU, W.-J.,
BLEU: a method for automatic evaluation of machine translation
Proceedings of the 40th annual meeting on association for computational linguistics, Association for Computational Linguistics, 2002, pp. 311–318

[RAH16a] RAHMAN, F.,
Recurrent Shop - Framework for building complex recurrent neural networks with Keras
https://github.com/datalogai/recurrentshop, accessed 20-December-2016, 2016

[RAH16b] RAHMAN, F.,
seq2seq - Sequence to sequence library add-on for Keras
https://github.com/farizrahman4u/seq2seq, accessed 20-December-2016, 2016

[RUM86] RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J.,
Parallel Distributed Processing: Explorations in the Microstructure of Cognition
In: ed. by RUMELHART, D. E.; MCCLELLAND, J. L.; PDP RESEARCH GROUP, C., Cambridge, MA, USA: MIT Press, 1986, chap. Learning Internal Representations by Error Propagation, pp. 318–362

[SUT14] SUTSKEVER, I.; VINYALS, O.; LE, Q. V.,
Sequence to Sequence Learning with Neural Networks
CoRR abs/1409.3215 (2014)

[TIM14] TIMMARAJU, A.; KHANNA, V.,
Sentiment Analysis on Movie Reviews using Recursive and Recurrent Neural Network Architectures
(2014)

[VIN14] VINYALS, O.; TOSHEV, A.; BENGIO, S.; ERHAN, D.,
Show and Tell: A Neural Image Caption Generator
CoRR abs/1411.4555 (2014)

[VIN15] VINYALS, O.; LE, Q. V.,
A Neural Conversational Model
CoRR abs/1506.05869 (2015)

[WIS17] WISTOFF, N.,

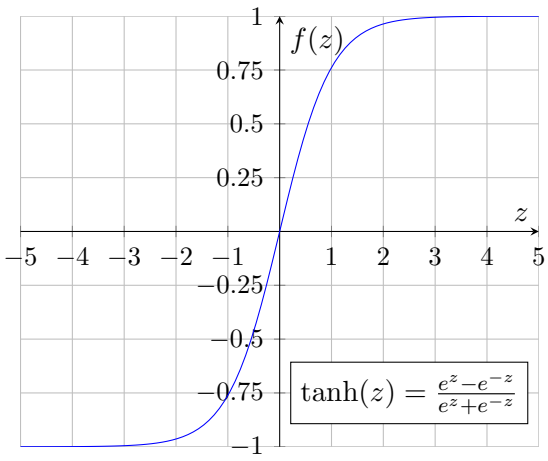
Type-Independent Prediction of Traffic Participants at Crossroads with Neuronal Networks
PhD thesis, RWTH Aachen, 2017

[YAO15] YAO, K.; COHN, T.; VYLOMOVA, K.; DUH, K.; DYER, C.,
Depth-Gated LSTM
CoRR abs/1508.03790 (2015)

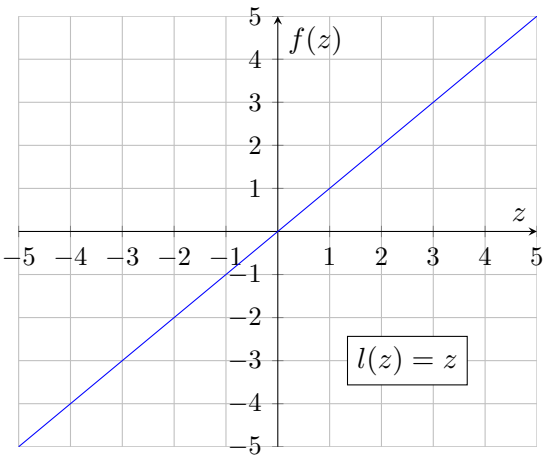
9 Appendix

9.1 Attachment B: Neural network activation functions

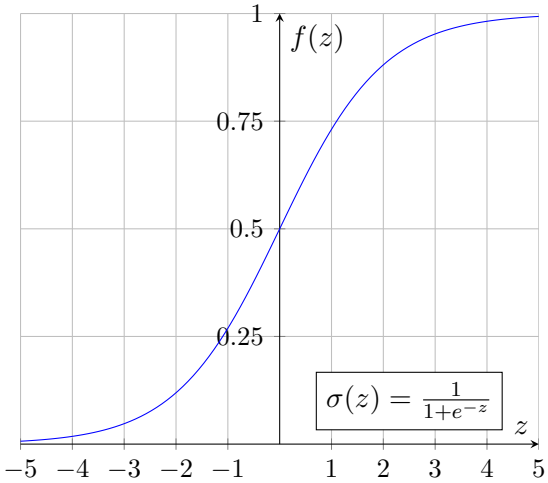
tanh



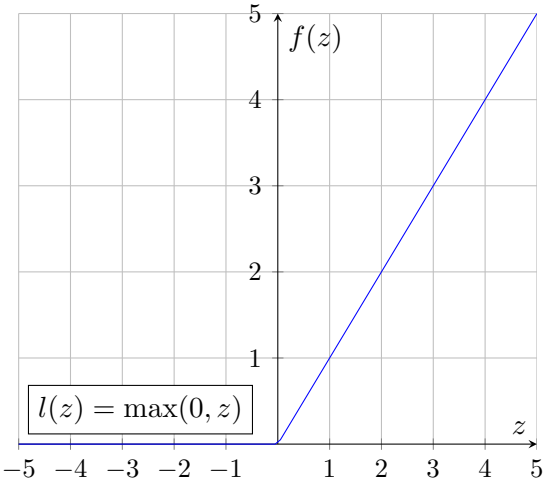
linear



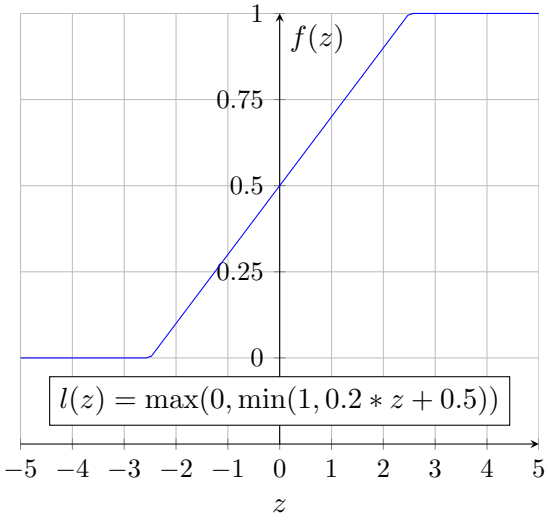
sigmoid



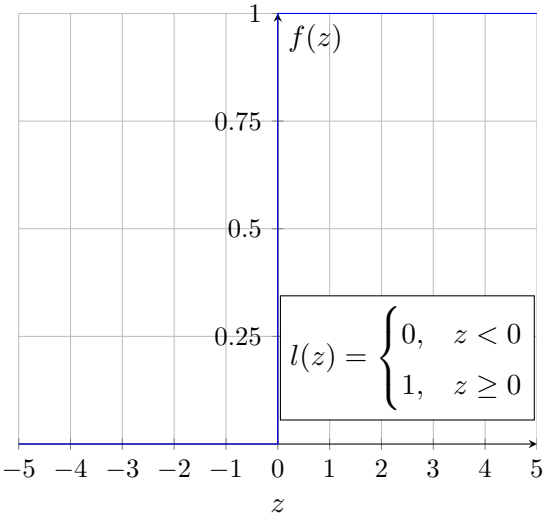
ReLU



Hard sigmoid



Heaviside step function



9.2 Attachment C: Loss functions

The loss functions defines the fitness of the prediction $\hat{Y}$ on the true datapoints Y. A *batch* of predictions of a the recurrent net used is a three-dimensional tensor of shape $Y \in \mathbb{R}^{S\times N\times 2}$ and $\hat{Y} \in \mathbb{R}^{S\times N\times 2}$ respectively. With S the number of samples and N the length of the sequences. The two rows in the third dimension represent the x and y-coordinates of one trajectory. Consequently, a sample trajectory is a two-dimensional matrix. This notation also applies for a *radius-alpha* vector $Y_i^{r\alpha}$:

$$Y_i = \begin{pmatrix} x_i^1 & y_i^1 \\ x_i^2 & y_i^2 \\ \vdots & \vdots \\ x_i^{N-1} & y_i^{N-1} \\ x_i^N & y_i^N \end{pmatrix} \in \mathbb{R}^{N\times 2}. \quad \text{Eq. 9-1} \qquad Y_i^{r\alpha} = \begin{pmatrix} r_i^1 & \alpha_i^1 \\ r_i^2 & \alpha_i^2 \\ \vdots & \vdots \\ r_i^{N-1} & \alpha_i^{N-1} \\ r_i^N & \alpha_i^N \end{pmatrix} \in \mathbb{R}^{N\times 2}. \quad \text{Eq. 9-2}$$

Mean Squared Displacement

$$\mathcal{L}_{MSD}(\hat{Y}, Y) := \frac{1}{S}\sum_{i=1}^{S}\sum_{k=1}^{N}\left(x_i^k - \hat{x}_i^k\right)^2 + \left(y_i^k - \hat{y}_i^k\right)^2 \quad \text{Eq. 9-3}$$

Mean Final Displacement

$$\mathcal{L}_{MFD}(\hat{Y}, Y) := \frac{1}{S}\sum_{i=1}^{S}\sqrt{\left(x_i^N - \hat{x}_i^N\right)^2 + \left(y_i^N - \hat{y}_i^N\right)^2} \quad \text{Eq. 9-4}$$

Mean Displacement

$$\mathcal{L}_{MD}(\hat{Y}, Y) := \frac{1}{S}\sum_{i=1}^{S}\sum_{k=1}^{N}\sqrt{\left(x_i^k - \hat{x}_i^k\right)^2 + \left(y_i^k - \hat{y}_i^k\right)^2} \quad \text{Eq. 9-5}$$

MSDOC

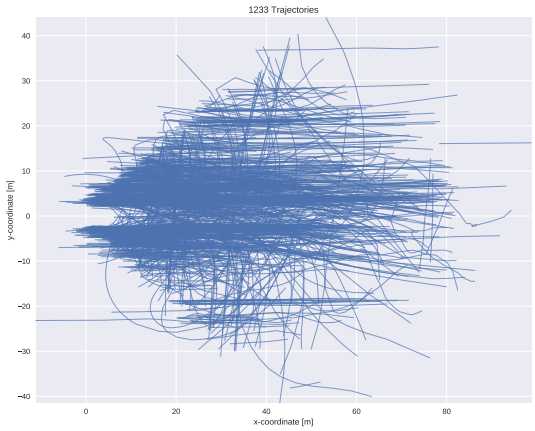
$$\mathcal{L}_{MSDOC}(\hat{Y}^{r\alpha}, Y^{r\alpha}) := \frac{1}{S}\sum_{i=1}^{S}\sum_{k=1}^{N}\Bigg(\left(\sum_{k=0}^{j}\hat{r}_i^k\cos\left(\sum_{l=0}^{k}\hat{\alpha}_i^l\right) - \sum_{k=0}^{j}r_i^k\cos\left(\sum_{l=0}^{k}\alpha_i^l\right)\right)^2 + \left(\sum_{k=0}^{j}\hat{r}_i^k\sin\left(\sum_{l=0}^{k}\hat{\alpha}_i^l\right) - \sum_{k=0}^{j}r_i^k\sin\left(\sum_{l=0}^{k}\alpha_i^l\right)\right)^2\Bigg) \quad \text{Eq. 9-6}$$

9.3 Attachment D: Dataset details

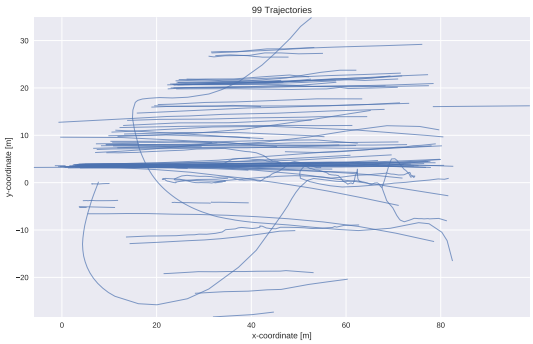
Dataset	Number of trajectories	Average trajectory length $[m]$	$v_{avg}[\frac{km}{h}]$	Sampling rate $[Hz]$	Data points
All	1233	33.22	33.53	9.66	52655
Road	99	47.30	65.86	9.66	4355
City	586	32.89	33.12	9.66	25759
Residential	548	31.04	28.12	9.66	22541

Fig. 9-1: Dataset statistics

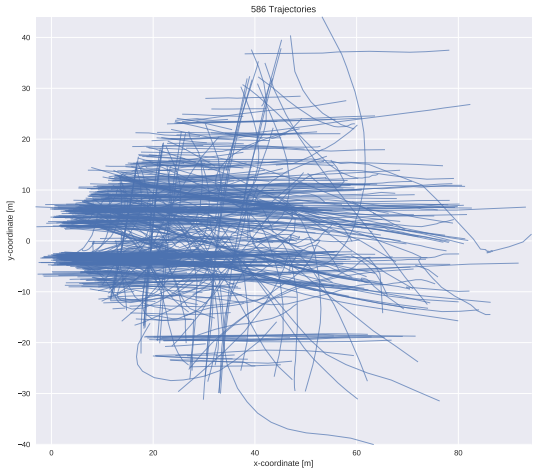
9.3.1 Dataset Visualisation



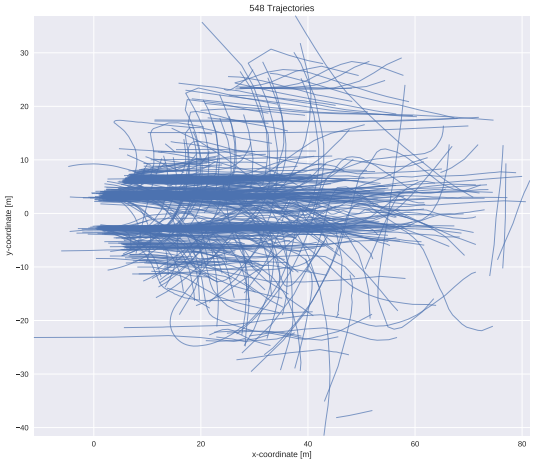
(i) Dataset All



(ii) Dataset Road



(iii) Dataset City



(iv) Dataset Residential

Fig. 9-2: Visualisation of the datasets All, Road, City and Residential