RWTH Aachen University
Faculty of Mechanical Engineering

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

Bachelor Thesis

Continuously Learning Prediction of Pedestrian Movements at Intersections with Recurrent Neural Networks

submitted by:

Till Beemelmanns, matriculation number: 32 06 02

supervised by:

Julian Bock, M.Sc.

Aachen, 30 March 2017

Contents

1 Introduction 7
1.1 Content and Structure 8
2 State of the Art 9
2.1 Prediction of Pedestrian Trajectories 9
2.2 Prediction of Pedestrian Trajectories from Driver’s Perspective 10
2.3 Related models for Sequence Prediction 12
2.4 Continuously Learning Neural Networks 13
3 Artificial Neural Networks 14
3.1 Background 14
3.2 Feedforward Neural Networks 15
3.2.1 Training and Backpropagation 17
3.2.2 Dropout 19
3.3 Recurrent Neural Networks 20
3.3.1 Simple Recurrent Neural Networks 21
3.3.2 Training of Recurrent Networks 22
3.3.3 Long Short Term Memory Networks 23
3.3.4 Bidirectional Recurrent Neural Networks 26
3.3.5 Stacked Recurrent Neural Networks 27
3.4 Prediction Network Architectures 28
3.4.1 Sequence to Sequence Model 30
3.4.2 Simple Sequence to Sequence Model 31
3.4.3 Sequence to Sequence Model with Peek 31
3.4.4 Attention Sequence to Sequence Model 31
4 Research Objective and Approach 33
4.1 Analysis of Existing Prediction Models 33
4.2 Research Questions 34
4.3 Approach 35
4.4 Definition of Performance Evaluation 36

5 Model Design and Implementation 38
5.1 General Code Setup 38
5.2 Datasets 39
5.3 Data Pre-processing 40
5.4 Model Details 42
5.5 LSTM Configuration Details 43
5.6 Training & Evaluation 43
5.7 Hyperparameter & Hyperparameter Tuning 44
5.8 Parameter Analysis 45
5.9 Stability Analysis 46
5.10 Continuous Learning 46
6 Results and Evaluation 49
6.1 Initial Hyperparameter Parameter Search 49
6.2 Architecture Comparison 50
6.3 Optimiser Comparison 50
6.4 Comparison with Baseline Models 51
6.5 Parameter Dependencies 57
6.6 Prediction Stability 61
6.7 Incorrect Predictions 64
6.8 Continuous Learning 64
6.8.1 Case “Construction Work” 64
6.8.2 Case “Blocked Road” 69
6.8.3 Evaluation Continuous Learning 70
6.9 Model Evaluation Latency 71
7 Conclusion and Outlook 72
8 Literature 74
9 List of Abbreviations 82
10 List of Symbols 83
11 Appendix 84
11.1 Attachment A: Default Configuration File 84

11.2 Attachment B: Neural network activation functions . . . 85
11.3 Attachment C: Loss functions . . . 86
11.4 Attachment D: Dataset details . . . 87
11.4.1 ETH Datasets . . . 87
11.4.2 Vissim Datasets . . . 88
11.4.3 Laser Datasets . . . 89
11.4.4 Aldenhoven Testing Center Dataset . . . 90

1 Introduction

Approximately 620.000 *Vulnerable Road Users* die every year on world wide roads [WOR17]. To remedy this situation and reduce the number of people affected, vehicle manufacturers have implemented active and passive pedestrian safety systems. Nevertheless, heavy accidents frequently occur in urban intersection scenarios which can be considered as one of the most complex driving scenarios [ADA17b]. An urban scenario is usually difficult to capture with the sensor information of a single car. Due to the existence of buildings, trees and other vehicles, the visual-, laser- and radar-based onboard sensor systems have a limited effective range. But, the precise knowledge about other traffic participants is essential for ADAS to securely evaluate the current situation and determine an optimal reaction. In order to overcome locally limited detection ranges, cooperative perception systems are currently focused in research [SEE14] [GMB16]. An elevated position of the infrastructure sensors should ensure a complete coverage of an urban intersection and enables seamless tracking of arbitrary road users. Currently developed *V2X* communication technology such as IEEE 802.11p or LTE-V2X could be used for an effective traffic information broadcasting from stationary infrastructure to vehicle and vice-versa [BOH08]. Consequently, locally available knowledge about a situation at an intersection of a single road user can be merged with the broadcasted information for road safety purposes and efficient trajectory planning.

Automotive companies and research institutes have started several R&D projects that investigate the potentials of infrastructure-based safety systems. For example, the *Ko-FAS* research initiative (cooperative driver assistance systems), which is supported by several automotive companies, investigates the potentials of an intelligent intersection that is equipped with a network of distributed sensors [MEI14]. Further projects are SADA (Smart Adaptive Data Aggregation) [DEU17], *Bosch Local Clouds* [GMB16] and *I2EASE* (Intelligence for efficiently electrified and automated driving through sensor networking) [I2E17].

Intelligent intersections that are equipped with permanently active traffic sensors are ideal for data-driven approaches due to continuous data collection. Information about positions, movement dynamics and other specific data of road users can be gathered over a long period of time, enabling predictions of future situations with the help of machine learning algorithms. The precise anticipation of future positions would give safety assistance systems and vehicle drivers more time to react to dangerous situations and thereby prevent possible collisions with other road users. Beside road safety aspects, further applications of movement prediction is the improvement of urban traffic flow and the development of intelligent automated driving functions.

For automated driving functions, the prediction of pedestrians in urban scenarios is particular relevant. Pedestrians can be easily overseen or obscured by obstacles and are due to their nature unprotected. A collision between vehicles and pedestrian often results in fatal injuries. On german roads, 95% of all fatal pedestrians accidents in the year 2015 where found at urban scenarios [ADA17c] and further the total number of injured or killed pedestrians has slightly increased in the last 4 years [ADA17a]. Therefore, there exists the need to develop new safety systems that are able to predict human movements at poorly observable urban scenarios with the help of infrastructure sensors and predictive machine learning. Consequently, in this thesis the prediction of future trajectories of pedestrians with the help of machine learning algorithms will be investigated.

Traffic flow and movement patterns at urban intersections undergo sometimes temporary changes. Road works, construction work, blocked roads or parades could temporarily influence the pedestrians movement behaviour. A static prediction model that learned movement characteristics by a fixed dataset from a previously captured time interval could fail in these cases. A reliable prediction model that is also capable of dealing with extraordinary events needs a self-learning mechanism that captures temporal evolution of a specific scene. In addition to that, the continuous observation of an intersection with stationary sensor leads to a permanently increasing dataset, that could make machine learning algorithms inefficient to train as local computing power at the intersection might be limited. Thus, it is necessary to develop an algorithm that is able to apply a data stream on a machine learning prediction model in such a fashion that already learned and validated pieces of information are preserved, whereas unknown information are added to a dynamic training dataset.

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 pedestrian motion. A state of the art review about Artificial Neural Networks can be found in section 3. Especially, Recurrent Neural Networks and neural networks for prediction tasks are discussed which where later implemented for the human motion anticipation. 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 pedestrian movements is given, followed by a presentation and discussion of machine learning techniques for sequence prediction tasks. Section 2.1 deals with different general approaches for human motion modelling. In this section, social forces and machine learning models are reviewed. Predictive models, that anticipate pedestrians future movement based only on the sensor measurements of a vehicle, are described in section 2.2. In the following section 2.3, different approaches that are not directly intended to predict pedestrian movements, but which are capable to model sequential predictions with machine learning algorithms in a different context, are discussed. Finally, section 2.4 gives an overview on existing techniques for continuously learning neural networks.

2.1 Prediction of Pedestrian Trajectories

Since automated driving robots and advanced driver assistance systems (ADAS) will play an increasingly important role in our daily lives, anticipating pedestrians future movements is important to improve road safety and trajectory planning [BRO16] [KEL14] [TAM10]. Human movements patterns are often uncertain and they depend from many individual influencing factors. Thus, it is a challenging task to design a model that is able to forecast future movements of pedestrians. In the following, an overview of literature that deals with the prediction of human motion is given.

At the beginning of the 1990s, scientists developed first pedestrians and traffic models which were inspired by physical gas-kinetics [HEL90]. One of the first *social-forces* models was introduced by Helbing et al. [HEL97]. This simple model described the social-forces, similar to energy potentials, that have usually influence on one pedestrian such as the desired destination of one individual, the distance to other pedestrians with respect to the private sphere and an attraction effect. Helbing et al. demonstrated in computer simulations, that their model was able to describe nonlinear interactions of pedestrians.

Nowadays, social-forces models are still a matter of interest to researchers. In the year 2010, Luber et al. [LUB10] developed a more sophisticated approach that considers beside usual social influences between humans also environmental constraints. Yamaguchi et al. [YAM11] modelled successfully movement patterns of groups with social forces. Handcrafted energy potentials regarding, social grouping, collision avoidance, intended destination of one individual and environment constraints where joined and evaluated in a energy minimising fashion in order to make a prediction for future time steps. A social-aware robot navigation system for collision avoidance for an shared environment was proposed in [TAM10]. The authors of this article used a social-force method for the estimation of the pedestrians behaviour and used this information to plan secure trajectories for robots. Inspired by the social-forces model, Robicquet et al. [ROB16] used an machine learning approach that utilises real pedestrian movement data in order to train a social sensitivity feature. This energy potential describes in which manner pedestrians and other individuals avoid each other and this leads in contrast to previous models to a higher flexibility in interaction possibilities. Thus, the model is capable to differentiate between different navigation styles of the individuals such as aggressive, mild and neutral behaviour. Further, the captured navigation styles are used for improved both forecasting models and a multi-target tracking.

A data driven approach that makes use of drone video footages of dense places was recently proposed by Ballan et al. [BAL16]. The researchers formulated an observed scene as a discretised navigation map that is assigned with rich information about movement behaviours of different road users. A stochastic model computed for each discrete element of the map the probabilities for moving from the current element to one of the neighbour elements with a probable velocity. In addition to that, specific functional properties of the scene such as grass, street or sidewalks are also embedded in the model. These properties where identified by an visual scene algorithm. With a given initial position and speed for a pedestrian it is possible to estimate its future path by evaluating the probabilities over the navigation map.

Walker et al. [WAL14] proposed a pure visual idea that relays on a large database of scene videos. Their public available framework, analyses an arbitrary road scene (e.g. a parking lot) by considering mid-level moving elements (e.g. cars, pedestrians) on a static background. The change of visual appearance in time is trained in an unsupervised manner with a decision theoretic algorithm. The visual prediction considers for a single forecast of one target object also surrounding moving elements. This technique allows a visualisation of possible future movements for traffic participants with a hallucinating image processing and with a prediction heat map.

In 2016, Alahi et al. [ALA16] proposed a deep learning model that is capable of predicting human movement with a so called "Social-LSTM". In their approach they considered that the movement of a person in a crowded scenario is usually influenced by its direct neighbours. In contrast to other social models they did not used handcrafted social forces functions, but they designed a new deep learning end-to-end architecture that allows an interaction between spatially proximal sequences through a pooling layer. The pooling layer ensures that a Long Short-term Memory (LSTM) cell has access to the hidden-states of all other LSTMs in a specific radius and this information is used for the prediction of the next time step. The authors of this paper evaluated their model on publicly available pedestrian tracking datasets and they could show that the deep learning network was able to anticipate future movements of individuals caused by social interactions among them.

2.2 Prediction of Pedestrian Trajectories from Driver's Perspective

The precise knowledge of the location and future estimated position of pedestrians at intersections gives car drivers and ADAS more time to react to dangerous situations and thereby prevent possible collisions. Automobile manufacturers have developed a variety of pedestrian models based on car's sensors. In the following, a review about different approaches with camera and laser measurements is given.

Brouwer et al. [BRO16] compared several existing pedestrian motion models for collision avoidance systems. The researches classified each approach into one of four classes. For each of these classes, one model is selected, implemented and benchmarked. To ensure comparability, all models generate a probability grid for an identical set of recorded situations. Thereby, each cell of the grid is associated with a probability that describes the possible presence of the pedestrian in this cell with respect to time. *Dynamic based Pedestrian Models* (class I) consider measurement of pedestrian's dynamics in order to estimate a maximal area of the possible future position. The authors used for their comparison a simple motion model with fixed acceleration. This results in a circular probability map around the walker. Second class models are described as *Physiological Grid Models*. In addition to the pedestrian's position, further properties of the human movement are used. Angular velocity, movement direction and empirically measured maximal and minimal accelerations for different situations are taken into account. As a result, the generated probability area has a conical shape. *Head Orientation Grid Models* (class III) assume that future pedestrian's motion depends on the direction it is looking at. A computer vision algorithm determines the head orientation and the model uses this information jointly with the pedestrian's dynamics to estimate the position probabilities. Uncertainties of the prediction are modelled by a Gaussian function that scales the probability area proportionally to the prediction time. An approach by Rehder et al. [REH15] can be categorised as a class III model. The researchers proposed a model that fuses environmental influence factors, pedestrian orientation and dynamics in order to solve a trajectory planning problem. Visual sensors capture the pedestrian's current state and estimate on this basis the probability distribution over a discretised domain. This is realised by a Monte Carlo particle filter with respect to uncertainties in the visual measurements.
Finally, class IV models (*Pedestrian Moving Behaviour Grid*) take typical human movements into consideration. The human behaviour is influenced by proximal static environmental features such as obstacles, sidewalks or crosswalks. A trajectory planning algorithm uses environmental information to calculate the distributed probabilities.
Brouwer et al. evaluated the different models and proposed a new model architecture that combines probabilities taken from all models. Consequently, the fused probability grid is computed by the weighted sum of each of the grids. It was shown that the fusion approach performed better than a single model.

A different stereo-visual approach based on previously labelled motion data was proposed by Keller et al. [KEL14]. The researchers considered in their publication the classification problem of a pedestrian that either crosses the road or stops walking. The only sensor source is a stereo video camera from driver's perspective. A visual feature extraction algorithm computes the position and motion of the pedestrian. Consequently, a probabilistic matching framework is applied on the current observed trajectory with respect to previously captured trajectories stored in a database. With the help of the stored trajectories, the prediction model classifies the motion of the pedestrian and is able predict its future path.

Another predictive model for road crossing was introduced by Hashimoto et al. [HAS15]. The authors of this article focussed on an urban signalised intersection scenario and used a machine learning network for their approach. A *Dynamical Bayesian Network* (DBN) anticipates pedestrian's behaviour based on current traffic signals, visual data, position and velocity of the pedestrian. All information streams were jointly merged in a stochastic manner. The prediction network was able to estimate pedestrian's decision after observing the input information streams for two seconds. Nevertheless, the data generation for the experiments in this paper where captured on a single intersection and the motion prediction considers only two states: *waiting* and *crossing*.

2.3 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 labeled 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.

Even in sports sciences deep learning algorithms are applied. Shah et al. [SHA16] trained a RNN with real tracking data from basketball trajectories in order to determine whether a three-point shot hits the bucket or not. In this classification problem, the researchers compared a RNN that was trained with sequential X-Y-Z coordinates with a complex handcrafted model that took in addition to the positional data also features such as angle, velocity and distance to the bucket as an input. It was reported that the RNN outperformed the traditional model and the RNN was able to predict with a high probability a miss or a hit after observing the trajectory for 500 milliseconds.

2.4 Continuously Learning Neural Networks

Neural networks are commonly trained on fixed datasets. In the neural network research community, static publicly available datasets are used to compare the performance of different net architectures and to develop new models [PEL09] [ROB16]. However, the human brain learns continuously something new, since we live a permanently changing world [KÄD16]. In context of this thesis, one can ask for example the question how to fit a predictive neural net when a specific urban scenario is perturbed due to construction zones or blocked roads. Only a few studies exist that deal with neural nets that are trained with ongoing partially changing or growing datasets.

Xiao et al. [XIA14] considered a convolutional neural network for image classification with an incrementally increasing labeled dataset. In their new approach, an algorithm hierarchically expands the convolutional network leading to bigger net capacities. A study on continuously learning neural nets with incoming data streams was performed by Käding et al. [KÄD16]. The researchers investigated the impact of training parameters for newly added image data and their corresponding labels. They found out, that the effort of retraining a neural net with new data can be decreased by reducing the number of weight update iterations. Furthermore, Käding et al. state that neglecting already known data during retraining leads to overfitting of the new added data. Thus, robust retraining in a continuous fashion should be performed with a fraction of new and old data.

The above presented works for continuous learning concerned only convolutional neural networks for image labelling tasks. Hence, a similar analysis for recurrent neural networks for sequence generation is performed in section 5.10.

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 3.4 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. In a biological network this activation is an electric or chemical impulse whereby the signal in an artificial network is simulated by a real number, mostly in range between -1 and 1 [BAS00]. 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. In doing so, we adopt the notation that was previously used in literature [HOC97] [LIP15]. 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 weights $w_{jj'}$ and the bias b_j form the parameters of the neuron that are randomly assigned at initial state. It is also common to omit the bias term and to consider an additional input x_0 with fixed value 1.0. This results in a additional weight w_{j0} plays the role of b_j. The activation function f_j or *transfer function* is often a $\mathrm{sigmoid}$, $\tanh$ or ReLu function (cf. Attachment 11.2). These functions commonly are nonlinear, monotonic and continuously differentiable which are important properties for computing the derivative during the training process. In scenarios where the neural net is trained to classify an input usually the $\mathrm{softmax}$ function is applied as it considers K outcome possibilities and computes the normed probabilities for each of them.

$$\mathrm{softmax}(\vec{z})_k = \frac{e^{z_k}}{\sum_{k'=1}^{K} e^{z_{k'}}} \; for\ k = 1, \ldots, K \qquad \text{Eq. 3-3}$$

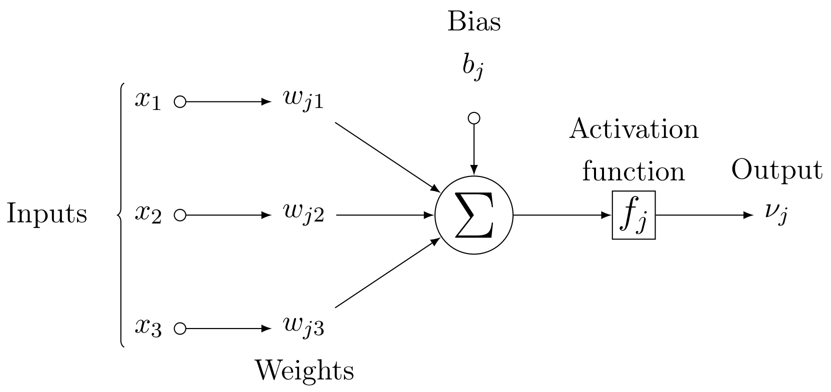


Fig. 3-1: The output of the artificial neuron j is a weighted sum of its inputs which is put through an activation function. In this example the neuron has three input edges. In theory, the number of input signals is not bounded. Figure adapted from [GER03b].

Moreover, in some cases a linear function is also used as an activation function especially for regression tasks. It is worth to mention that in the later implementation a slight different $\mathrm{sigmoid}$ function is used which is called $\mathrm{hard\ sigmoid}$. This has the benefit that it does not involve the computationally expensive exponential function (cf. Attachment 11.2) [COU15] [THE16] . Since activation functions (or the derivatives) are evaluated very often during the training, this approximation decreases computational runtime.

There exists a variety of possibilities to arrange and structure artificial neurons and the information flow between them. In this way it is possible to differentiate between several types of ANNs. These types have typically strong different properties and show their best performance in a specialised task. For example, *convolutional neural networks* achieved state of the art performance in computer visions tasks [SER11], where *recurrent neural networks* were able to outperform traditional natural language processing models [CHO14] [SUT14].

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 simple 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. This successive evaluation of the neural net is referred to as the *forward pass*.

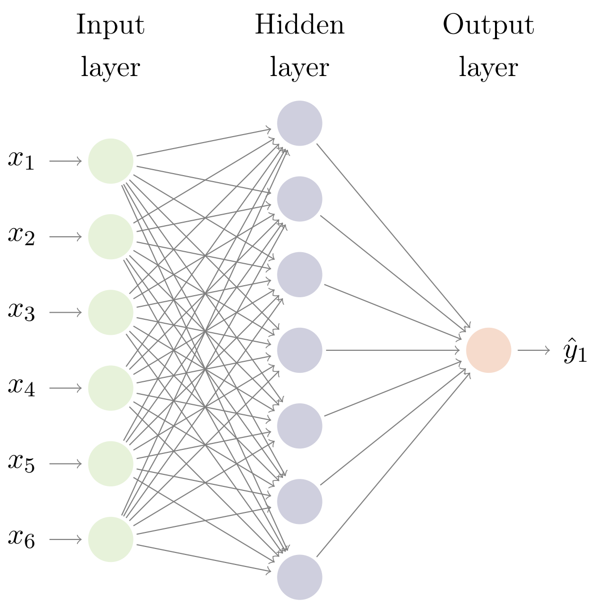


Fig. 3-2: Simple fully connected feedforward neural net with six input nodes and one output unit.

FNN are used for regression and classification task and are usually trained with the supervised learning methodology. Hornik proved in the *universal approximation theorem* [HOR91] that a standard feedforward network is an universal approximator with arbitrary accuracy for *any* continuous, bounded function with the assumption that the net consists of a sufficient number of hidden units. The proof holds for almost any common activation function.

Neural networks with more than one hidden layer are called *Deep Neural Networks* (DNNs) or *Multi Layer Perceptrons* (MPL). On the one hand, additional layers in a deep net allow to model more complicated dependencies between input and output, but on the other hand, deep nets are more difficult to train [BA14]. Figure 3-3 depicts a deep fully connected network with three hidden layer.

A very popular subclass of feedforward neural nets are *Convolutional Neural Networks* (CNNs), that are designed to capture local features of the input, especially for computer vision data. CNNs reached astonishing results in image classification and human face recognition [KRI12] [SER11] [LAW97].

Although we distinguish here between different types of neural nets, feedforward and recurrent

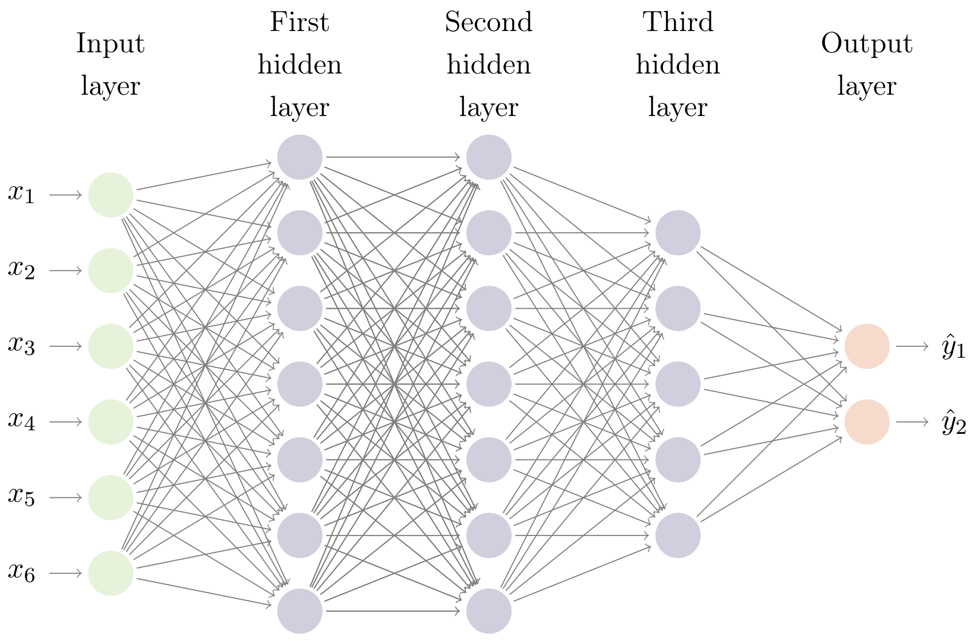


Fig. 3-3: Fully connected deep feedforward neural net with three hidden layers and two output units. Note that there is no connection between units from the same layer.

neural nets, it is also possible to combine them into one architecture, as shown by the image caption generator established by Vinyals et al. [VIN14]. Vinyal uses a convolutional and a recurrent neural net, linked to one encoder-decoder architecture (cf. Section 3.4).

3.2.1 Training and Backpropagation

Supervised training for neural nets describes the method how the weights of the neurons are modified in a such way that the nets output $\hat{y}$ fits the target observation y of the input x. Hereby, it is necessary to define a norm that measures the accuracy between these two quantities which is usually denoted by *loss function* $\mathcal{L}(\hat{y}, y)$ (different loss functions that are used later in this thesis can be found in Attachment 11.3).

There exist diverse training methods, but the most successful and most applied algorithm is *backpropagation* [RUM86]. Since a neural net can be considered as a big composite function, it is possible to compute the partial derivative of the loss function after each parameter of the neurons $\frac{\partial \mathcal{L}(\hat{y},y)}{\partial w_{ij}}$. This is effectively done by the recursive application of the chain rule beginning at the output layer and progressing backwards through the hidden layers until the input layer is reached.

In the first step of the backpropagation algorithm the forward pass is executed by feeding an input x to the input neurons and compute for every hidden unit its output ν_j and also the nets prediction $\hat{y}$. As it is required that the activation function is differentiable, it is possible to compute for each output neuron k its *error signal* δ_k,

$$\delta_k = \frac{\partial \mathcal{L}(\hat{y}_k, y_k)}{\partial \hat{y}_k} f_k'(a_k). \qquad \text{Eq. 3-4}$$

For the sake of convenience, a simple squared error as a loss function is chosen,

$$\mathcal{L}(\hat{y}, y) = \frac{1}{2}\sum_k (y_k - \hat{y}_k)^2 \qquad \text{Eq. 3-5}$$

whose straightforward derivative is inserted in equation 3-4:

$$\delta_k = (y - \hat{y}) f'_k(a_k). \qquad \text{Eq. 3-6}$$

The error signal for each hidden neuron can be determined in a recursive process with the help of the error signals of the units from the previous layer

$$\delta_j = f'_j(a_j) \sum_k \delta_k w_{kj}. \qquad \text{Eq. 3-7}$$

The formulations of δ_k and δ_j are now used to derive the desired equation for the loss function with respect to each weight. First we can express the partial derivative of $\mathcal{L}(\hat{y}, y)$ by using the chain rule

$$\frac{\partial \mathcal{L}(\hat{y}, y)}{\partial w_{ij}} = \frac{\partial \mathcal{L}(\hat{y}, y)}{\partial \nu_j} \frac{\partial \nu_j}{\partial a_j} \frac{\partial a_j}{\partial w_{ij}}. \qquad \text{Eq. 3-8}$$

The last factor of this term can be rewritten with equation 3-1 as

$$\frac{\partial a_j}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \left(\sum_{j'} w_{jj'} x_{j'} + b_j \right) = x_{j'} \qquad \text{Eq. 3-9}$$

for the nodes in the input layer and for the hidden units the derivative is equal to $\nu_{j'}$. The second term of equation 3-8 is

$$\frac{\partial \nu_j}{\partial a_j} = \frac{\partial}{\partial a_j} f_j(a_j) = f'_j(a_j) \qquad \text{Eq. 3-10}$$

and if we put equations 3-9, 3-7 and 3-6 together, we obtain finally the error function with respect to an arbitrary weight $w_{jj'}$,

$$\frac{\partial \mathcal{L}(\hat{y}, y)}{\partial w_{jj'}} = \delta_j \nu_{j'} \qquad \text{Eq. 3-11}$$

with

$$\delta_j = \begin{cases} f'_j(a_j) \sum_k \delta_k w_{kj}, & \text{for } j \text{ in hidden layer} \\ f'_j(a_j)(y - \hat{y}), & \text{for } j \text{ in output layer.} \end{cases} \qquad \text{Eq. 3-12}$$

Usually, the aim of the training procedure is to minimise the loss function. Hence, a gradient decent optimiser such as the *stochastic gradient decent* (SGD) updates each weight with *learning rate* η,

$$\Delta w_{jj'} = -\eta \frac{\partial \mathcal{L}(\hat{y}, y)}{\partial w_{jj'}} \qquad \text{Eq. 3-13}$$

with $0 < \eta \leq 1$. For faster computation, the gradient update is performed after evaluating a batch of input and target pairs using a fixed *batch size*. For simplicity, a batch size equal to one is used in equation 3-13.

The downside of the described technique is that the loss function is often non-convex with local minima. Hence, an optimisation algorithm that follows always the steepest gradient might easily get stuck without reaching the global optimum. Therefore, a large number of heuristic algorithms were developed in order to improve the standard SGD. Popular examples for these optimisers are AdaGrad [DUC11], AdaDelta [ZEI12], RMSprop [TIE12] and Adam [KIN14]. These methods make use of adaptive learning rates and a momentum that helps accelerate the convergence of the optimisation and to overcome local extrema.

Nowadays, the complex optimisation procedure is carried out by automatic symbolic differentiation for arbitrary valid net structures with highly optimised libraries such as Theano [THE16]. The computations are usually executed on GPU accelerators that enable training for big nets using a lot of data in acceptable runtime.

The training process is usually executed on a big dataset that is divided into a *training split* and a *test split*. As a rule of thumb, the size of the training data should be around 70-80% of the whole dataset, whereas the test split consists of the remaining 20-30%. The optimisation process is commonly organised in *epochs*. Executing one epoch means that the whole test dataset runs once through the net. Simultaneously, the weights are updated via backpropagation when parts of the test split with size “batch size” have passed the net. The test split is intended to evaluate the models final quality. Accordingly, the neural network is not trained on the test split.

Training is often performed with several epochs feeding the training data split repeatedly into the neural net. Choosing the right number of epochs in order to obtain an optimal fitted neural network is sometimes a balancing act. If not enough epochs are executed, the net might suffer from *underfitting* which means that the neural has not yet reached its full potential. On the contrary, when the net is trained for too many epochs it reaches a state of *overfitting*. Hence, the neural network would not be able to generalise the underlying trend of the dataset and it performs poor on the test split. Several regularisation techniques were introduced to address these problems. In the following section, the most successful method will be explained.

3.2.2 Dropout

When an artificial neural network is trained too intensively it tends to *overfit*. *Dropout* is an effective regularisation technique to overcome this problem. This methodology is likewise an artificial neuron (see above) inspired from a phenomenon out of nature [SRI14]. During sexual reproduction randomly mutated maternal and paternal gametes fuse together and form an individual with a unique combination of genes. Hence, different sets of genes of both parents are randomly combined. After the *mixability theory* [LIV10], each gene becomes more robust, since it can not rely on the presence of the same genetic set at all times [SRI14]. Thus, genetic diversity allows evolution to avoid dead-ends and to optimise fitness of species.

The dropout technique adapts the concept of the mixability theory. During a training step, each unit is randomly *dropped* with a probability of $q = 1 - p$. Every dropped node (in the hidden, input and output layer) and its corresponding connections is ignored during the ongoing forward and backward pass, which is referred to as training a *thinned* network. The “survived” nodes must instantly develop a correlation among the other randomly present units with the effect that each unit becomes more robust by reducing extraordinary strong co-adaptations to a subset of nodes. Moreover, this method initiates cooperations among a diversity of nodes. After training, the neural net is then evaluated without any dropout probability, as usual.

Srivastava et al. [SRI14] reported that adding dropout to neural nets drastically improves their performance, especially for convolutional neural networks. As shown in the same publication, dropout outperformed other regularisation methods such as weight decay, max-norm regularisation and early stopping. However, training a neural net with dropout takes 2-3 times longer than the same network without dropout.

Figure 3-4 shows a simple feedforward neural net that has a dropout probability of $p = 0.5$. Randomly chosen nodes are deactivated in order to force some of the nodes to “cooperate” with other randomly chosen units of the whole network.

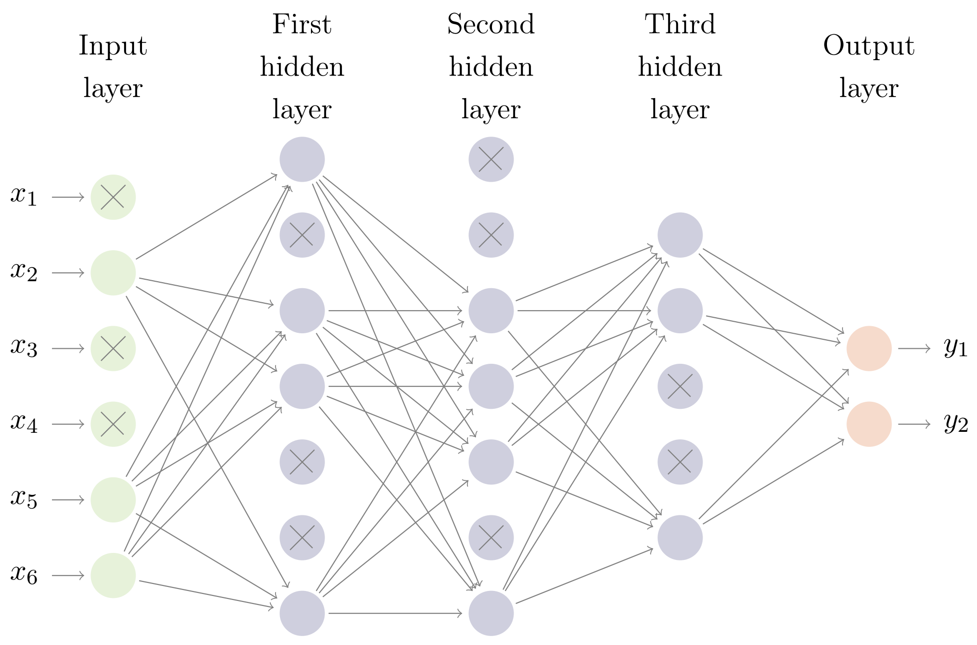


Fig. 3-4: Same deep feedforward neural network like in figure 3-3 but thinned with a dropout probability of $p = 0.5$. All units that have a cross and their connecting edges have been dropped in the current training batch. Figure adapted from [SRI14].

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 DNNs 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.

In Section 3.3.1 a simple minimalistic recurrent network is presented. The application of backpropagation for recurrent nets is shown in 3.3.2. In section 3.3.3, *Long Short Time Memory Networks* (LSTMs) modern and more complex RNN architectures that are capable of overcoming disadvantages of traditional recurrent networks are discussed. For completeness *Bidirectional recurrent neural networks* (BRNNs) are explained in section 3.3.4. Section 3.3.5 presents “deep” recurrent neural networks. 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,

$$X = (x_1, x_2, \dots x_T) \qquad \text{Eq. 3-14}$$

and the corresponding output sequence,

$$Y = (y_1, y_2, \dots y_T) \qquad \text{Eq. 3-15}$$

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

$$x_t \in \mathbb{R}^n, \quad y_t \in \mathbb{R}^n. \qquad \text{Eq. 3-16}$$

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.

3.3.1 Simple Recurrent Neural Networks

In 1990, Elman proposed the *Simple Recurrent Network* (SRN) which was able to capture structures in sequences [ELM90]. In that seminal work, Elman reported a SRN that manages to learn the correct output of a sequential *XOR* operator. Furthermore, the net was capable of predicting the likelihood of occurrences of words in a sentence. The main feature of the “Elman-network” is a *hidden context layer* that is fed by recurrent feedback which induces a time-dependent memory. The SRN is almost equivalent to the following simple architecture.

The first neural node of the basic recurrent neural cell computes at time step t with the current input x_t and the previous hidden state h_{t-1} the actual hidden state h_t (cf. equation 3-17). After that, the activation function is applied on the new hidden state leading to the output $\hat{y}_t$ (cf. equations 3-18). It is obvious to see, that an input at time $t-1$ can influence the output of $\hat{y}_t$ and the following outputs through the recurrent connections. The following equations define the described minimalistic RNN:

$$h_t = \tanh(W_{xh}x_t + W_{hh}h_{t-1} + b_h) \qquad \text{Eq. 3-17}$$

$$\hat{y}_t = \sigma(W_{hy}h_t + b_y), \qquad \text{Eq. 3-18}$$

while the weight matrices $W_{xh|hh|hy}$ and the bias vectors $b_{h|y}$ build the parameters of the cell (e.g. W_{xh} denotes the input-hidden weight matrix). During the training process these parameters are slowly modified in order to fit the likelihood of the input and output of the net. Here

σ represents the $\mathrm{softmax}$ activation function. The recurrent model is visualised by Figure 3-5, where the yellow rectangles represent the neural layers from equations 3-17 and 3-18 respectively. The recurrent process can get *unfold* across time as shown in Figure 3-6.

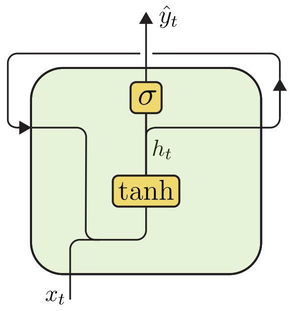


Fig. 3-5: Simple recurrent cell at time step t, biases $b_{h|y}$ are not shown. Figure adapted from Olah [OLA15].

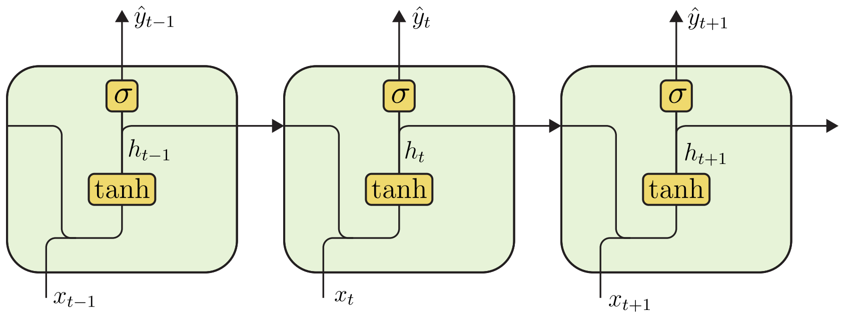


Fig. 3-6: The recurrent cell from figure 3-5 unfolded along three time steps $t-1$, t and $t+1$. Figure adapted from Olah [OLA15].

3.3.2 Training of Recurrent Networks

Training RNNs is performed using a similar procedure as training FNNs. Considering the unfolded structure in Figure 3-6, the net is basically a deep network with two hidden layers and a hidden state that is transferred from every node to the next at each time step. The gradient update algorithm *Back-Propagation Through Time* (BPTT), which was derived by Weberos [WER90], makes use of this unfolding through time. It then proceeds in the same way as training a FNN by applying back-propagation. Since the net is a nested composite function, the partial derivatives of the error $\frac{\partial \mathcal{L}(\hat{y},y)}{\partial w_{ij}}$ with respect to the weights is used to update W and b during the training.

Learning long-range dependencies with RNNs is nevertheless a difficult task. Simple RNNs suffer from the *exploding* and *vanishing* gradients problem, which has a huge negative impacts on net's performance. This well studied phenomenons were first investigated by Bengio et al. [BEN94]. They occurs during the BPTT procedure across many time steps. The temporal evolution of the gradient depends on the size of the weights and can lead either to an exponential increased or to a vanished gradient. In the first case, the weights may become oscillating, and in the second case the learning process of long sequences becomes very slow or stops working [HOC01] (cf. Figure 3-7). This has the effect that the net is unable to store and access long-range dependencies. A mathematical analysis of this shortcoming was performed by Pascanu et al. [PAS12]. In their study they derived a correlation between the biggest eigenvalue of the weight matrices and behaviour of the system. Consequently, Pascanu proposed a method that clips or regularises the gradient which would lead to a stable error signal and improved performance.

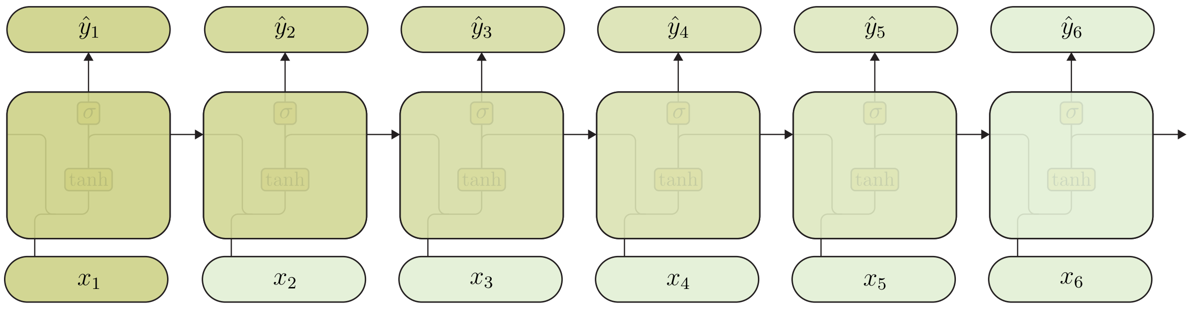


Fig. 3-7: The vanishing gradient problem occurs, if the biggest eigenvalue of the weight matrix on the recurrent edge is smaller than 1.0. This leads to a decaying influence of the input x_1 over the time. The influence is schematically represented in this figure by the shades of green (where dark green represents high influence and bright green low impact). The influence of the input at the beginning vanishes until time step six removing any association between x_1 and $\hat{y}_6$. Visualisation similar to Graves [GRA12].

To prevent the gradients from exploding, the *Truncated Back-Propagation Through Time* (TBPPT) algorithm was developed by Williams et al. [WIL90]. The algorithm “processes the sequence one time step at a time and every k_1 time steps, it runs BPTT for k_2 time steps” [SUT13]. Consequently, this technique damps back flowing error signals but also cuts long-range dependencies. The RNN architecture explained in the next section was designed to overcome these shortcomings.

3.3.3 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 RNN. 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].

The *original* Long Short Term Memory Cell was proposed by Hochreiter & Schmidhuber [HOC97]. In their first approach they worked on solving the problem of *exploding* and *vanishing* gradients which occurred by the application of Back-Propagation Through Time gradient update technique while training standard recurrent neural networks. In this first design Hochreiter et al. made use of four different elements in order to overcome these error back-flow problems. Figure 3-8 visualises the LSTM cell in the cell-state centred representation.

- *Input node:* This node computes the candidate value $\tilde{C}_t$ (using the notation of [OLA15]) for the states of the memory at time t. The value of this node is computed with the current input x_t and the previous hidden state h_{t-1} and eventually by using the $\tanh$ activation function. With the weight matrices and the bias we obtain in vector notation the explicit formulation $\tilde{C}_t = \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c)$. Note that in the original paper of Hochreiter & Schmidhuber [HOC97] a $\mathrm{sigmoid}$ function was applied, but we use here the $\tanh$ activation according to recent studies [LIP15] [ZAR14].

- *Input gate:* The gate i_t was introduced to protect the memory contents from currently unwanted input. This construction is a gate in the sense that it can control the values of the *input node*. This is done by the element-wise multiplication (denoted by $\otimes$) of i_t and $\tilde{C}_t$. The control value is thereby computed with $i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i)$.

- *Internal state:* The core of the cell is denoted with C_t. In the original paper the node is depicted with a linear activation function and is called *internal state* of the cell. The self-recurrent edge with fixed weight 1.0 causes the delay of one time step and is also called *constant error carrousel* (ECE) [HOC97]. With the input node and the input gate the update formula for the internal state is given by $C_t = \tilde{C}_t \otimes i_t + C_{t-1}$.

- *Output gate:* 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 which is computed in the same way as the input gate: $o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o)$.

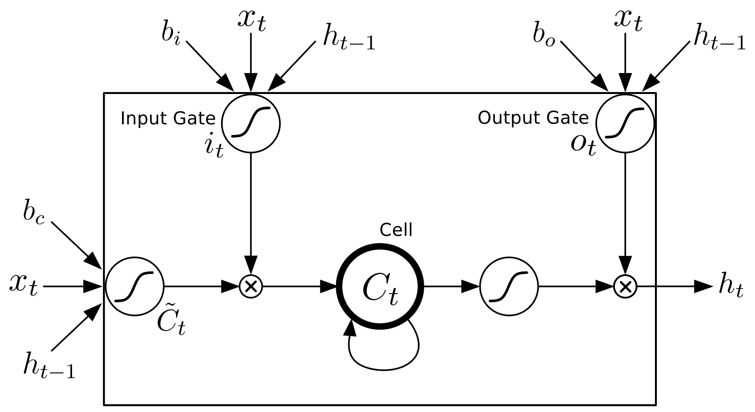


Fig. 3-8: First version of the LSTM Memory Cell by Hochreiter & Schmidhuber [HOC97], cell-state centred representation.

In 2000 Gers and Schmidhuber introduced an improved design adding a *forget gate* to the original LSTM cell [GER00]. They showed in their investigations, that the standard LSTM reached its limit when dealing with continuous long time series, causing an unbounded linear increase of the cell state. This has the effect that the output activation function reaches its saturation which leads to a vanishing gradient and h_t becomes equal to the output gate o_t. To overcome the decreasing memory functionality when using long sequences, a new gate f_t was integrated that continuously resets the memory of the cell state. Now the LSTM implementation can be summarised by the following composite function:

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \qquad \text{Eq. 3-19}$$
$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f) \qquad \text{Eq. 3-20}$$
$$\tilde{C}_t = \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \qquad \text{Eq. 3-21}$$
$$C_t = f_t \otimes C_{t-1} + i_t \otimes \tilde{C}_t \qquad \text{Eq. 3-22}$$
$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \qquad \text{Eq. 3-23}$$
$$h_t = o_t \otimes \tanh(C_t). \qquad \text{Eq. 3-24}$$

Here σ (the *logistic sigmoid function)* and $\tanh$ (the *tangens hyperbolicus*) are applied element-wise. Figure 3-9 visualises these equations of the LSTM cell, but for convenience the biases b_i, b_f, b_c and b_o are not shown. In this this figure we visualise the LSTM in an explicit fashion with an additional input and output of C_t, respectively C_{t-1}, in contrast to the cell centred approach in figure 3-8.

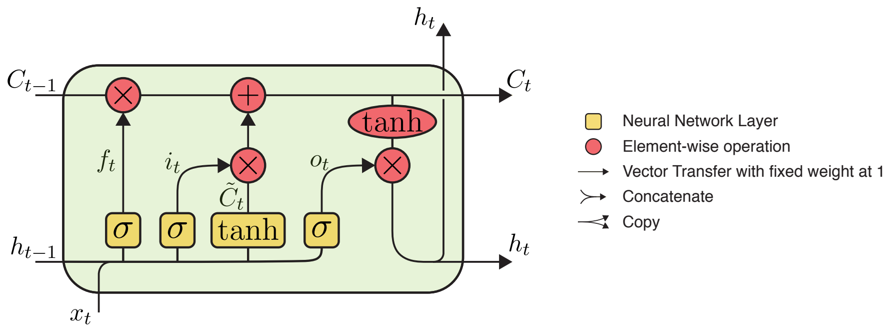


Fig. 3-9: Long Short Term Memory Cell with forget gate, explicit representation similar to Olah [OLA15].

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 5.4) and is considered in the literature as the state of the art LSTM structure [GRA13], a detailed overview is now given.

Figures 3-10 and 3-11 depict the LSTM memory cell architecture by Gers and Schmidhuber [GER03a] which extends the original LSTM with the so called *Peephole Connections*. Both diagrams represent the same cell structure. While figure 3-10 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. The equations from 3-19 to 3-24 are thus modified as following leading to the modern LSTM cell network structure,

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}C_{t-1} + b_i) \quad \text{Eq. 3-25}$$

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}C_{t-1} + b_f) \quad \text{Eq. 3-26}$$

$$\tilde{C}_t = \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \quad \text{Eq. 3-27}$$

$$C_t = f_t \otimes C_{t-1} + i_t \otimes \tilde{C}_t \quad \text{Eq. 3-28}$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}C_t + b_o) \quad \text{Eq. 3-29}$$

$$h_t = o_t \otimes \tanh(C_t). \quad \text{Eq. 3-30}$$

Overall, the LSTM has been successfully applied in many research areas. Thank to a multiplicatives gates approach, it preserves gradient information over long sequences, banning the problem of vanishing gradients which enables long ranged memory. After all benefits, LSTMs can still suffer from exploding gradients especially during an overfitting scenario [GRA13], therefore clipping the gradient while training is recommended.

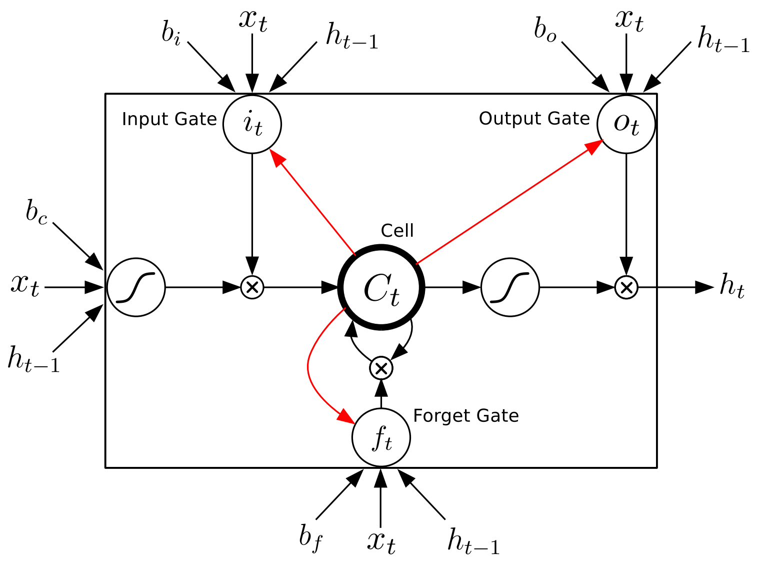


Fig. 3-10: Long Short Term Memory Cell with Peephole connections (red arrows) [GRA13], cell-state centred representation.

3.3.4 Bidirectional Recurrent Neural Networks

Bidirectional Recurrent Neural Networks (BRNNs) have been first described by Schuster et al. [SCH97]. They extend a simple RNN by adding a second hidden layer that has also a recurrent connection but to the future time step. The BRNN cell a time t receives information both from the past time step $t-1$ and the future one $t+1$. The new hidden layer or the *backward hidden layer* is denoted by z_t and produces together with the standard hidden layer h_t the output value $\hat{y}_t$ after passing the output layer (cf. Figure 3-12). The corresponding BRNN is defined by the following composite function.

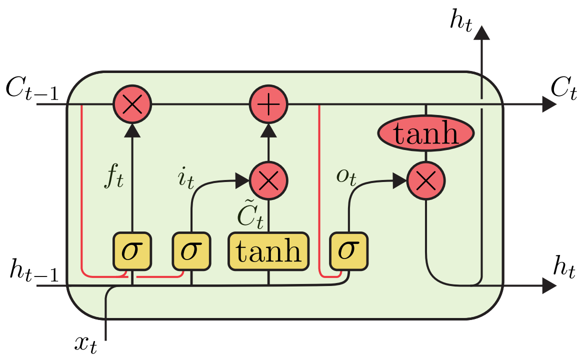


Fig. 3-11: Long Short Term Memory Cell with Peephole connections (red lines), explicit representation similar to Olah [OLA15]. Legend as in Figure 3-9.

$$h_t = \sigma(W_{hx}x_t + W_{hh}h_{t-1} + b_h) \qquad \text{Eq. 3-31}$$

$$z_t = \sigma(W_{zx}x_t + W_{zz}z_{t+1} + b_z) \qquad \text{Eq. 3-32}$$

$$\hat{y}_t = softmax(W_{yh}h_t + W_{yz}z_t + b_y) \qquad \text{Eq. 3-33}$$

Unfolding the cell along time enables BPPT. Because of the nature of this design it is not possible to use BRNN in online applications, as it is impossible to obtain observations from future events. In tasks where the current output is not only dependent on the previous data points of the sequence, but also on future elements, the BRRN seems to be the more suitable approach. Graves and Schmidhuber [GRA05] combined the ideas of BRNNs and LSTMs obtaining *BLSTMs*. They were used in [GRA09] and outperformed other models in handwriting recognition.

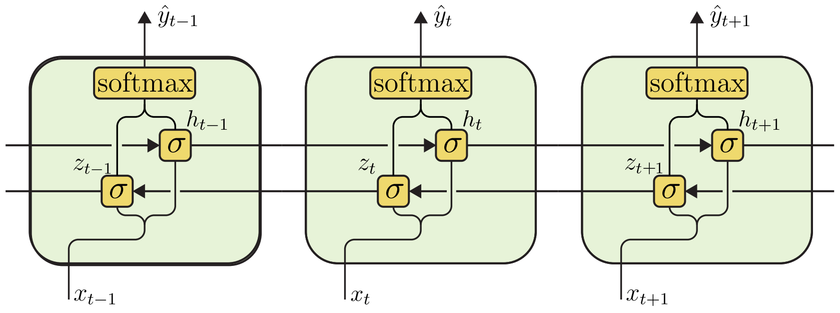


Fig. 3-12: Bidirectional recurrent neural network unfolded along time step $t-1$, t and $t+1$. Figure adapted from Lipton et al. [LIP15].

3.3.5 Stacked Recurrent Neural Networks

A recurrent neural network is naturally deep in time, if we consider more than one time step. But it is also possible to stack them vertically with *depth* N, obtaining a “deep” recurrent neural network. If we denote the previous introduced simple RNN architecture (equations 3-17 and

3-18) by $\mathcal{H}$, then we can summarise the formulation for a deep RNN by

$$h_t^1 = \mathcal{H}(W_{ih^1}x_t + W_{h^1h^1}h_{t-1}^1 + b_h^1) \qquad \text{Eq. 3-34}$$
$$h_t^n = \mathcal{H}(W_{ih^n}x_t + W_{h^{n-1}h^n}h_t^{n-1} + W_{h^nh^n}h_{t-1}^n + b_h^n) \qquad \text{Eq. 3-35}$$

for all $n = 2, \ldots, N$ and for $t = 1, \ldots, T$. Figure 3-13 visualises that each LSTM cell transfers its hidden state h_t^n as usual to the next instance in time direction h_{t+1}^n, but also to the next cell in vertical direction h_t^{n+1} and in addition to that this hidden state is passed to the output layer in order to determine the prediction $\hat{y}_t$:

$$\hat{y}_t = \sigma\left(\sum_{n=1}^{N} W_{h^ny}h_t^n + b_y\right). \qquad \text{Eq. 3-36}$$

Different formulation for the output layer are also possible like in [HER13] where only the last hidden state is considered:

$$\hat{y}_t = \sigma\left(W_{h^Ny}h_t^N + b_y\right). \qquad \text{Eq. 3-37}$$

In newer literature, for instance in Kalchbrenner et al. [KAL15], LSTMs where also stacked in *height*. The so called *Grid LSTM* consists of LSTM cells arranged in a multi-dimensional grid. As we can see, there exist uncountable possible architecture and different ways of arranging a variety of different recurrent cells opening a large field of further research.

3.4 Prediction Network Architectures

The preliminary sections introduced the fundamental *building blocks* for a 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 3-14 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 datapoint. 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.

This section focuses on the *sequence-to-sequence* approach since the topic of this thesis is to predict sequential spatial data based on the history movement of an observed object. In the following sections, four different architectures are introduced that are implemented in the *seq2seq* library [RAH16b], which is later used in chapter 5 for the trajectory prediction task. All designs have in common that they have an *encoder-decoder* structure and were 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

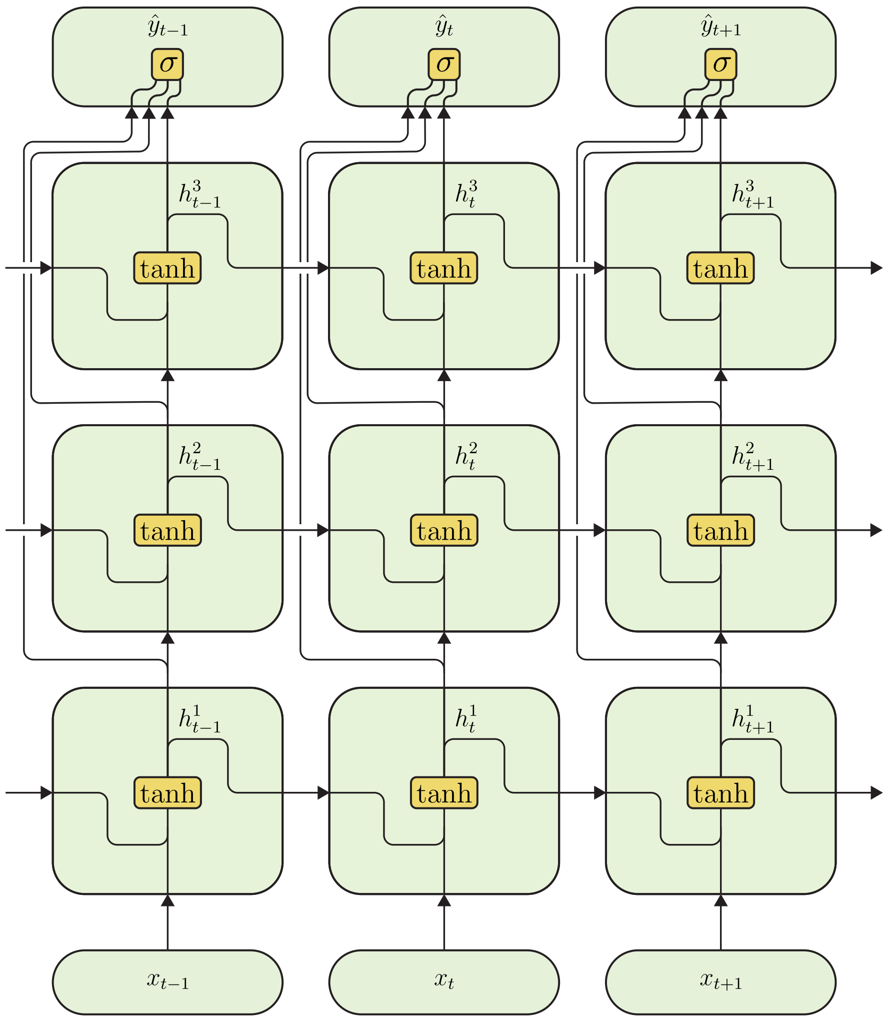


Fig. 3-13: Stacked recurrent neural network unfolded along time step $t-1$, t and $t+1$ with depth $N = 3$. All hidden states are passed to the output layer (cf. equation 3-36). Note that different non-linear activation functions than the $\mathrm{sigmoid}$ function can be used in the output layer. Figure similar to Graves [GRA13]

sequential output. The following architectures differentiate themselves from another in the way how the information flow between context vector and the decoder is constructed.
We introduce now an abbreviation for a Long-Short-Term Memory cell,

$$\mathrm{LSTM}(X) = (C_{t-1}, h_{t-1}, x_t) \quad \forall t \qquad \text{Eq. 3-38}$$

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

$$X = (x_1, x_2, \dots x_{N_{obs}}). \qquad \text{Eq. 3-39}$$

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

$$\hat{Y} = (\hat{y}_1, \hat{y}_2, \dots, \hat{y}_{N_{pred}}). \qquad \text{Eq. 3-40}$$

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

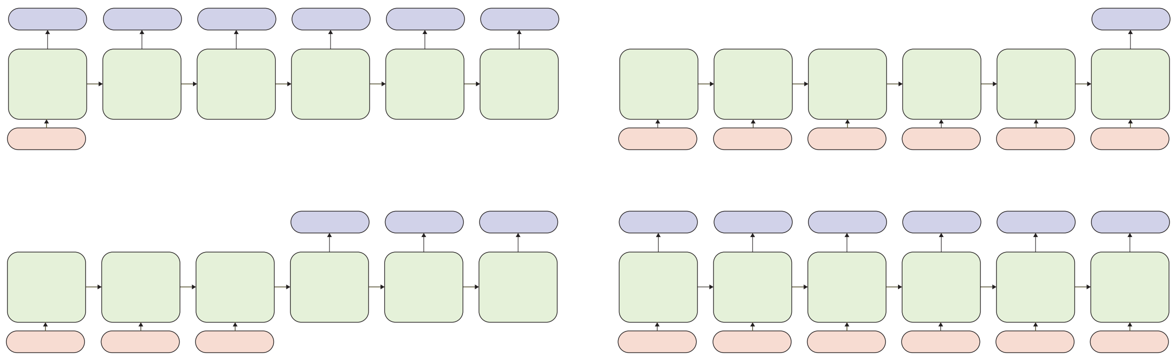


Fig. 3-14: Sequence mapping scheme based on figure from Karpathy [KAR15] with input represented as red rectangles, recurrent cells in green and output in blue. The top left subplot shows a sequence generating scenario. On the top right a sequence classification task is visualised. Sequence to sequence mapping is shown in the bottom left and “realtime” prediction on the bottom right.

3.4.1 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 English to French translation task and it achieved compelling results with the *BLEU* benchmark method [PAP02]. Furthermore, the researchers found that the net performs especially well on long sentences and reversing the order of the input sequence has a positiv effect on nets accuracy. In order to deal with all possible different lengths of the input and output sequences, they used a `<EOS>` (end-of-sentence) token.

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

$$\begin{aligned} X &= (x_1, ..., x_{N_{obs}}) \\ V &= \mathrm{LSTM}(X) \end{aligned} \qquad \text{Eq. 3-41}$$

and for the decoder by,

$$\begin{aligned} \hat{y}_1 &= \mathrm{LSTM}(C_0, h_0, V) \\ \hat{y}_t &= \mathrm{LSTM}(C_{t-1}, h_{t-1}, \hat{y}_{t-1}) \ \ \forall\, t = 2, \ldots, N_{pred}. \end{aligned} \qquad \text{Eq. 3-42}$$

In addition to that figure 3-15 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.

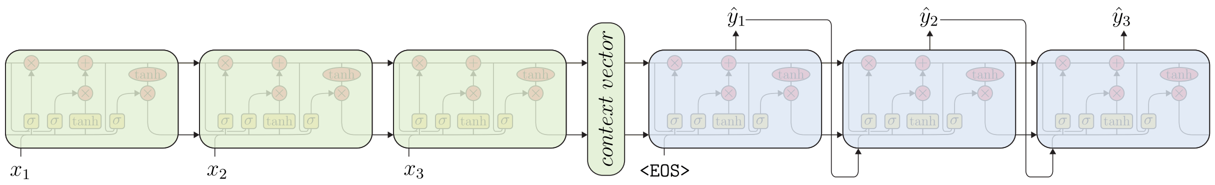


Fig. 3-15: Sequence to sequence architecture by [SUT14] with $N_{obs} = N_{pred} = 3$. In green the encoding LSTM and in blue the decoder. The prediction of the decoder $\hat{y}_t$ at each time step t becomes the input for the decoder at the next time step. Figure adapted from Lipton et al. [LIP15].

3.4.2 Simple Sequence to Sequence Model

This framework is a simplification of the previous one and is implemented in [RAH16b]. This time $\hat{y}_{t-1}$ is not passed to the decoder and hence the formulation of the decoder is given by,

$$\begin{aligned}\hat{y}_1 &= \mathrm{LSTM}(C_0, h_0, V)\\ \hat{y}_t &= \mathrm{LSTM}(C_{t-1}, h_{t-1}, V)\ \ \forall\, t = 1, \dots, N_{pred}.\end{aligned} \quad \text{Eq. 3-43}$$

3.4.3 Sequence to Sequence Model with Peek

The model by [CHO14] extends the architecture in section 3.4.1. Now the encoder is modelled in such a way that at each output time step the LSTM has a *peek* at the summary V. For the decoder a slight modified LSTM architecture is used,

$$\begin{aligned}\hat{y}_1 &= \mathrm{LSTM}(C_0, h_0, V, V)\\ \hat{y}_t &= \mathrm{LSTM}(C_{t-1}, h_{t-1}, y_{t-1}, V)\ \ \forall\, t = 2, \dots, N_{pred}.\end{aligned} \quad \text{Eq. 3-44}$$

3.4.4 Attention Sequence to Sequence Model

Bahndanau et al. [BAH14] developed this framework for further performance improvement of encoder-decoder architectures. This model deploys a BLSTM as encoder and an additional FNN between encoder and decoder. It is reported that it generates better alignment between input and output data for language translation tasks. The formulation of the decoder can be summarised by

$$\begin{aligned}X &= (x_1, ..., x_{N_{obs}})\\ S_i &= \mathrm{BLSTM}(X)\end{aligned} \quad \text{Eq. 3-45}$$

where S_i is a sequence of hidden states of all inputs, hence it has a lengths of N_{obs}. The context vector is then a weighted sum,

$$v_i = \sum_{j=1}^{N_{obs}} a_{ij} S_j \quad \text{Eq. 3-46}$$

where the weight matrix a_{ij} for each h_j is computed with the energy matrix e_{ij},

$$\begin{aligned} e_{ij} &= a(C_{i-1}, h_{i-1}, S_j) \\ a_{ij} &= \mathrm{softmax}(e_{ij}). \end{aligned} \qquad \text{Eq. 3-47}$$

Hereby, represents a a FNN which weights are trained simultaneously with the other weights of the architecture. Then with the help of the context vector the hidden states of the encoder are computed they are put through an output layer with a non-linear activations function to determine the prediction $\hat{y}_t$. This implementation is realised by a single *Attention Decoder cell* (further details in [RAH16b]). Note that there is no direct transfer of the hidden state from the encoder to the decoder.

4 Research Objective and Approach

In this chapter, the general research objective and approach of this thesis is presented. First of all, the introduced existing prediction models (cf. section 2) are analysed. On this basis, the demand for further research is figured out, so that two research questions can be formulated in section 4.2. In the next section 4.3, the approach to answer these research questions is explained. 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 4.4.

4.1 Analysis of Existing Prediction Models

A variety of different classes of pedestrian prediction models are discussed in section 2. The Social Forces (SF) models are one of oldest approaches to model pedestrian dynamics. Most of these SF models are based on handcrafted functions that simulate "energy" potentials between individuals, environmental constraints and the individuals intended ground truth destination [HEL90] [YAM11]. Evidently, the intended destination for one pedestrian in a intersection scenario is not known *a-priori*. Hence, these models are suitable to simulate the movement of a pedestrian, but the deployment of a SF model for the prediction of single pedestrians seems not to be appropriate.

In almost all presented statistical prediction models, the observed scene was discretised in finite elements. During the training of such models constraints, movement directions, positions and velocities of different classes of road users were associated with these elements (cf. [BAL16] [BRO16]). In order to predict the most probable future trajectory of one individual, the probabilities over the grid are evaluated. Therefore, the prediction quality would suffer if the chosen element size is too large. Although the model path prediction is not on the continuous domain, some approaches showed promising results [BAL16]. Nevertheless, regarding a safety-related system it is questionable, whether statistical models are able to correctly predict individual outliers and abnormal behaviour.

Pedestrian trajectory prediction with RNN was already investigated by Alahi et al. [ALA16]. In order to model social behaviour between individuals, they connected spatial approximate LSTMs. Despite their efforts, the described approach suffers from some drawbacks. First, the research did not investigate the potential of recurrent sequence to sequence models for this task. Further, their model quality evaluation was performed on small and simple datasets. For example the *ETH* and *Hotel* datasets have only two prevailing movement directions (cf. section 11.4.1). In the first case, the majority of the tracked pedestrians heads simply towards and away from a building's entry. And in the second case, the pedestrians walk straight bi-directional along the pavement. In contrast to that, a usual intersection scenario is often much more complex. Pedestrians wait at signal lights (or not), they do spontaneous street crossing manoeuvres and they move with different velocities in much more directions. These different movement patterns where not covered in the proposed datasets. In addition to that, the used datasets in [ALA16] are not very dense. The biggest collection of trajectories is the ETH dataset with approx. 9000 data points and 359 single trajectories whereas the other datasets are remarkably smaller[1]. So far, it is unclear if these small number of trajectories for a single scene are sufficient to give valid statistical statements of the model quality and further there is a need to investigate the impact of large sized datasets on recurrent neural net's performance.

Other analysed approaches make serious simplifications on the movement of the pedestrian. In [KEL14] and in [HAS15] the model's response is either "the pedestrian will cross the road" or "the pedestrian stops walking at the kerb". The precise position estimation for future time steps is not included and moreover, these models were developed for vehicle's camera systems and are therefore not directly deployable for infrastructure-base sensors.

A literature search revealed that until today no continuous self-improving pedestrian prediction model was used or proposed. More general approaches for continuously learning ANNs were found for CNNs (e.g. [XIA14]), but not for RNNs.

4.2 Research Questions

The analysis of the relevant existing literature for prediction models shows that it is desirable to investigate the potentials of recurrent neural networks for pedestrian movement prediction in urban spaces. 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. This leads to the question, how these potentially powerful neural nets can be applied to our setting? In the literature, there are no indications which special architecture design could perform well on this problem. Hence, the presented encoder-decoder designs in section 3.4 have to be investigated.

In general, neural nets have a lot of fine tuning parameters such as the choice of the optimiser, number of hidden dimensions and training epochs. The net's prediction quality steadily depends on the correct estimation of these hyper parameters, but exploring the parameter space for an optimal fit is a very time consuming task especially with large-scaled datasets. As a result, there is a need to systematically discover the parameter space for sequence to sequence learning with respect to relevant datasets.

Further, every deep learning approach makes use of intense data preprocessing in such a way that the neural net is able to successfully process and store the given training data. In the literature, many different methods for various kinds of problems and for different net architectures are proposed [VIN14] [GRA05]. To the best of the authors knowledge, a preprocessing method for trajectory datasets has not been investigated so far. In the context of the given setting, it is a crucial task to work out a preprocessing method for a spatial sequential dataset that consists of many single trajectories captured at one scene.

[1] e.g. ZARA 1: 1500 data points and 148 trajectories

As the performance of a neural net is an interplay between data preprocessing, data quality, network architecture and hyperparameters, it is not sufficient to optimise only one of these domains. Thus, it is necessary to develop a holistic approach within the framework of this thesis. Furthermore, another goal of this work is to analyse the impact of large sized training datasets captured at intersection scenarios.

Another aim of this thesis was to develop a *continuously* learning forecasting model. The steady data stream from an urban intersection observed with sensor infrastructure can be used to continuously improve a recurrent neural net over time. Since related literature has not yet treated this issue, there exists the need to develop and examine basic approaches.

Based on the obtained results of the model configuration exploration, it is necessary to compare and evaluate the prediction quality. Therefore, another goal in this thesis is to analyse, whether the forecasting reliability and stability is comparable to baseline models and whether the model's predictions are acceptable for a safety-critical application at urban intersections.

The goals and topics of this bachelor thesis can be summarised in two research questions:

1. **What is the optimal recurrent neural net design for an optimal pedestrian trajectory prediction quality ?**

2. **How to design the neural net's training process with a continuous data stream captured at intersection scenarios ?**

4.3 Approach

The approach to answer the respective research questions is the following: At first, the implementation of the presented prediction network architectures in section 3.4 with the help of third-party deep learning libraries such as Keras [CHO15] and the Keras add-on Seq2Seq [RAH16b] should be performed. Based on this implementation, basic validation tests with self-generated and public available datasets should be carried out to prove fundamental viability of the neural network. Then, an iterative process should be carried out, in order to optimise the interaction of different methods of data preprocessing and network configuration. This process is controlled by visualising and evaluating different performance criteria (see also section 4.4) of the model's output.

With this results, a well-performing preprocessing and model configuration should be validated and compared with results by Alahi et al. [ALA16]. To realise this, the network should be trained with the datasets ETH and Hotel (cf. section 11.4.1) and identical implemented performance criteria should lead to a comparability of both approaches.

In the next step, more complex datasets from urban intersection scenarios have to be investigated. But so far, there exist only a few datasets captured at intersection that are publicly available. Because of that, measurements with a laser scanner should be carried out to generate more suitable datasets. In addition to that, datasets generated by the traffic and pedestrian simulation software Vissim [PTV17] should be used. Further, hyper-parameter fine tuning has to be applied on these new datasets to determine the best performing model configuration for each of them. Moreover, the performance of the RNN on these datasets has to be compared with basic baseline models such as a Kalman filter and a feedforward neural net.

For the continuously learning mechanism, a single intersection, but with two different pedestrian movements scenarios is considered. Consequently, the first dataset represents the intersection under normal conditions while the second dataset contains an obstacle, for example a construction area or a blocked road, in the pathways of the pedestrians. With this setting, a temporal change of an intersection is simulated. Starting from the premise that the neural net is fitted on the first dataset, motion characteristics from the second dataset should be additionally trained into the existing model. Thereby, it is crucial to perform this continuous learning with minimal additional training effort. The approach is to *retrain* the neural net with a new *merged* dataset consisting of trajectories of the first and the second dataset. This should ensure, that already invested computational effort into the model is preserved, whereas new unknown data samples taken from the new scenario are added to the model. One possibility to construct the new training dataset would be to merge the first dataset with badly predicted trajectories of the second one. A further approach would be to retrain only the last captured pedestrian trajectories. Hence, it is necessary to analyse the effects of different dataset merging methods.

4.4 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. In this thesis we will distinguish between *primary* and *secondary* performance evaluation criteria. Primary criteria are defined as quantities that directly measure the accuracy of a predicted trajectory in relation to the true future path of an individual. To persevere consistency with relevant literature, some of the metrics used by Alahi et al. [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 coordinates 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 11.3.

Road safety systems must have a reaction latency of several milliseconds in order to prevent possible collisions. A pedestrian prediction model would be useless if it would take several seconds to compute an estimate future position. Additionally, to keep long-term efficiency of a safety system, the training duration of the deployed ANN should not be too expensive, especially when a continuously learning approach is deployed. Thus, by secondary performance evaluation criteria we refer to properties of a model that describe the prediction speed and the training duration. To proof the practicability of such a prediction model, these quantities are measured and interpreted, but an in depth analysis is not focus of this thesis.

5 Model Design and Implementation

This chapter gives a detailed description of the implementation of the approach of this thesis. In the first section 5.1, a general overview about utilised deep learning libraries and the developed code structure will be presented. Section 5.2 specifies the datasets on which the deep learning approach will be applied. Consequently, section 5.3 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 5.4 and in 5.5 respectively. Details about the training procedure and the evaluation of the model quality can be found in section 5.6. Further, section 5.7 deals with the optimisation of the model's hyper-parameters. Several basic sampling methods used to investigate the impact of model parameters are elucidated in section 5.8. In order to measure the robustness of a trajectory prediction model, a stability analysis method is proposed in section 5.9. Finally, section 5.10 presents several approaches that are intended to analyse a continuous learning mechanism for a steady data stream.

5.1 General Code Setup

In the early stages of this work technical requirements had to be assessed. For deep learning there exists several popular libraries such as Theano [THE16], Tensorflow [ABA15] and Caffe. All of them are able to compile efficient code which can be executed and accelerated by GPU devices. On basis of the accessible hardware at the *RWTH Compute Cluster* and available libraries, the choice fell on Theano as back-end library that is accessed by the high-level library *Keras* [CHO15]. On top of that, the add-on *seq2seq* [RAH16b] is used for the sequence to sequence models presented in section 3.4. This setup allows in combination with the programming language Python 2.7 a very flexible and fast code development. An implementation of the presented approach in 4.3 with the very popular Tensorflow library was not suitable, as it needs special hard- and software requirements which were not fulfilled.

The flexibility of the implementation is introduced by a modular programming design. That means, different kinds of subroutines for pre-processing, model generation, training process, results evaluation and visualisation are organised in modules. These modules do not have data encapsulation nor an object orientated design, as the developed framework has a conceptual focus. Since the training of a neural net can take from a few hours up to a several days depending on dataset size and training configuration, it is necessary to use the GPU batch mode system of the RWTH Compute Cluster. Therefore, the code and data structure are designed for a job execution system.

When submitting a job, or a so called "run", to the Compute Cluster or to the IKA Machine, all important functionalities of the developed framework can be controlled by a single configuration file. This configuration file (cf. attachment 11.1) consists of several control options, that define the model, the used dataset, preprocessing settings and further options. A configuration file is committed to the framework via command-line parameter. Figure 5-1 depicts roughly the code structure of the project folder. Directory `TrajectoryPrediction` contains all python source code

```
/
├── TrajectoryPrediction
│   └── ⋮
├── configs
│   ├── config1.ini
│   ├── config2.ini
│   └── ⋮
├── data
│   ├── eth
│   ├── laserscanner
│   ├── vissim
│   └── ⋮
├── output
│   ├── output_config1
│   └── output_config2
└── jobs
    ├── job_config1.sh
    └── job_config2.sh
```

Fig. 5-1: Code and data structure

files that are needed to perform the deep learning task (further details in the next sections), whereas the folders `configs`, `data`, `output` and `jobs` contain the configurations files, the raw datasets, the results and the batch job configurations. Every run has its own output folder in which the weights of the trained neural net, the results, the visualisation files and logged information are saved. This has the advantage, that is always possible to reproduce the results by reusing the weights of the prediction model.

5.2 Datasets

In this thesis, nine datasets from four different sources are used which were captured with diverse measurement methods. A visualisation and statistics for each of the datasets can be found in Attachment 11.4. Due to a lack of real intersection measurements with pedestrians, trajectories from vehicle positions are also applied, although car movements are not focus of this thesis. Nevertheless, performing the trajectory prediction task with vehicle data should basically be the same as dealing with pedestrian trajectories. Hence, the following dataset sources are studied in order to show proof of concept.

- **ETH**: Camera-based pedestrian tracking at crowded spaces in Zürich. The trajectories were manually annotated and the whole dataset was published by [PEL09].

- **Laser**: Intersection scenarios in Aachen captured by a stationary 4-phase laser scanner. The majority of the tracked objects are vehicles.

- **Vissim**: Pedestrian trajectories that are generated by the software Vissim [PTV17] that is equipped with a social forces model.

- **ATC Dataset**: Trajectories from pedestrians that were tracked with a high-precision GPS receiver at the Aldenhoven Testing Center.

5.3 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 pedestrian in a planar cartesian coordinate system at time t. Thus, the trajectory Y_i can be written as a two-dimensional matrix where the first column denotes the x-coordinates and the second column consists of the y-coordinates measured in $[m]$:

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

where $i = 1, \dots, K$ with K the total amount of trajectories in the dataset and N_i the total number of sample points of trajectory Y_i. This notation is important and also appropriate, since a single trajectory is also implemented in the source code as a NumPy ndarray [JON15] with the above denoted shape.

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. **Downsampling**
   The selected datasets are captured with different sample rates, e.g. the laser datasets with $25\ [Hz]$ or the Vissim datasets with $10\ [Hz]$. In order to maintain comparability between them and to reduce the size, all trajectories in the relevant datasets are downsampled to $2.5\ [Hz]$, as described in [ALA16]. Due to the reason that all datasets have a sample rate that can be divided by $2.5\ [Hz]$ without remainder, the downsampling is performed by a decimation factor without interpolation.

2. **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 each column of Y_i in order to smooth the trajectories. Thereby, the parameter Σ, which denotes standard deviation of the Gaussian kernel, controls the intensity of smoothness applied to each trajectory. This smoothing factor can be modified in the configuration file, but in the following we take $\Sigma = 1$ as a standard parameter.

Listing 5.1: Configuration file snippet - Smoothing factor

```
[preprocessing]
smoothing_sigma = 1
```

3. **Normalisation**
   Data normalisation is an essential step of the pre-processing for machine learning. Hereby, all data points of the dataset are linearly scaled from the original domain to the new region $[-1, 1]^2$. This ensures 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 pedestrian's movement. The observation sequence is also known as the *history* of the pedestrian. Like in [ALA16], the observation length is $T_{obs} = 3.2\ [s]$ which corresponds to $N_{obs} = 8$ data points sampled with $2.5\ [Hz]$. 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. Hereby, the prediction window has a length of $N_{pred} = 12$ sample points that corresponds to an observation time of $T_{pred} = 4.8\ s$. This pre-processing step noticeably introduces redundancy in the training data, but according to literature "redundancy acts like regulariser and reduces overfitting" [JAI15] during training.

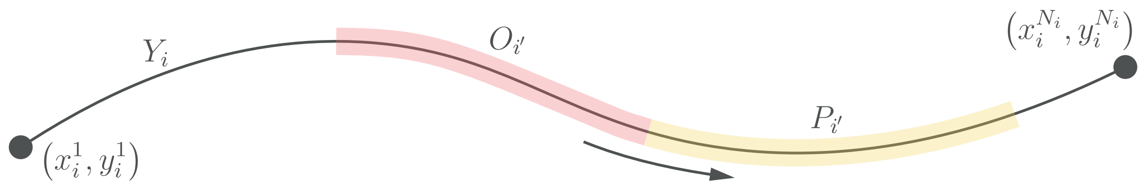


Fig. 5-2: Sliding Window applied to trajectory Y_i. In red the window, the observations are taken into account while the prediction trajectories are captured by the yellow window. That applies for $1 \leq i' \leq N_i - (N_{obs} + N_{pred}) + 1$.

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 $P_{i'}$. The number of gained observation-prediction pairs is $S = N_i - (N_{obs} + N_{pred}) + 1$ where each pair has the index $i' = 1, \ldots, S$.

$$O_{i'} = \begin{pmatrix} x_{i'}^1 & y_{i'}^1 \\ x_{i'}^2 & y_{i'}^2 \\ \vdots & \vdots \\ x_{i'}^{N_{obs}-1} & y_{i'}^{N_{obs}-1} \\ x_{i'}^{N_{obs}} & y_{i'}^{N_{obs}} \end{pmatrix} \in \mathbb{R}^{N_{obs} \times 2} \qquad P_{i'} = \begin{pmatrix} x_{i'}^1 & y_{i'}^1 \\ x_{i'}^2 & y_{i'}^2 \\ \vdots & \vdots \\ x_{i'}^{N_{pred}-1} & y_{i'}^{N_{pred}-1} \\ x_{i'}^{N_{pred}} & y_{i'}^{N_{pred}} \end{pmatrix} \in \mathbb{R}^{N_{pred} \times 2}$$

Eq. 5-2

Further, a collection of observation or prediction trajectories can be written as a three-dimensional tensor. This notation corresponds also to the actual implementation of the training dataset.

$$O \in \mathbb{R}^{S \times N_{obs} \times 2} \qquad P \in \mathbb{R}^{S \times N_{pred} \times 2} \qquad \text{Eq. 5-3}$$

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.

Listing 5.2: Configuration file snippet - Sliding Window configuration

```
[preprocessing]
observationlength = 8
predictionlength = 12
```

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 ETH dataset is taken as an example. The original trajectory collection consists of 359 trajectories with different length that have together 8888 data points. The preprocessing method returns the observation tensor $O \in \mathbb{R}^{2613 \times 8 \times 2}$ and the corresponding prediction collection $P \in \mathbb{R}^{2613 \times 12 \times 2}$, which consists of slightly smoothed trajectories that are rescaled to $[-1, 1]^2$. Both datasets have now together 52260 data points.

5.4 Model Details

The structure of the described encoder-decoder models in section 3.4 consists of two parts. In the first component, the input sequence is mapped to a fixed-sized internal vector using the encoding LSTM. In turn this representation is used to generate the target sequence with another LSTM. 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. During training, these two components of the sequence to sequence models are jointly trained in order to minimise a loss function $\mathcal{L}(\hat{P}, P)$ for all the trajectories. Hence, the arising general optimisation problem can be written as

$$p_w(P|O) = \underset{p_w(P,O)}{argmin} \{\mathcal{L}(\hat{P}, P)\} \qquad \text{Eq. 5-4}$$

where $\hat{P} = f(O; w)$ is the model's prediction and w are the weights of the RNN. Hereby, the training dataset that consists of S sample pairs is used

$$\{(O_i, P_i) : i = 1, \ldots, S\}. \qquad \text{Eq. 5-5}$$

On this basis, the **default prediction model** of this thesis is defined by the following functional components:

- **Loss function:** Mean squared displacement $\mathcal{L}_{MSD}$ (cf. section 11.3)

- **Architecture:** Sequence to sequence model by [SUT14] described in section 3.4.1
- **Optimiser:** Adam optimiser [KIN14] with a learning rate of $\eta = 0.001$

The reason for these specific choices are early experiences and comparisons, but a detailed investigation can be found in section 6. The actual implementation allows to exchange every of this components via the configuration file.

Listing 5.3: Configuration file snippet - Model default configuration

```
[model]
architecture = seq2seq
loss = mean_squared_displacement
optimizer = adam
```

5.5 LSTM Configuration Details

As we know from section 3.3.3, the designation “LSTM” does not define a specific RNN architecture, but there exist different variants of LSTM cells with different configurations. In the following the deployed LSTM cell will be described.

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] (cf. equations 3-19 to 3-24), but with a $\mathrm{hard\ sigmoid}$ activation function instead of the usual $\mathrm{sigmoid}$ activation function (cf. attachment 11.2). The $\mathrm{hard\ sigmoid}$ activation function is a piecewise linear approximation of its counterpart, with the effect that it does not use the computational expensive exponential function. This leads to a shorter training duration. Identical to the original definition of the peephole LSTM, the activation function for the transferred hidden state is the $\tanh$ function (cf. equation 3-27). At the initialisation of the LSTM cell, the weights of the gates are assigned randomly with the *glorot uniform* distribution whereas the weight matrix of the input node is determined by an orthogonal initialisation method so that the eigenvalues are equal to one [RAH16a].

5.6 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 pre-processed dataset is split with a factor of $\alpha = 0.7$ into two parts. The first and bigger dataset is referred to as the *train split*, whereas the smaller dataset is called *test split*.

$$\text{Train split:} \quad \{(O_i, P_i) : 1 \leq i \leq \lfloor S * \alpha \rfloor\} \qquad \text{Eq. 5-6}$$

$$\text{Test split:} \quad \{(O_i, P_i) : \lceil S * \alpha \rceil \leq i \leq S\} \qquad \text{Eq. 5-7}$$

As already explained, the execution of one training epoch implies that all trajectories of the train split are put once through the neural network to the determine the error signal and hereby modify the model weights. But before an epoch is executed, all samples in the test split are randomly shuffled, in order to archive a possible faster convergence of the training [BEN12].

Moreover, in order to prevent the model from overfitting, an *early stopping* mechanism is used during training. Therefore, an additional split is introduced that consists of 10% of the train split. This *validation split* is continuously evaluated after each epoch and if the prediction quality of this part stops increasing for a predefined number of epochs, the training will be stopped [CHO15].

Likewise in [GRA13], an additional regularisation method is applied during the training procedure that rescales the gradient if it exceeds a certain value. A default gradient clip value of 1 is used, so that all weight gradients are clipped in the range $[-1, 1]$ [CHO15].

When the training of the RNN has finished due to early stopping or when the maximal number of epochs has been reached, the observation trajectories of the test split are taken as an input to the final model $f(\cdot, w)$. Consequently, the output of the RNN are the corresponding trajectory predictions $\hat{P}_i$:

$$\hat{P}_i = f(O_i, w) \quad \forall i = \lceil S * \alpha \rceil, \ldots, S \qquad \text{Eq. 5-8}$$

Subsequently, the predictions are measured against the true trajectories P by computing the introduced performance evaluation criteria $\mathcal{L}(\hat{P}, P)$. Additionally, several visualisation methods are called that generate a variety of graphical representations of the results.

5.7 Hyperparameter & Hyperparameter Tuning

Typically, ANNs depend on many configuration settings that have to be determined in order to use the net's full potential. Thus, discovering this parameter space is an important step before making further investigations. When the default model (cf. section 5.4) is considered, the following parameters are now denoted as the *hyperparameters* of the prediction model.

- **Number of Hidden Dimensions** $h \in \{32, 64, 128, 256, 512, 1024\}$
- **Learning Rate** $\eta \in \{0.0005, 0.001, 0.002, 0.003\}$
- **Number of Training Epochs** $e \in (1, 1000)$
- **Training Batch-Size** $b \in \{16, 32, 64\}$
- **Depth of Encoder and Decoder** $d \in \{1, 2, 3\}$
- **Dropout Probability** $p \in [0, 1)$

In addition to that, reasonable limitations for each parameter are defined which leads to a six-dimensional parameter space. These special choices are based on basic considerations and experiences taken from relevant literature. To automate the exploration of a hyperparameter space several strategies can be applied.

In early experiments, the cross-validation method GridSearchCV [PED11] was deployed. This algorithm, partitions the datasets into several test and train splits. Then, the model is repeatedly trained and tested while test and train splits are frequently rotated. Thereby, the hyperparameters are evaluated on the full grid of the parameter space. However, this method caused an impracticable overall runtime, so that a search algorithm was used instead.

Hence, the *Hyperopt* library [BER13b] was applied on this setting. The package is equipped with the TPE search algorithm that showed good performance for different types of ANNs [BER13a]. The optimisation is performed by minimising the loss $\mathcal{L}_{MSD}$ of the test split while the net is trained with the fixed and randomised train split.

5.8 Parameter Analysis

A hyperparameter optimisation is a great tool for an initial exploration of the parameter space, but this kind of search does not reveal any in-depth parameter dependencies or sensitivities. To gain a better understanding of certain model parameters or to visualise special cross-correlations among them, it is necessary to sample some particular parameter subspaces. Therefore, three basic types of parameter analyses are presented.

- **1D parameter analysis**
  Simple sampling of a single parameter.

  Listing 5.4: Example configuration - 1D parameter analysis

```
[parameter_analysis_1D]
hidden_dim = 32, 64, 128, 256, 512
```

- **2D parameter analysis**
  Sampling on a two dimensional mesh, so that all combinations are covered.

  Listing 5.5: Example configuration - 2D parameter analysis

```
[parameter_analysis_2D]
hidden_dim = 64, 128, 256
observationLength = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
```

- **2D epoch analysis**
  Basically a 1D parameter analysis, but the model is evaluated after every prescribed epoch count until the last epoch is reached.

  Listing 5.6: Example configuration - 2D Epoch Analysis

```
[epoch_analysis_2D]
nb_epoch = 50, 100, 150, 200, 250, 300, 350, 400, 450
depth = 1, 2, 3
```

During the sampling, all other options stay fixed and unless otherwise stated the early stopping method is deactivated. On each sample point of these analyses, the performance evaluation criteria are applied and the results are eventually plotted.

5.9 Stability Analysis

In section 4.3 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

$$\hat{P}^{\sim} = f\left(O^{\sim}, w\right). \qquad \text{Eq. 5-9}$$

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 6.6). The whole process is not only performed with a single observation, but with a selected subset of all test observations.

5.10 Continuous Learning

A steady data stream captured at an intersection scenario enables the possibility for a continuous self-improving predictive model. Nevertheless, a review of the relevant literature indicated that this approach has not yet been investigated. Hence, to analyse basic continuous learning approaches, the following simplified situation is considered.

We start from the premise that an urban intersection scenario undergoes an abrupt change of its traffic flow. This could be caused by a start of construction works or a blocked road that forces the pedestrian to change their movement patterns. This event is denoted in the following by $\mathcal{A}$. Prior to $\mathcal{A}$, potential infrastructure sensors have gathered trajectories that are pre-processed and merged into dataset I

$$\{(O_i^{\mathrm{I}}, P_i^{\mathrm{I}}) : 1 \leq i \leq S^{\mathrm{I}}\}. \qquad \text{Eq. 5-10}$$

This dataset is subsequently used to train the proposed architecture obtaining the prediction model $f^{\mathrm{I}}(\cdot, w)$. If we work on the premise that the model $f^{\mathrm{I}}(\cdot, w)$ is deployed for a prediction task, the occurrence of event $\mathcal{A}$ could cause bad predictions at specific areas, ergo the areas that are afflicted by $\mathcal{A}$.

These erroneous predictions can be determined by computing the prediction error E_i for each

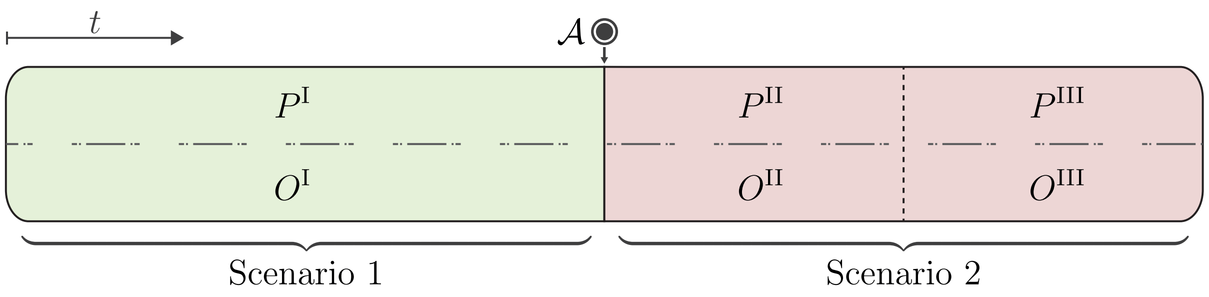


Fig. 5-3: Dataset splitting for continuous-learning experiments. In green, the dataset captured previously to event $\mathcal{A}$. The second dataset that contains a different movement pattern induced by $\mathcal{A}$ is coloured red. Hereby, the second dataset is divided into split II and split III.

observation trajectory O_i^{II}. That means, observations captured after event $\mathcal{A}$ (cf. Figure 5-3) are used to predict their counterparts and the resulting model predictions $\hat{P}^{\mathrm{II}}$ are measured against the ground truth P^{II}. Using our notation we can summarise,

$$E_i^{\mathrm{II}} = \mathcal{L}_{MSD}(\hat{P}_i^{\mathrm{II}}, P_i^{\mathrm{II}}) \quad \forall\, i = 1, \ldots, S^{\mathrm{II}} \qquad \text{Eq. 5-11}$$

with the predicted trajectories $\hat{P}^{\mathrm{II}} = f\left(O^{\mathrm{II}}; w\right)$ and the true trajectories P^{II}. Note, that all trajectories are rescaled to the original domain during the computation of the error values

On this basis, it is the aim to analyse different approaches that retrain the existing model $f^{\mathrm{I}}(\cdot, w)$ with a new dataset consisting of trajectories of split I and of split II. Retraining the model has the effect that computational effort that was already invested into the model is partially preserved. Furthermore, a specific merging method prevents the new dataset from getting too large. In the following, four different methods are proposed.

- **Worst trajectories**
  The worst performing observation-prediction pairs of split II based on E_i^{II} are merged together with split I. Then this new dataset is used to retrain $f^{\mathrm{I}}(\cdot, w)$. Thus, already good performing trajectories of split II are not unnecessarily added to the new training set. Hereby model $f_W^{\mathrm{II}}(\cdot, w)$ is obtained.

- **Random trajectories**
  Randomly chosen observation-prediction pairs of split II are merged together with split I. Then this new dataset is used to retrain $f^{\mathrm{I}}(\cdot, w)$. Hereby model $f_R^{\mathrm{II}}(\cdot, w)$ is obtained.

- **Sliding Window**
  Retraining of $f^{\mathrm{I}}(\cdot, w)$ is done with the last S^{I} captured observation-prediction pairs. Hereby the model $f_{SW}^{\mathrm{II}}(\cdot, w)$ is obtained.

- **Best trajectories**
  The best performing observation-prediction pairs of split II based on E_i^{II} are merged

together with split I. Then this new dataset is used to retrain $f^{\mathrm{I}}(\cdot, w)$. Hereby the model $f_{B}^{\mathrm{II}}(\cdot, w)$ is obtained. This approach is only taken as reference.

The retraining process is performed with a remarkable smaller number of epochs compared to the initial training duration of $f^{\mathrm{I}}(\cdot, w)$. Finally, the performance of the new models $f_{W|R|SW|B}^{\mathrm{II}}(\cdot, w)$ is evaluated on split III, thus a dataset that the model has not been trained on.

In order to perform the presented experiments, three synthetic datasets were generated with Vissim (cf. Attachment 11.4). Hereby, Figure 11-3i is used as scenario 1 whereas Figures 11-3ii and 11-3iii are used for the second scenario captured after $\mathcal{A}$. Note that the trajectories in Figure 11-3ii show the pedestrians movement behaviour influenced by a construction work scenario and a blocked road scenario is depicted in 11-3iii.

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 four different sequence to sequence architectures is compared in section 6.2. A variety of optimisers were tested and evaluated in section 6.3. On this basis, an optimal configured sequence to sequence architecture is measured against two baseline models in section 6.4. An in-depth parameter sensitivity analysis of the RNN is performed in section 6.5. The prediction stability of the presented approach is visualised and assessed in section 6.6. Further, section 6.7 deals with prediction errors that are caused by missing training data. Two experiments were executed in order to evaluate the continuous learninig approach. The results are discussed in section 6.8. Finally, section 6.9 briefly discusses the real-time capabilities of the sequence to sequence approach.

All computations were executed on GPU devices. In most cases, the GPU Batch System of the *RWTH Compute Cluster* was used. It provides several *NVIDIA Quadro 6000* GPUs with limited runtime. Long time computations had to be executed on a desktop computer equipped with a *NVIDIA GeForce GTX 1070*.

6.1 Initial Hyperparameter Parameter Search

During the hyperparameter search, it turned out that finding optimal *parameter* set for each dataset is an almost impossible task. The high-dimensionality of parameter space leads to the so-called “curse of dimensionality”. This term implies, that there exists a huge number of possible parameter combinations that increases exponentially with the amount of dimensions. In addition, every sample needs a relative high evaluation duration. This results in an enormous overall runtime for a single search. Nevertheless, an initial hyperparameter search was performed with the tool Hyperopt [BER13b] in order to obtain a rough 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 the test split. Hereby, 60 optimisation iterations were executed for all smaller datasets (ETH, Hotel, ATC). 30-40 Iterations were applied for the large datasets (Laser, Vissim). The default hyperparameter space was deployed, as described in section 5.7. Further, the default model was used.

All datasets where pre-processes as described in section 5.3, except the ATC dataset. As it consists of 8 single trajectories, it is statistically inadequate to split these few trajectories with the standard test train ratio. Therefore, the sliding window method is applied in the first step. Then, the resulting trajectories were shuffled and this resulting dataset was divided into a train and test split. Furthermore, the dataset was thinned out with a factor of 0.5 in order to reduce redundancy.

The optimisation did not lead to an *unique* optimal parameter configuration for each dataset. But this method revealed, that there exists a variety of configurations which resulted in equivalent model qualities. For example, a RNN with a depth of one and with 128 hidden dimensions trained for 70 epochs on the ETH datasets had an almost equal loss as a neural net with a depth of three with 1024 hidden dimensions trained for 150 epochs. But in the latter case, the training duration was several times higher than in the first case. Hence, the following summary is a combination of results from the optimisation and a manual selection of samples with a rather low training duration compared to similar performing samples.

Dataset	Depth d	Hidden Dimensions h	Learning Rate η	Batch Size b	Epochs e	Dropout p
Small Datasets: ATC, ETH, Hotel	1	128	0,001 - 0,003	32	70-150	0.0
Large Datasets: Vissim, Laser	1	128	0,001 - 0,003	32	200-500	0.0

Fig. 6-1: An initial parameter search led to a very basic RNN configuration. Equivalent performing configurations were dismissed when the sample training duration was higher.

6.2 Architecture Comparison

In order to compare the four different sequence to sequence architectures, the performance was measured using identical settings for each of them. Figure 6-2 shows the mean displacement of the four designs depending on the number of training epochs. All models were trained on the Laser II dataset and the prediction quality was evaluated on the test split. Evidently, the *Simple Sequence to Sequence* and the *Attention Sequence to Sequence* architectures are outperformed by the *Advanced Sequence to Sequence* and *Advanced Sequence to Sequence with Peek* designs. The bi-directional encoder in the *Attention* model seems not to be appropriate for positional sequences. This poor performance can be explained by the fact, that the movement of a pedestrian does not have any bi-directional dependencies unlike sequences from NLP tasks. Further, training such a model takes approx. three times longer to finish due to the high number of model weights (cf. Figure 6-3). Hence, it can be concluded that the *Advanced Sequence to Sequence* design with and without Peek are the most effective architecture for the trajectory prediction task. The Advanced Sequence to Sequence architecture without Peek was chosen as the default model. For simplicity, it is referred to as *sequence to sequence model* in the following.

As already stated before, GPUs accelerate the training computations of neural nets. The runtime measurements in Figure 6-3 underline the effectiveness of GPUs. Training a LSTM architecture on a CPU needs a significant higher training duration compared to a GPU. Hereby, the GTX 1070 and a Intel Westmere X5675 CPU were used to train the ATC dataset for 200 epochs. The encoder-decoder design had 128 hidden states an a depth of one.

6.3 Optimiser Comparison

Similar to the previous section, the different optimisers are compared by using a fixed datasets and identical settings except for the learning rates. The learning rate η for each optimiser follow

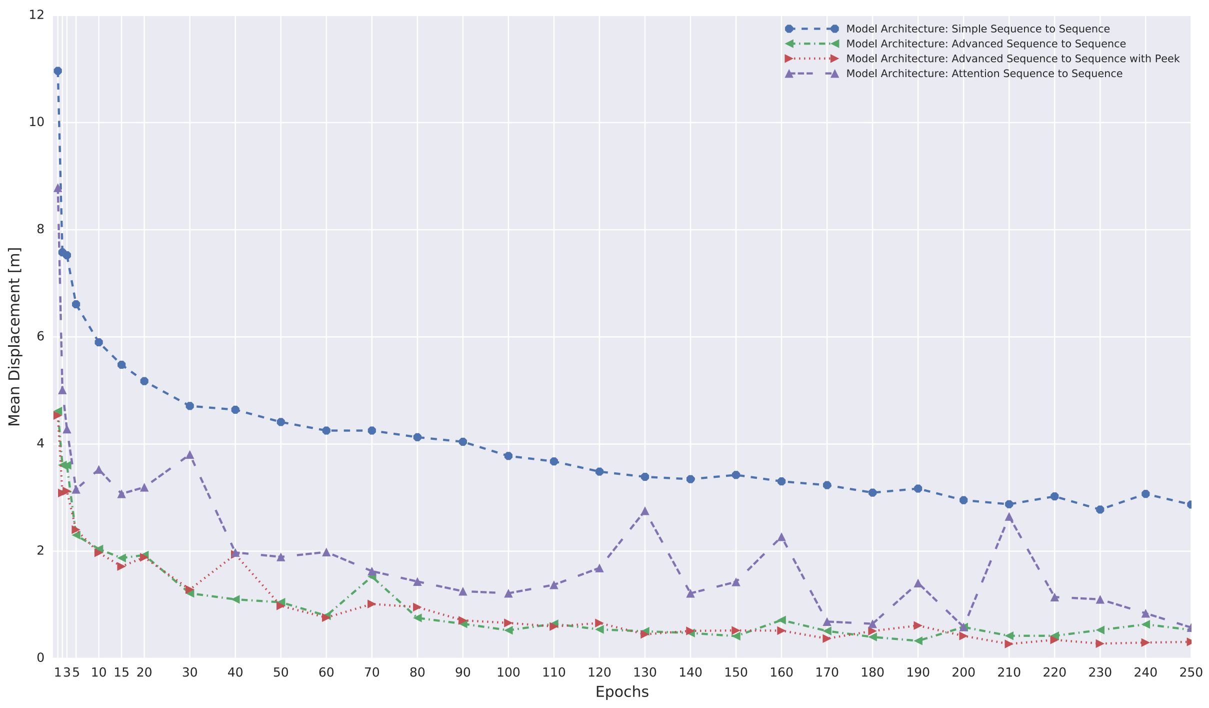


Fig. 6-2: Performance of the sequence to sequence architectures on dataset Laser II

Architecture	Training on CPU	Training on GPU	Parameters	Hidden Dimensions
Simple Sequence to Sequence section cf. 3.4.2	40 min	6 min	68120	128
Advanced Sequence to Sequence section cf. 3.4.1	124 min	7 min	199556	128
Advanced Sequence to Sequence with Peek cf. section 3.4.3	124 min	7 min	199556	128
Attention Sequence to Sequence cf. section 3.4.4	239 min	22 min	266243	128

Fig. 6-3: Sequence to sequence architectures statistics

those provided in the original papers (cf. Figure 6-5). A simple study on the ETH dataset shows that the Adam optimiser has the best performance, as shown in Figure 6-4. All other optimisers, except the standard SGD, perform slightly worse. Similar results where obtained using different datasets. On this basis, the Adam algorithm was chosen as the default optimiser for the following experiments. Note that there exists the possibility to tune η for each optimiser, but for simplicity the default values were taken.

6.4 Comparison with Baseline Models

In order to assess the prediction quality of the sequence to sequence model, two baseline models were implemented and applied on the datasets.

The first baseline model is a Kalman filter with a constant velocity movement model. The obser-

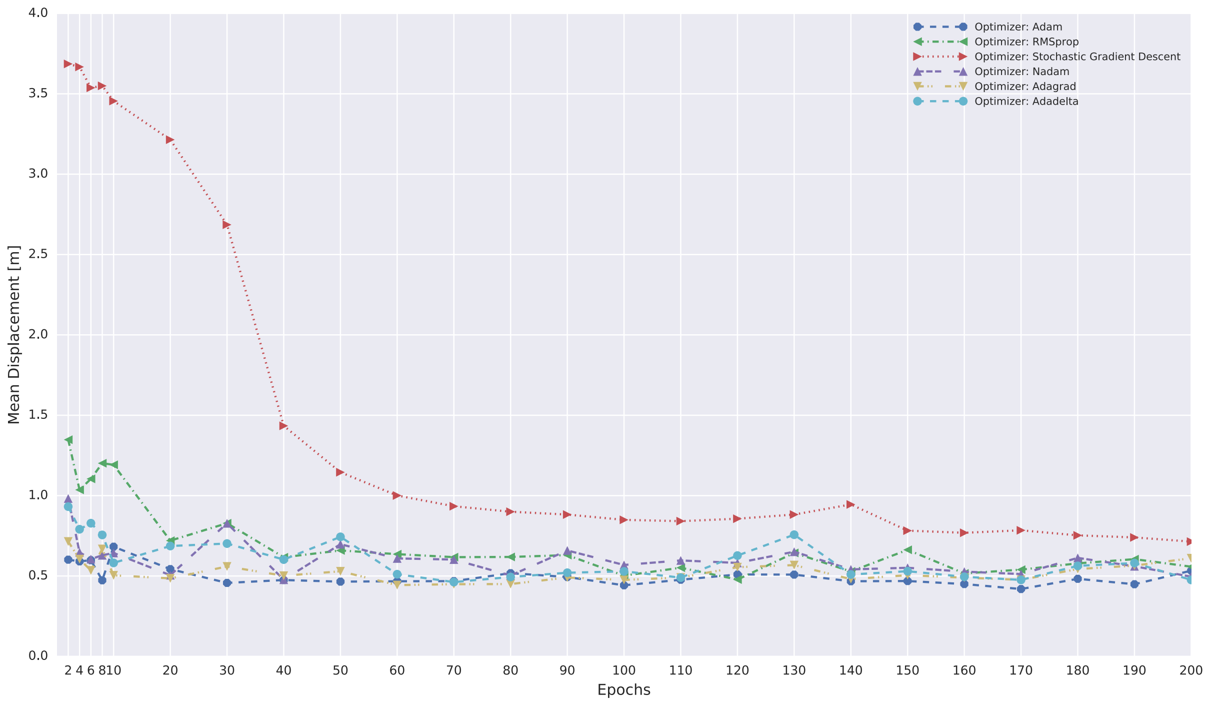


Fig. 6-4: Performance of different *optimisers* on dataset ETH

Optimiser	Default η
Adam	0.001
RMSprop	0.003
SGD	0.01
Nadam	0.002
Adagrad	0.01
Adadelta	1.0

Fig. 6-5: Optimiser default learning rates

vation 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 second baseline model is a FNN, in particular a CNN. A special architecture with 3 convolutional layers and an up-sampling layer allows an end-to-end learning in a sequence to sequence manner. The convolutional layers have a kernel size of eight and a linear activation function is deployed.

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 encoder-decoder architecture are obtained by hyper-parameter optimisation. The uncertainty parameters in the Kalman filter where also determined through an optimisation for each dataset. Further, the

FNN was roughly tuned and an early stopping mechanism was applied. Both, baseline and sequence to sequence architectures obtain identical preprocessed trajectories. This should ensure comparability among them. A train split size of 70% was used. The ATC dataset was pre-processed as described in section 6.1.

The data in Figure 6-6 indicate, that the RNN outperforms the baseline models on all datasets. In particular, the performance of the RNN is significantly better in the case of the Laser datasets as on all other datasets. This can be explained by the different average velocities in the datasets. The tracked objects in the Laser datasets have a much higher object speed compared to the other ones (cf. Attachment 11.4). Thus, prediction inaccuracies by the linear and the FNN model lead to a much worser overall score compared to the pedestrian datasets. Moreover, it is quite plausible that the Kalman filter performs especially poor in winding datasets. That applies for ATC, Vissim Case 3 and the laser datasets.

In the case of the Vissim datasets, the RNNs perform remarkably well. This is caused by the density, homogeneity and redundancy of these simulated trajectories. Hence, the LSTM is able to develop is full memory capability on these rather unrealistic datasets. The trajectory collections ETH, Hotel and ATC have approximately the same number of captured data points and a similar pedestrian average speed. Nevertheless the performance on ATC is the worst compared to the other two. Apparently, the RNN has a higher prediction error for curvy trajectories.

A comparison with the paper of Alahi et al. [ALA16] yielded, that the prediction error of the sequence to sequence architecture has a similar order of magnitude as the vanilla LSTM on the ETH datasets. But the model performance is worse than the *Social LSTM* approach. Note that it is not evident whether the metrics and the preprocessing are identically implemented as in the work of Alahi et al.

Figures 6-7 and 6-8 visualise the results of the Kalman Filter, the CNN and the RNN. In addition, the real future trajectory and the corresponding observation is shown. These figures indicate, that the RNN is able to approximate the ground truth much more accurately than the baseline models. It is clearly visible that the predicted path is fitted on the underlying training data (cyan-coloured in the background). Further, the prediction of the CNN has a much more volatile behaviour compared to the other predictions. It underlines that this type of ANN is not supposed to process sequential data.

Metric	Dataset	Kalman filter	Sequence to Sequence	Feedforward Neural Network
$\mathcal{L}_{MSD}$	ETH	0.651	**0.411**	0.522
	Hotel	0.251	**0.140**	0.259
	ATC	2.380	**0.729**	1.766
	Vissim Case 1	0.309	**0.011**	0.173
	Vissim Case 2	0.875	**0.012**	0.289
	Vissim Case 3	1.290	**0.011**	0.268
	Laser I	21.216	**1.100**	18.344
	Laser II	55.941	**0.717**	25.914
	Laser III	21.134	**1.396**	20.058
$\mathcal{L}_{MD}$	ETH	0.566	**0.504**	0.593
	Hotel	0.280	**0.272**	0.386
	ATC	0.962	**0.590**	0.949
	Vissim Case 1	0.192	**0.034**	0.341
	Vissim Case 2	0.521	**0.059**	0.418
	Vissim Case 3	0.653	**0.042**	0.419
	Laser I	3.214	**0.817**	3.392
	Laser II	4.554	**0.614**	3.833
	Laser III	2.912	**0.815**	3.259
$\mathcal{L}_{MFD}$	ETH	1.127	**0.876**	1.092
	Hotel	0.531	**0.410**	0.593
	ATC	1.926	**1.097**	1.781
	Vissim Case 1	0.456	**0.076**	0.533
	Vissim Case 2	1.159	**0.114**	0.749
	Vissim Case 3	1.492	**0.086**	0.669
	Laser I	6.966	**1.293**	5.726
	Laser II	9.785	**0.873**	7.325
	Laser III	6.095	**1.346**	6.222

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

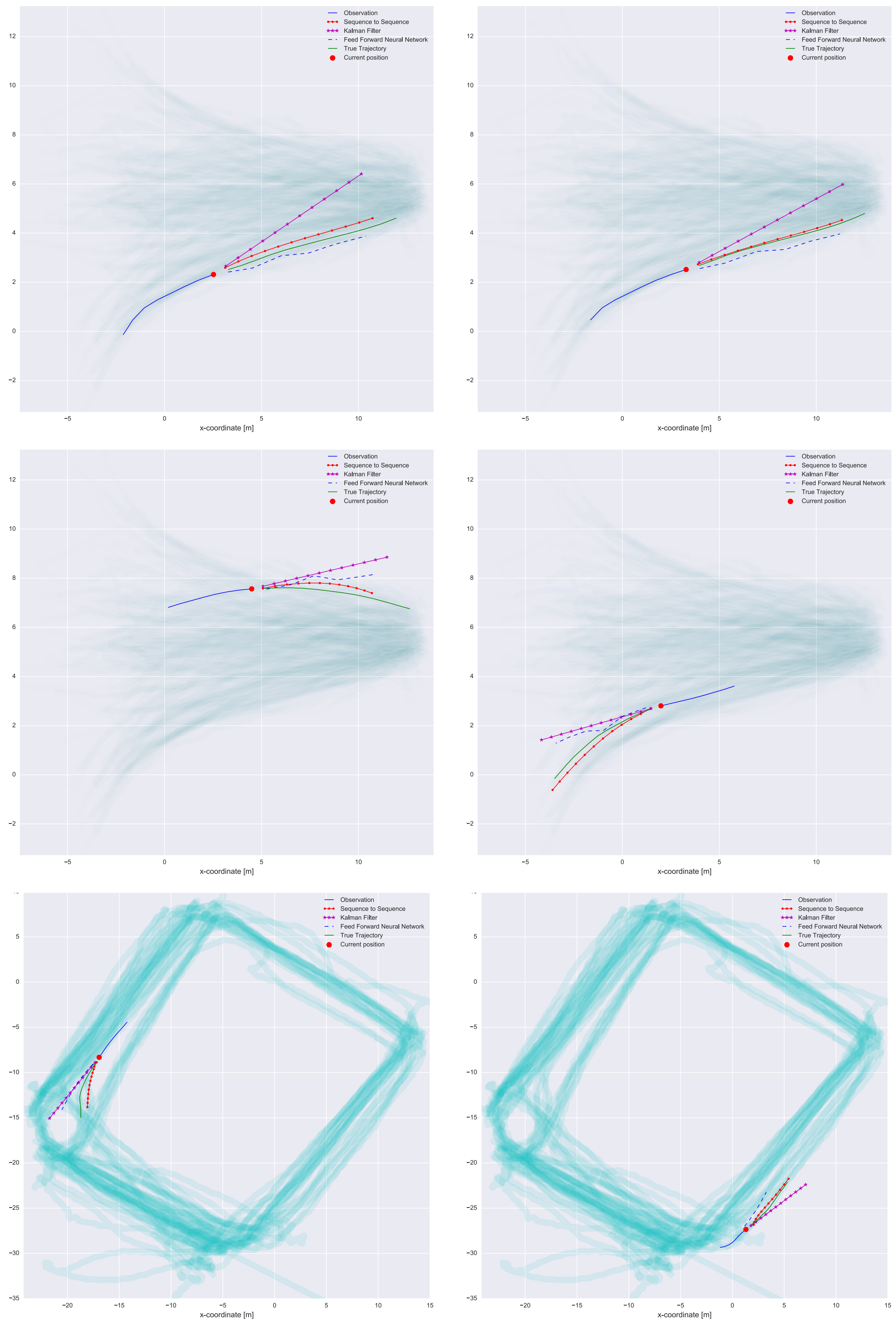


Fig. 6-7: Sequence to sequence prediction in comparison with baseline models on datasets ETH and ATC

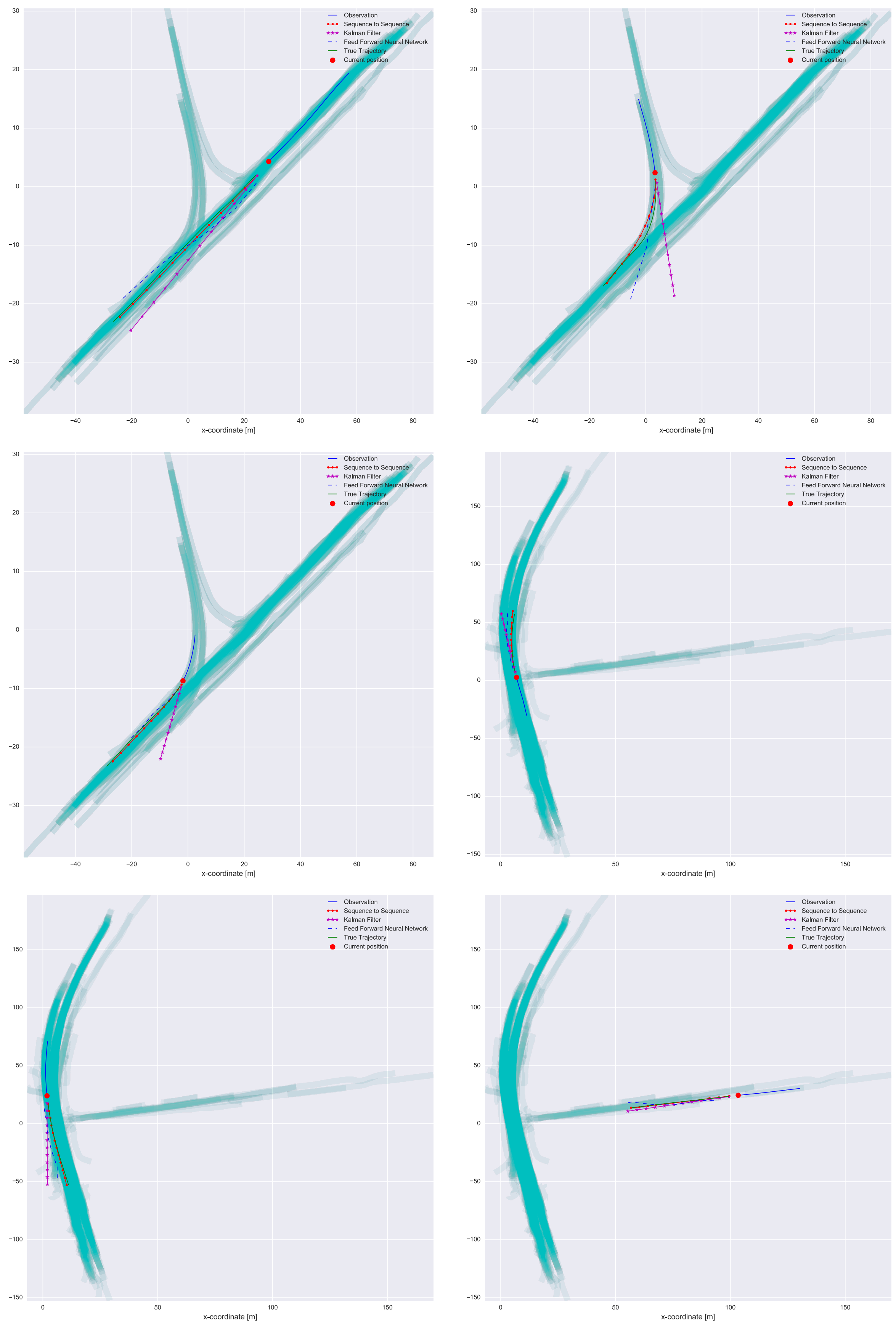


Fig. 6-8: Sequence to sequence prediction in comparison with baseline models on datasets Laser I and II

6.5 Parameter Dependencies

In the following, some of the most important model parameters are examined in order to obtain a better understanding for their impact on the neural net's performance. To achieve this, the sampling methods described section 5.8 were applied.

In general, it can observed that some parameter responses have a slight scatter due to the existence of local minima in the loss function. But, in most cases it is possible to recognise the underlying trend of the particular configurations. A more statistically valid approach would be the application of a Monte Carlo method. But this approach is impracticable to implement due to the enormous resulting computational runtime.

Depth Sensitivity

Recent studies on machine learning have shown that *"deep"* RNNs outperform shallow RNNs for NLP tasks [HER13] [SUT14]. As can be seen from Figure 6-9, this does not apply to our setting. Additional stacks on the encoder-decoder architecture do not improve the prediction quality of the LSTM architecture. However, stacked LSTMs have a higher number of model weights which causes higher computational cost. In this given example, executing 200 epochs with a depth of three took also three times longer to finish than with with a depth equal to one.

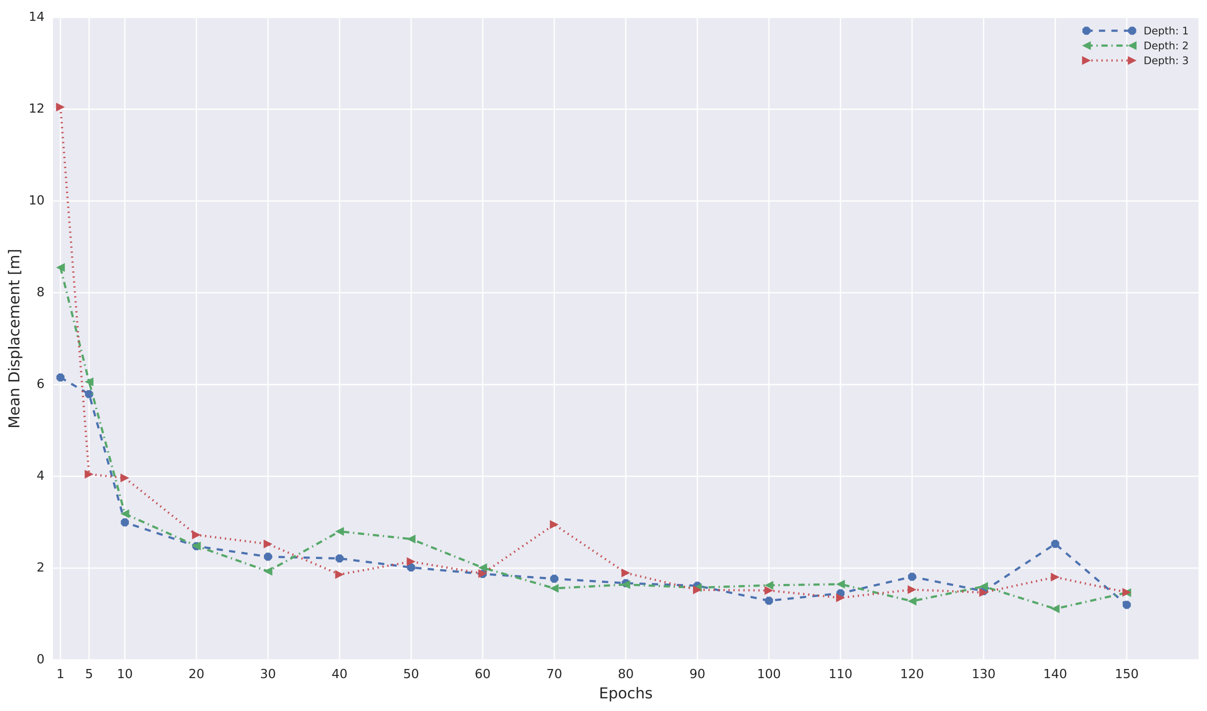


Fig. 6-9: Sensitivity of the encoder-decoder *depth* on dataset Laser III

Batch Size

At the beginning of this thesis, the training *batch size* was a freely tuneable hyperparameter with unknown effect on the net's performance. Figure 6-11 indicates that sequence to sequence architecture trained with lower batch sizes perform slightly better in comparison with models trained with higher batch sizes. Nevertheless, likewise in the previous case, this parameter influences directly the training runtime, as indicated in Figure 6-10. As a compromise between runtime and model accuracy, a batch size of 32 was chosen as default value in all other experiments.

Batch size	8	16	32	64	128
Runtime	53 min	26 min	15 min	9 min	5 min

Fig. 6-10: Different batch size and the corresponding training duration for dataset Laser I using 200 epochs and 128 hidden dimensions.

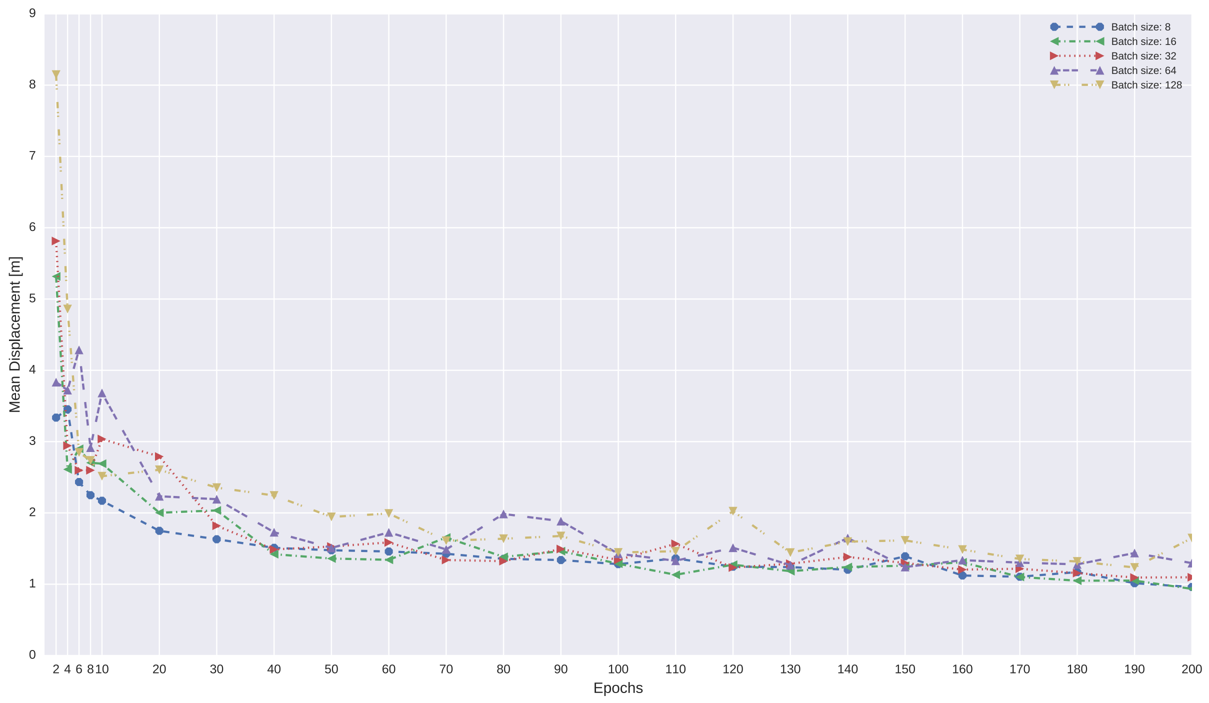


Fig. 6-11: Sensitivity of the training *batch size* on dataset Laser I

Learning Rate

The influence of the *learning rate* on the training progress was investigated by using the Adam optimiser and variations of its default learning rate $\eta = 0.001$. Figure 6-12 suggests that there is no discernible trend whether higher or lower learning rates have a positive effect on the model's quality. It appears that the adaptivity of the algorithm enables robust training without an in-depth learning rate tuning.

Dropout

Diverse studies have shown that *dropout* can improve the performance of neural networks, this applies in particular to CNN. Further, it prevents the neural net from overfitting [SRI14]. Adding dropout to our setting had no unequivocal impact on the models performance, as indicated in Figure 6-13. In general, the preprocessing step sliding window causes that the train dataset

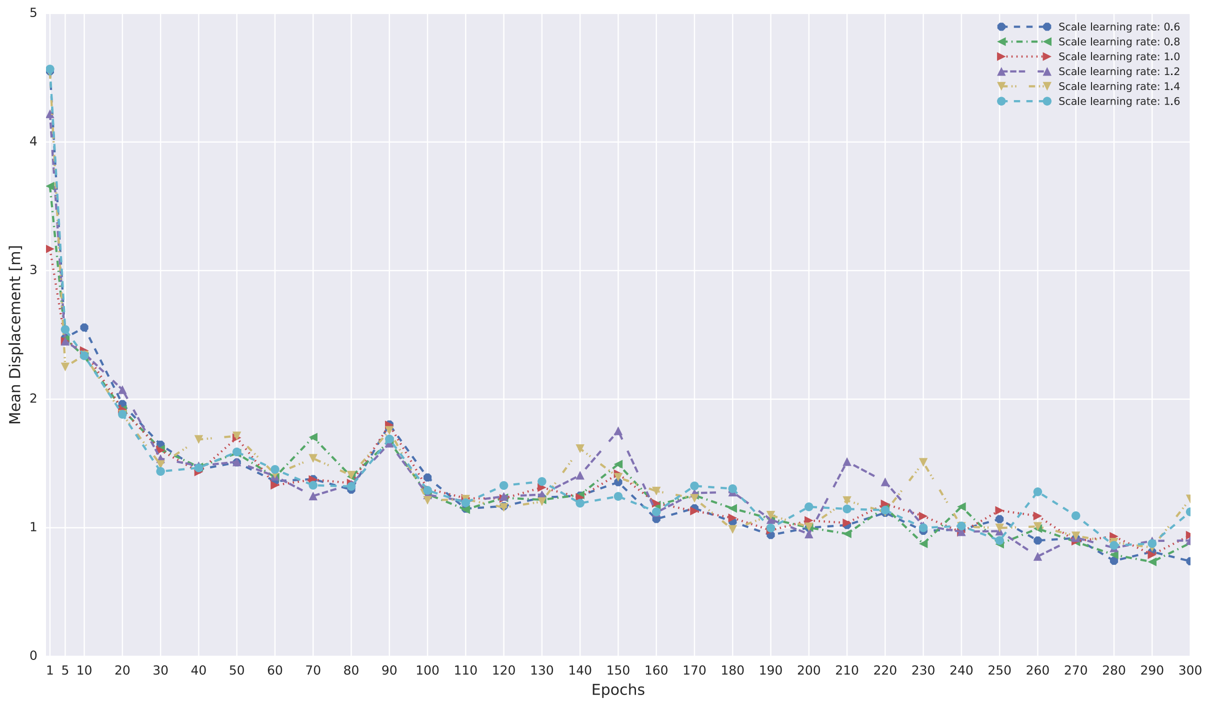


Fig. 6-12: Sensitivity of the *learning rate* of the Adam optimiser using dataset Laser III

consists of partially very similar trajectories and this redundancy could naturally prevent the net from overfitting [JAI15]. Hence, an additional regularisation method such as dropout is not necessary. However, a considerable disadvantage of activating dropout is that it halves the training speed. As a result, dropout was deactivated by default for all other experiments.

Observation Length & Hidden Dimensions

Another interesting aspect of the whole approach is the influence of the *observation length* N_{obs} on the prediction quality. To investigate the observation length in combination with the *hidden dimension* of the RNN, the two-dimensional sampling method was applied. As can be seen in Figure 6-14, the prediction network needs at least two data points in order capture the movement dynamics of the tracked object. Providing only a single datapoint causes a large prediction error. Subsequently, higher observation times lead to a better model prediction. A similar behaviour was observed for all other datasets. To ensure comparability between literature and this thesis, the default observation length was set to $N_{obs} = 8$ that corresponds to $3.2\ [s]$ using a sample rate of $2.5\ [Hz]$.

Moreover, Figure 6-14 shows that the RNN with 128 hidden dimension performs slightly better compared to the same network with 64 and 32 hidden dimensions. Additional studies have proven that the sequence to sequence model with 128 hidden dimensions had good cost-performance ratio with the result that this value is used by default.

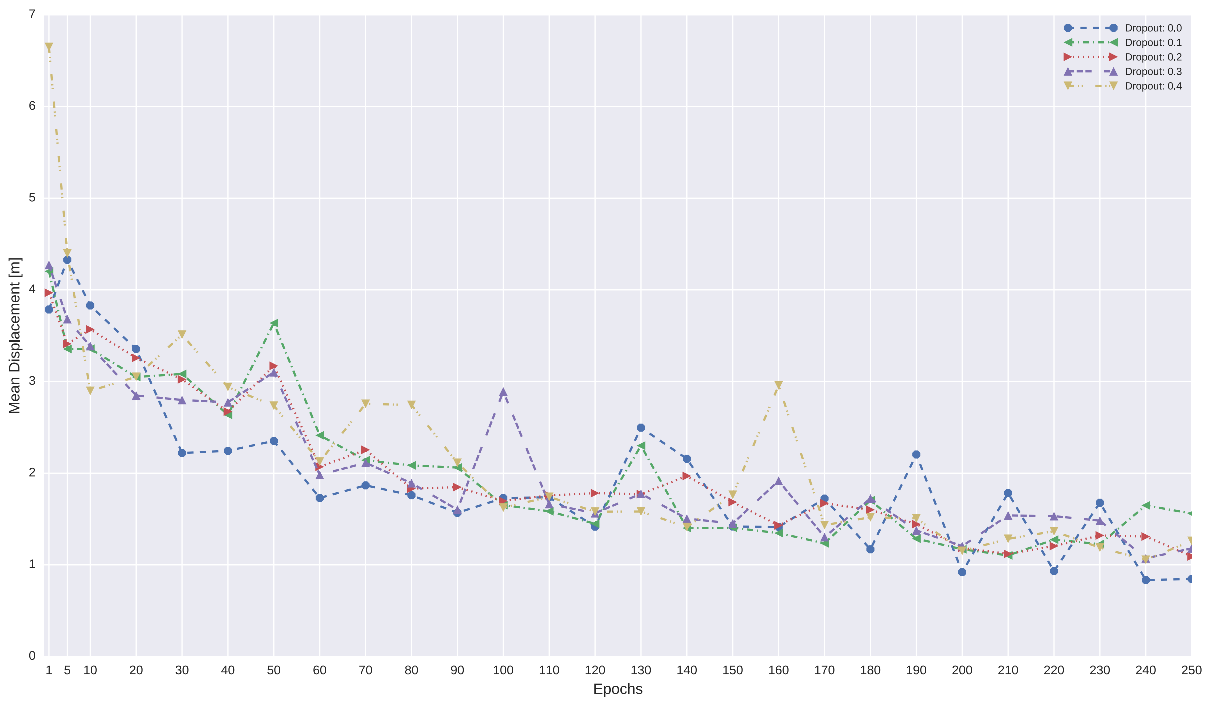


Fig. 6-13: Sensitivity of *dropout* using dataset Laser III

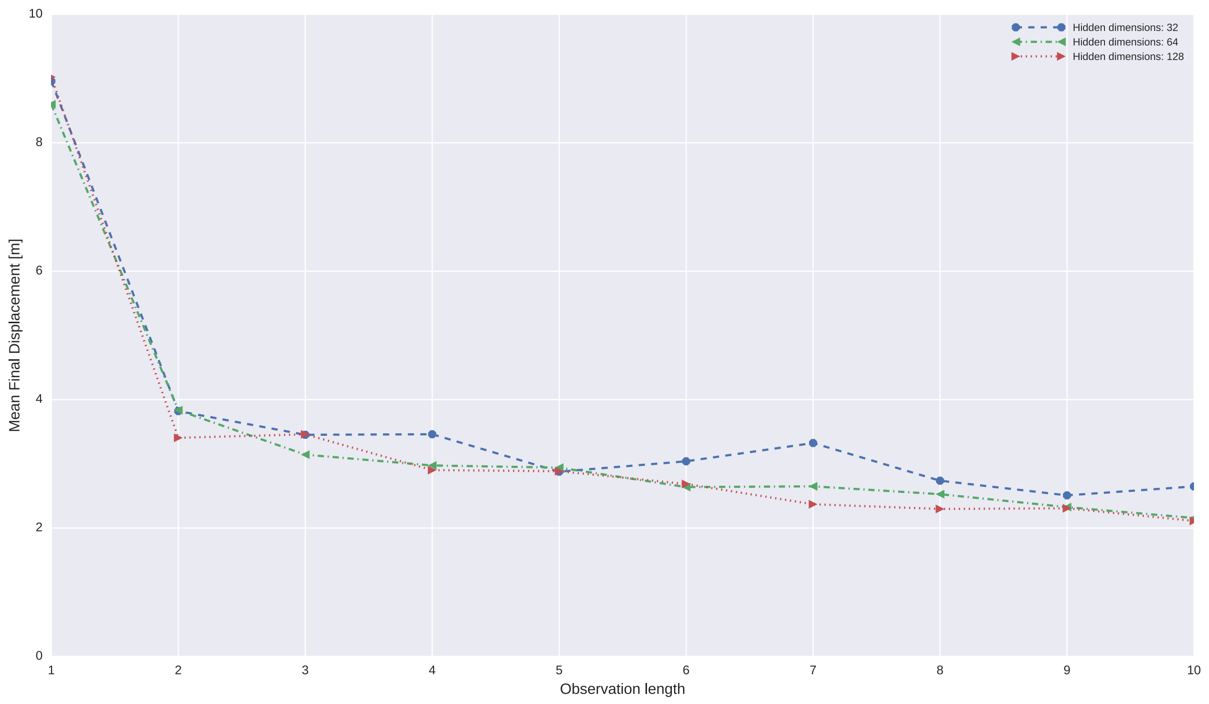


Fig. 6-14: Sensitivity of the *observation length* and the number of *hidden dimensions* on dataset Laser I. Every sample was computed with 200 epochs.

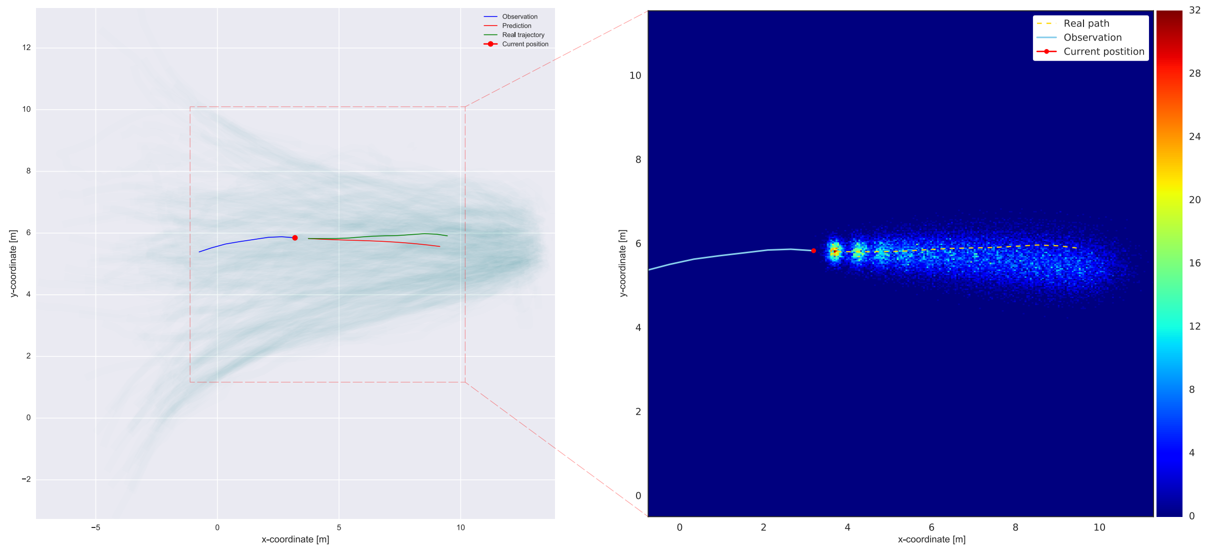
6.6 Prediction Stability

The prediction stability analysis is performed by using the presented uncertainty analysis in section 5.9. Each dataset was investigated with the default parameter set and the sequence to sequence model. The training was stopped when the prediction quality converged. The resulting stability heat maps were manually examined. It was generally observed that the model prediction did not show any serious instabilities. A few chosen examples of the stability visualisations are depicted in Figure 6-15 and 6-16.

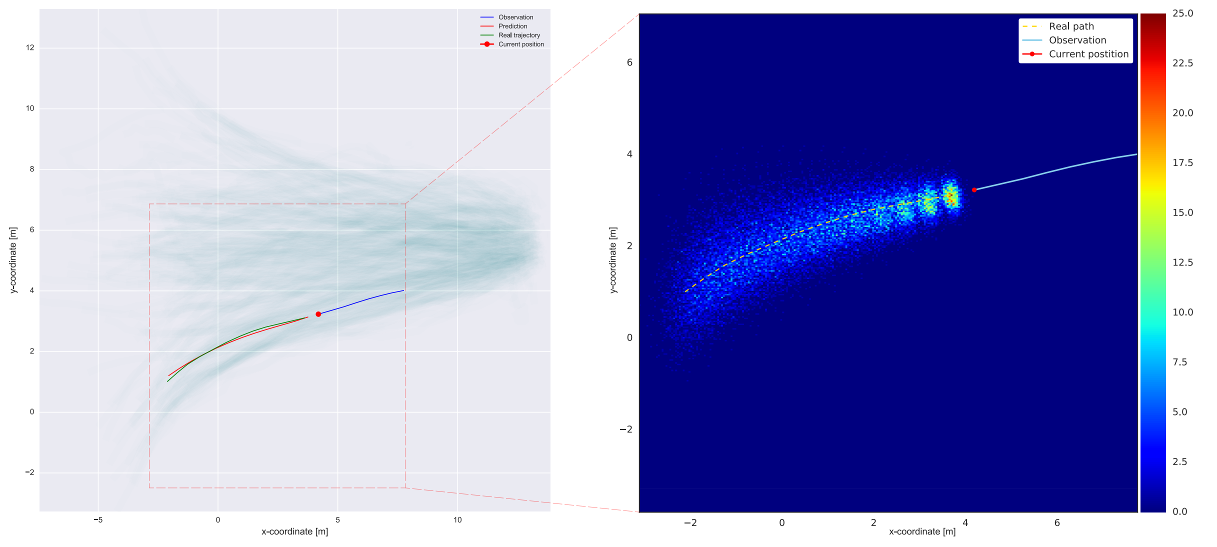
Figure 6-15i indicates that the predictions of the perturbed observation trajectories of one moving individual results in a distribution with a conical shape. The model prediction can be categorised as a *stable*, because distribution is located in the close proximity of the ground truth. This applies for also an pedestrian moving in the direction. But, due to the diverging movements on the left half of the dataset, the distribution of the “leaving pedestrian” has a much bigger range than in the previous example (cf. 6-15ii). The predicted distribution for a standing pedestrian is indicated in Figure 6-15iii. This example was chosen, because the observation trajectory is located in an area with a sparse trajectory density. However, the positional variations seem to be plausible, when regarding the maximal extent of the predictions. Thus, a regional lack of training data does not lead necessarily to an unstable model behaviour. Other examined samples of this dataset underline this conclusion.

Prediction models that were obtained by training one of the laser datasets have slight different characteristics than in the previous cases. This is caused by the fact, that these datasets contain mainly vehicles with high velocities. That induces a smoother movement pattern compared to pedestrian trajectories. Figure 6-16i depicts a right turn manoeuvre and the corresponding heat map implies that the model predicts a variety of different curves in the close proximity of the real future path. Furthermore, twelve clusters are visible which is equivalent to the prediction length. This implies that the predicted trajectories have an approximately equal velocity. This pattern can also be observed in Figure 6-16iii, but with a much more precise prediction. From this it appears that predicting vehicle movements is much more reliable. Similar results where obtained on the other laser datasets.

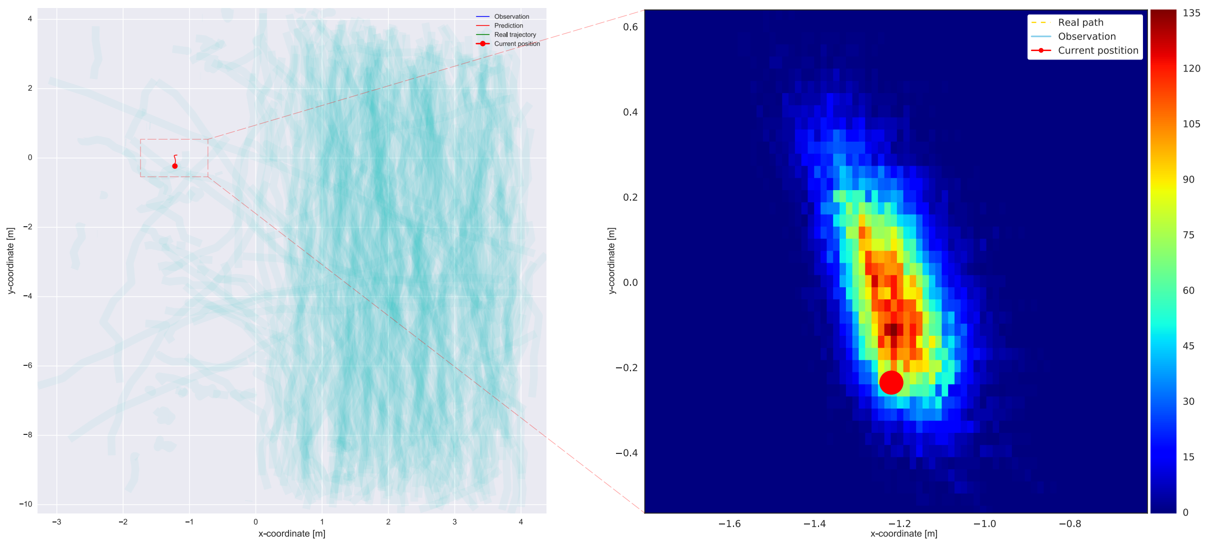
It is important to stress, that the above presented examples are believed to be representative samples. It cannot be excluded that certain model inputs could cause an unstable model response. Thus, a more statistically valid method has to be elaborated and applied in future investigations.



(i) Arriving pedestrian on dataset ETH

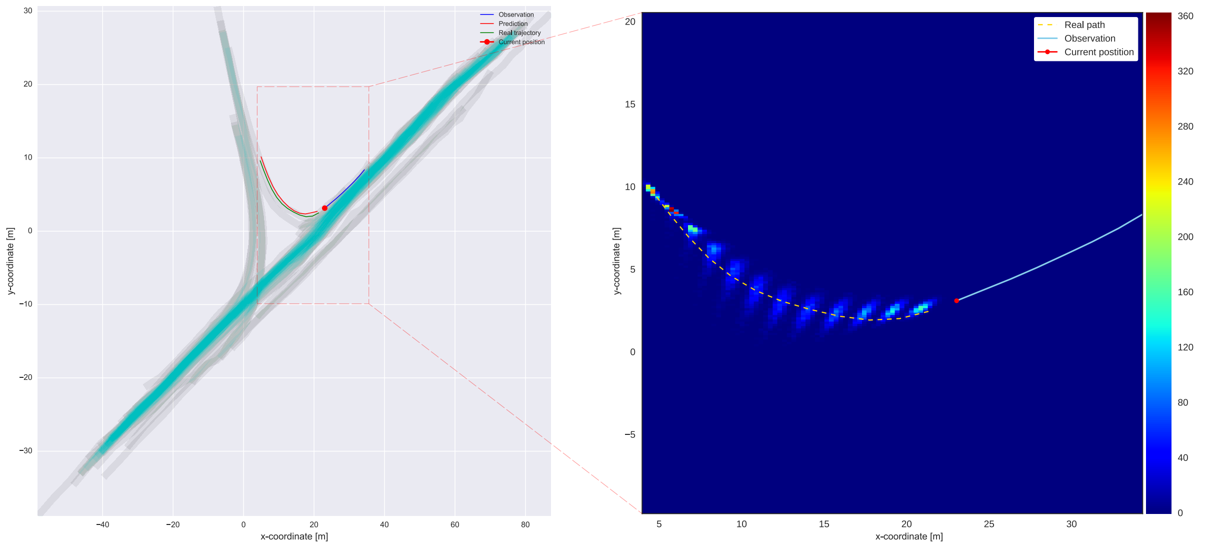


(ii) Leaving pedestrian on dataset ETH

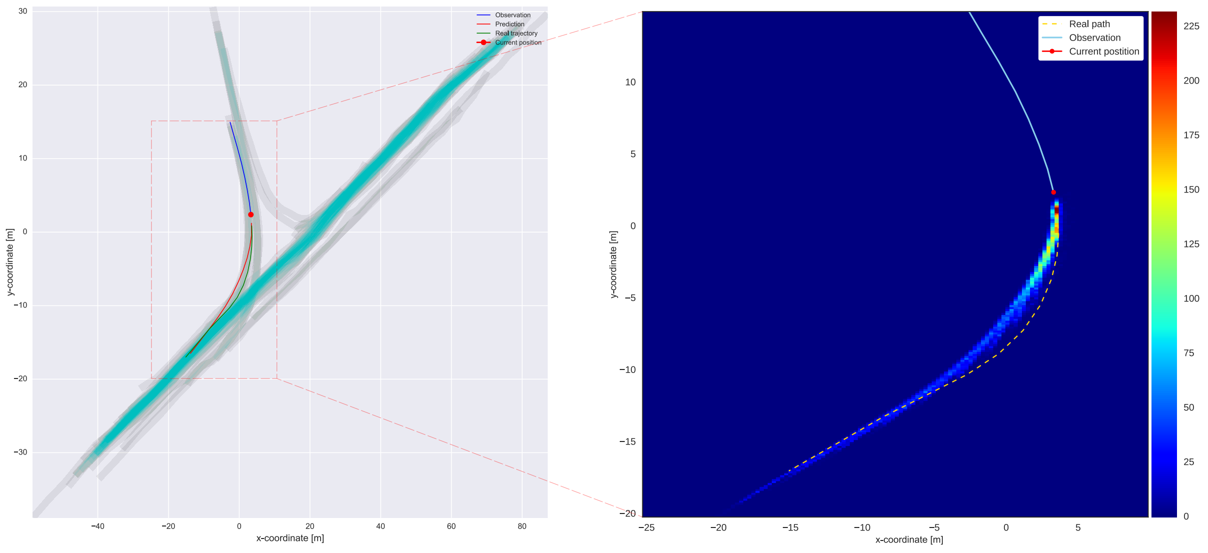


(iii) Standing pedestrian on dataset Hotel

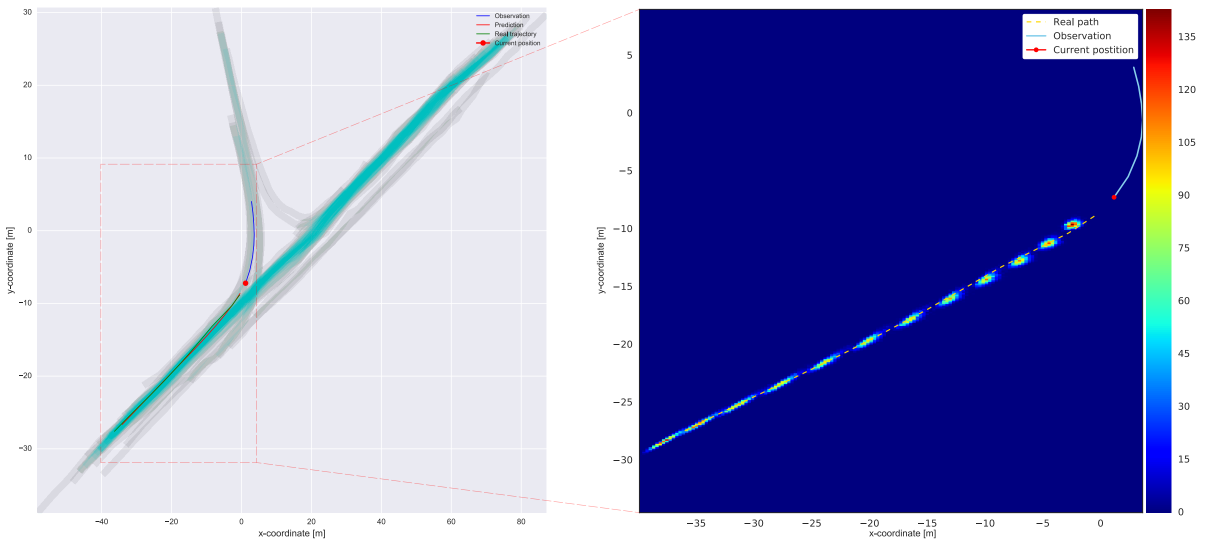
Fig. 6-15: Model prediction stability using dataset ETH and Hotel



(i) Right turn with small scatter in the prediction



(ii) Right turn with stable prediction

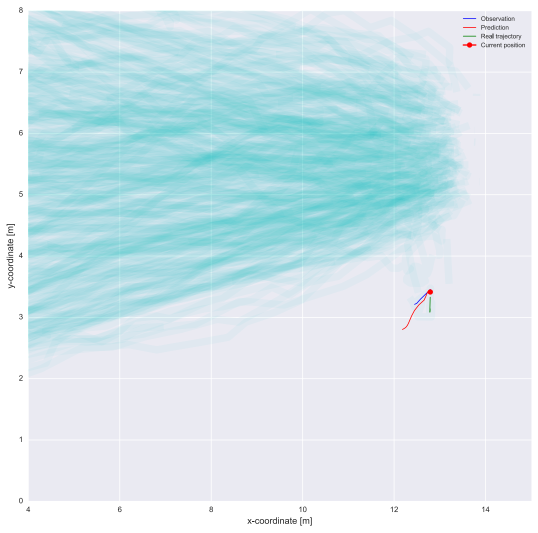


(iii) Right turn with stable prediction

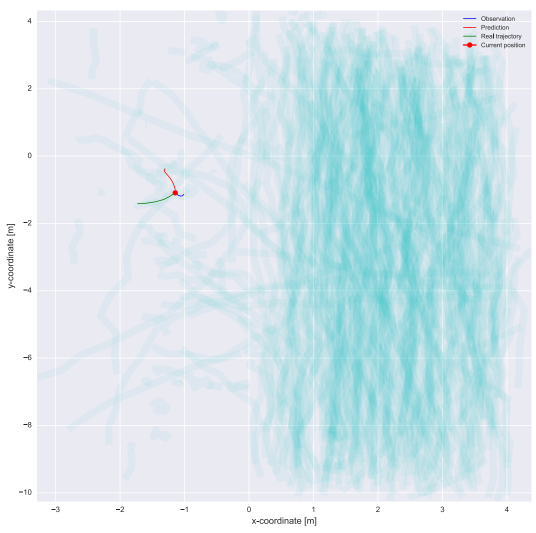
Fig. 6-16: Model prediction stability using dataset Laser II

6.7 Incorrect Predictions

The prediction quality of the sequence to sequence approach is directly dependent on the density of the training data. Hence, a regional lack of captured pedestrian movements results in poor predictions in these certain areas. Figure 6-17i and 6-17ii depict two examples for an incorrect model prediction. Both figures show that the RNN is not able to generalise the underlying data. A possible workaround for this issue would be to use an alternative prediction model, for instance a linear model, in areas with sparse data.



(i) Example for a poorly predicted pedestrian trajectory on dataset ETH



(ii) Example for a poorly predicted pedestrian trajectory on dataset Hotel

Fig. 6-17: Bad model predictions due to a lack of regional training data

6.8 Continuous Learning

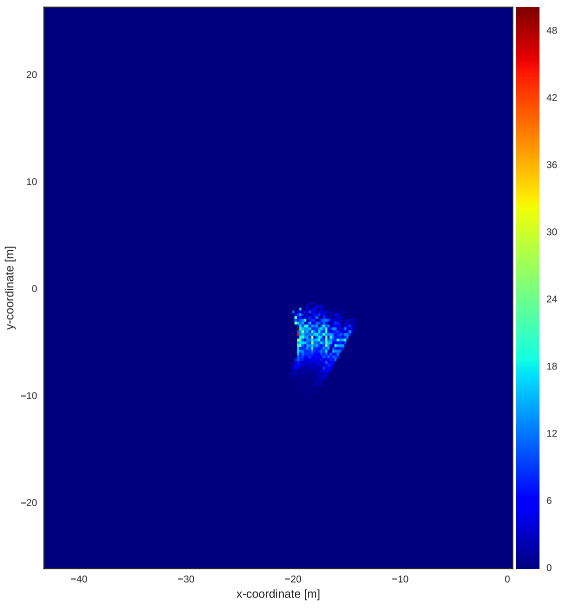
The results of the continuous learning approach are presented in the following sections. Hereby, two different intersection scenarios are simulated by using the synthetic Vissim datasets.

6.8.1 Case "Construction Work"

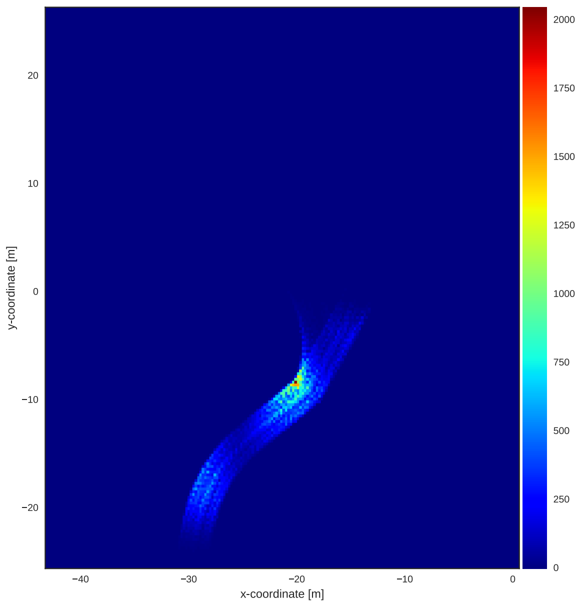
The following continuous learning experiment was performed with dataset *Vissim Case 1* as split I and *Vissim Case 2* the second scenario. By this means, a virtual event $\mathcal{A}$ is introduced that forces the pedestrians to move differently on the lower sidewalk (cf. Figure 11-3ii).

Due to the homogeneity, redundancy and the density of the synthetic datasets, the initial training of $f^I(\cdot, w)$ leads to a very well-fitted model. Figure 6-18i visualises the squared error for each predicted datapoint on basis of a test dataset taken from split I. The coloured partition of the heat map represent the error between the predictions and the ground truth trajectories. The highest prediction error occurs in the bifurcation area where the pedestrian movement diverges. The error vanishes in all other regions.

Now we consider the occurrence of event $\mathcal{A}$. Subsequently, the prediction error of split II is measured and the resulting error map is show in Figure 6-18ii. Clearly, a prediction error is induced by the changed movement behaviour in the close proximity of the virtual obstacle.



(i) Prediction error map of $f^{\mathrm{I}}(\cdot, w)$ evaluated on a test dataset taken from split I



(ii) Prediction error map of split II evaluated on $f^{\mathrm{I}}(\cdot, w)$

Fig. 6-18: Error Maps of split I and split II evaluated on $f^{\mathrm{I}}(\cdot, w)$

On this basis, a continuously learning prediction model would detected the abnormal high prediction error in certain areas. Hence, in order to improve the models's prediction, a retraining of $f(\cdot, w)$ would be initiated. To determine the effectiveness of the different retraining methods proposed in section 5.10, the cost-performance ratio for each of them is investigated in the following.

As cost we understand the retraining duration that dependents linearly on the training dataset size whereas performance is defined by a loss function $\mathcal{L}$. As we know, all proposed methods combine trajectories from split I and II in order to retrain $f^{\mathrm{I}}(\cdot, w)$. Hereby, only a fraction α^{II} of split II is added to the new train dataset, because it should be avoided that the new dataset becomes too large. Therefore, the sensitivity of α^{II} with respect to the prediction quality for each approach is investigated.

For this purpose, fraction α_i^{II} is varied from 0 to 1 by

$$\alpha_i^{\mathrm{II}} = 0.05 * i \quad \forall\, i = 0, \ldots, 20 \qquad \text{Eq. 6-1}$$

and the retraining is performed for each of these values. Note that training the initial model $f^{\mathrm{I}}(\cdot, w)$ needs 500 epochs, whereas the retraining takes 250 epochs. Other settings stayed fixed. To determine the "cost" for each method, the retraining duration was measured during the experiment. All methods, except *Sliding Window*, need for an $\alpha_1 = 0.05$ 45 minutes and for an $\alpha_{20} = 1$ 90 minutes to retrain. *Sliding Window* has a constant sized retraining dataset size and thus a fixed training duration of 45 minutes.

Figure 6-19 depicts the results for different retrain methods for all α_i^{II}. One can see, that the methods *Random Trajectories* and *Sliding Window* have very similar behaviour. Both approaches need only a fraction $\alpha_1^{\mathrm{II}} = 0.05$ of split II to improve the model's predictions significantly. Furthermore, the model improves only slightly, when more than 5 % of the trajectories are added to the training dataset.

The method *Worst Trajectories* performs marginally worse than *Random Trajectories* and *Sliding Window*. A similar performance is reached, when more than 40% of the worst observation--prediction pairs of split II are added to the training dataset. This corresponds to a runtime of 62 minutes. Figure 6-20iii and Figure 6-20vi give an explanation for this behaviour. The worst 5% and 10% trajectories of split II are located on a relative small area compared to the covered pathways in Figure 6-20v and in Figure 6-20vi. Hence, during the retraining with the *Worst Trajectories* method, the afflicted area is not fully covered. As shown in Figure 6-20ii, a small prediction error remains in the proximity of the "construction work" after retraining with $\alpha^{\mathrm{II}} = 0.05$. Thus, the *Worst Trajectories* method has in this special setting an inferior price-performance ratio in comparison with the two previous methods.

Note that the method *Best Trajectories* is intended to be a reference. One can see, that the model performance on split III is not improving before $\alpha_i^{\mathrm{II}} > 0.55$. Evidently, training only good performing trajectories from split II does not continue to improve the model. This method reaches a similar prediction quality for $\alpha^{\mathrm{II}} > 0.9$. At this point almost the whole split II and split I are jointly trained, which leads to an overall retraining duration of 85 minutes.

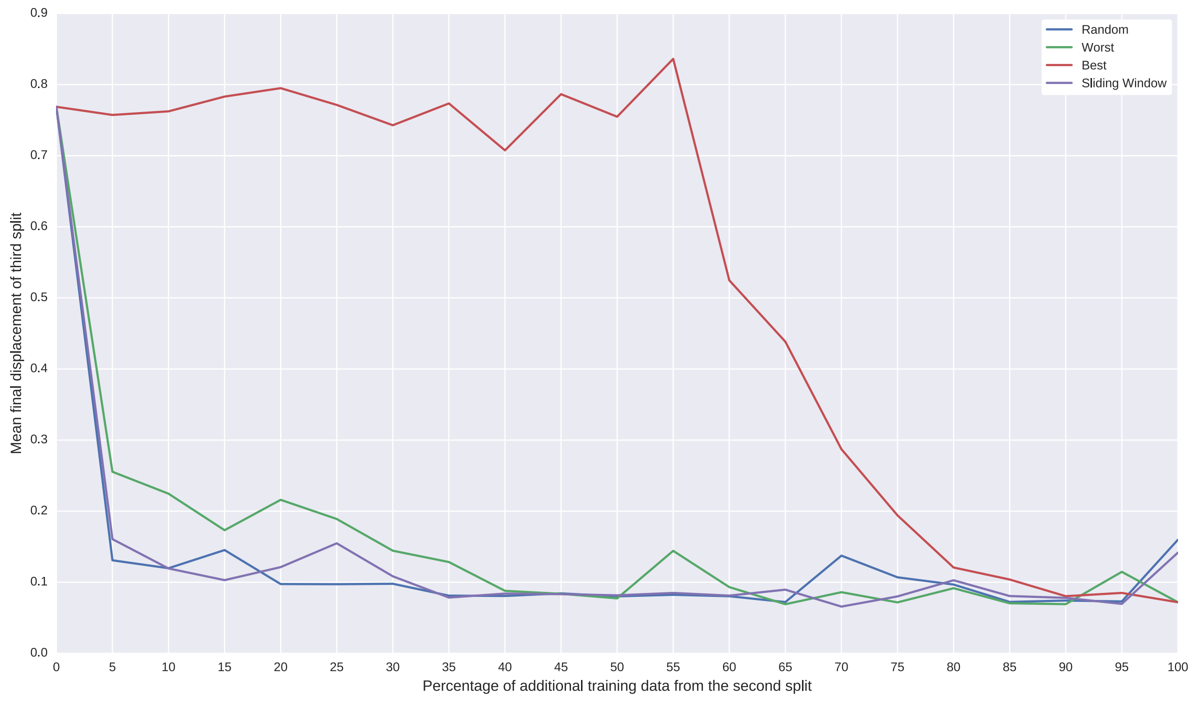
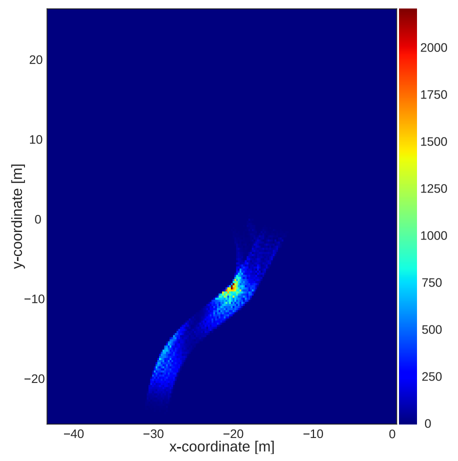
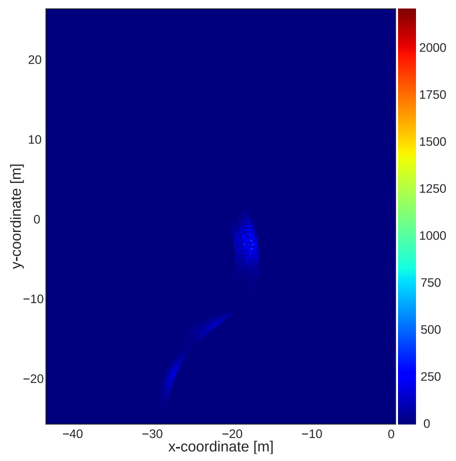


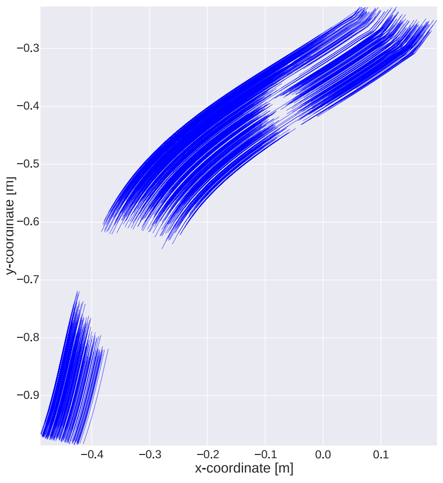
Fig. 6-19: Sensitivity of $\alpha_i^{\mathrm{II}} * 100\%$ on the model's prediction performance on split III. The retrain methods *Random Trajectories*, *Worst Trajectories*, *Sliding Window* and *Best Trajectories* (only as reference) were used.



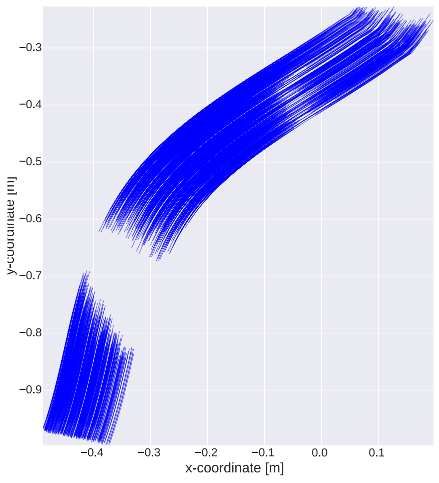
(i) Prediction error map of split III after training split I



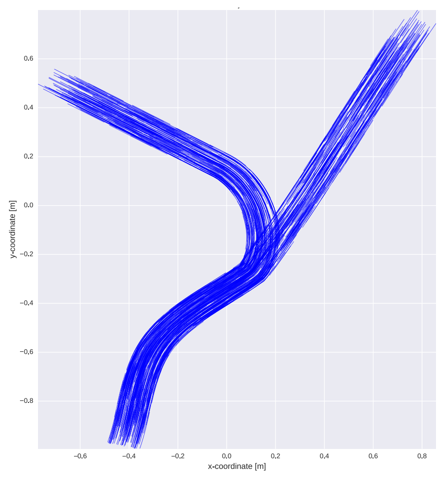
(ii) Prediction error map of split III after training split I merged with the worst 5% observation trajectories of split II



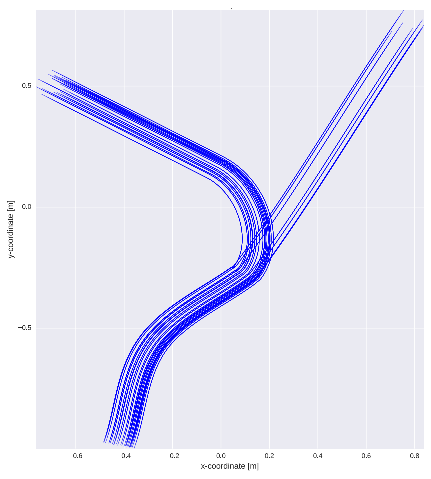
(iii) Worst 5% observation trajectories of split II



(iv) Worst 10% observation trajectories of split II



(v) Random: 5% random observation trajectories of split II



(vi) Sliding Window: First 5% of the observation trajectories of split II

Fig. 6-20: The continuous learning setting in the case “Construction Work”

6.8.2 Case “Blocked Road”

In the second experiment, dataset *Vissim Case 3* is divided equally into split II and split III. As in the previous experiment, split I is represented by *Vissim Case 1*. This setting is intended to simulate a blocked road scenario (cf. Figure 11-3i and Figure 11-3iii).

Due to the extraordinary different movement pattern, a high prediction error occurs in the area where the pavement of the pedestrians is blocked. This is clearly visible in Figure 6-22i. Note that the squared error is added in the cells of the heat map, so that this experiment has a maximal error which is many times higher than in the first experiment. Subsequently, the trajectories with the worst prediction performance are located in the same zone (cf. Figure 6-22ii).

The qualitative results for the methods *Sliding Window* and *Random Trajectories* are identical to the experiment above. Retraining the model with a few additional trajectories from split II leads immediately to a good model performance, which is evident in Figure 6-21. In contrast, retraining the model with the worst observation-prediction pairs leads to a significant inferior performance. Figures 6-22ii, 6-22iii and 6-22iv indicate that the area covered by the worst trajectories is much smaller compared to the covered area of the two other methods. Hence, for small α^{II} only a limited area located in the curve is improved but not the error-prone surrounding area. Thus, retraining with the *Worst Trajectory* method is less effective in this special setting.

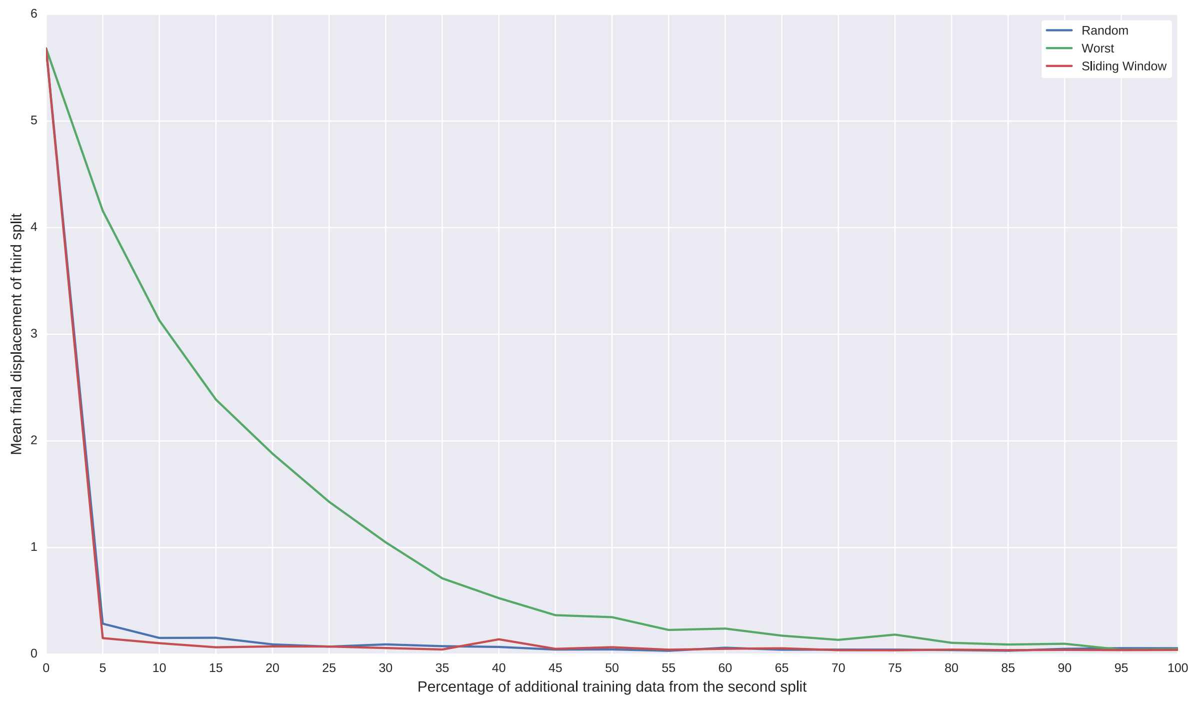
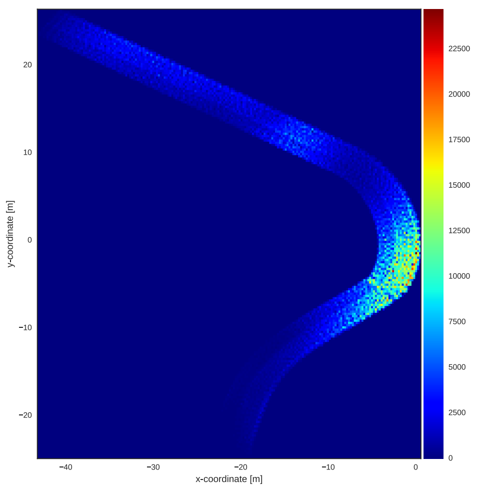
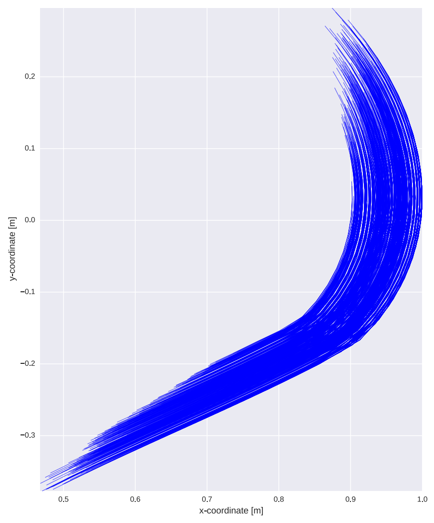


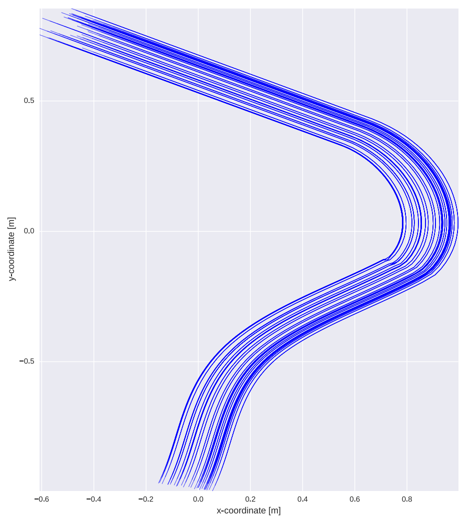
Fig. 6-21: Impact of $\alpha_i^{\mathrm{II}} * 100\%$ on the model’s prediction performance of split III. The retrain methods *Random Trajectories*, *Worst Trajectories* and *Sliding Window* were used.



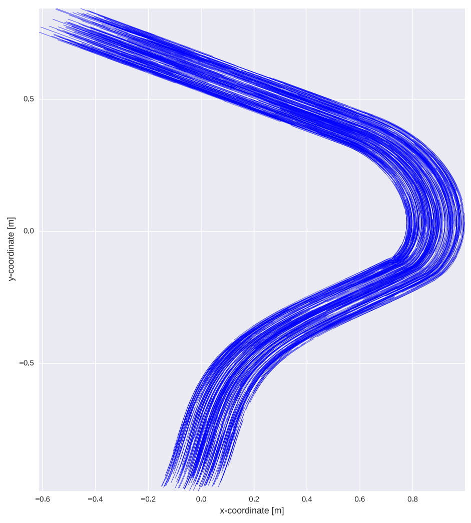
(i) Prediction error map of split II evaluated on $f^{\mathrm{I}}(\cdot, w)$



(ii) Worst 5% of the prediction trajectories of split II



(iii) Sliding Window: First 5% of the observation trajectories of split II



(iv) Random: 5% of the observation trajectories of split II

Fig. 6-22: The continuous learning setting in the case “Blocked Road”

6.8.3 Evaluation Continuous Learning

It was shown that retraining a fitted model is an effective approach for a continuously learning mechanism. The reuse of the pre-trained model saves computational runtime when adapting to new movement patterns. The initial idea that reinforced learning of error-prone trajectories improves the prediction quality is validated. But this specific setting and the applied synthetic datasets caused a worse performance than the *Sliding Window* and the *Random Trajectory* method. Due to the density of the datasets, training the worst trajectories caused a local “overheat” of training data. Hence, redundant bad performing samples are unnecessarily trained together while less error-prone observation-prediction pairs were ignored during the retraining.

This undesirable behaviour was identified by the error map visualisation method.

For future investigations, a more sophisticated approach has to be developed that has a better regularised merging method. An excessive redundancy in good performing trajectories but also in bad performing trajectories has to be avoided. By this means, the retraining dataset has to be combined in such a fashion that the trajectories cover the scenario in the most effective way. There is a presumption that such an approach has a better cost-performance ratio than a Sliding Window or a randomised method. Furthermore, there is a urgent need for realistic datasets.

6.9 Model Evaluation Latency

In order to measure the latency of the default sequence to sequence model, an observation dataset with 70716 single trajectories of length 8 was evaluated. The RNN had 128 hidden states and a depth of 1. The GeForce GTX 1070 needed $0.68\ [s]$ to perform this task. Hereby, a batch size of 512 was used. When the batch size was doubled to 1028, the model evaluation took $0.60\ [s]$. A smaller dataset consisting of 7469 observation trajectories needed a runtime of $0,045\ [s]$ to be evaluated. Without any further in-depth investigations, it can be stated that such a model seems to be real-time capable on modern GPU devices.

7 Conclusion and Outlook

This thesis presented a holistic approach for pedestrian trajectory prediction with recurrent neural networks at urban scenarios. State of the art Long Short-Term Memory designs where deployed in order to learn preprocessed trajectories in an end-to-end fashion. The architecture was tested and validated on publicly available pedestrian datasets. In addition to that, datasets obtained by laser scanner measurements of various road users and synthetic human trajectory datasets have been successfully applied. 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 model was hereby measured by several performance evaluation criteria. On this basis, it was qualitatively shown that the proposed method outperformed two basic baseline models on all datasets. During the development of this work, a flexible Python framework was implemented that contains complex pre-processing and visualisation routines.

Further analyses demonstrated that uncertainties in the input of such model do not necessarily lead to instabilities in the prediction. This behaviour was observed on manually evaluated stability visualisations. 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.

To answer the **first research question**, several in-depth investigations have been carried out. In this respect, the effect of hyper-parameters, training configurations and RNN architectures were assessed. It turned out that a shallow advanced sequence to sequence architecture with 128 hidden dimensions had the best cost-performance ratio. Furthermore, the ideal amount of training epochs and the best performing optimiser were determined.

A continuous learning scenario was simulated by two synthetic scenarios. Different data merging methods were proposed that are intended to retrain an already fitted prediction model. In this way, the model efficiently adapted to time-dependent changes of movement patterns. The result of this approach provided insights to answer to the **second research question**: An efficient continuous learning method for intersections scenarios can be achieved by a reinforced training of badly predicted observations. In this process, redundancy in the training dataset has to be avoided. Future investigations should validate the findings on more realistic datasets.

The presented machine learning approach steadily depends on the quality and density of the spatial datasets. It was shown that a regional limited lack of training trajectories can lead to high prediction errors in these certain areas. Hence, the model is not able to learn physical movement dynamics, but rather position-dependent movement patterns. For future work, it is recommended to investigate different pre-processing methods. For example, additional information such as the yaw rate of the tracked object could be taken into consideration in order to provide the neural net additional information about movement dynamics. A different approach to improve the prediction would be to replace the RNN with a traditional prediction model, such as the Kalman filter, if the current observation is located in an area with low data density.

Although compelling results have been achieved in this thesis, it has to be admitted that the applied datasets are not quite appropriate for the presented setting. The ETH datasets do not depict at an intersection scenario, the laser datasets contain vehicles and the Vissim datasets are quite unrealistic. Therefore, it is inevitable to validate the results of this thesis by using realistic pedestrian trajectory datasets captured at intersections. These new datasets could be obtained by camera-equipped drones that record crowded urban scenarios.

Acknowledgements

I would like to thank my supervisor Julian Bock for the help, proofreading and advice during the development of this thesis.

8 Literature

[ABA15] ABADI, M.; AGARWAL, A.; BARHAM, P.; BREVDO, E.; CHEN, Z.; CITRO, C., et al.,
TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems
Software available from tensorflow.org, 2015

[ADA17a] ADAC - ALLGEMEINER DEUTSCHER AUTOMOBIL-CLUB,
Statistik - Fußgänger
https://www.adac.de/_mmm/pdf/statistik_5_1_fu%C3%9Fgaenger_z_43081.pdf, accessed 04-March-2017, 2017

[ADA17b] ADAC - ALLGEMEINER DEUTSCHER AUTOMOBIL-CLUB,
Statistik - Unfallarten und Verunglückte
https://www.adac.de/_mmm/pdf/statistik_8_2_unfallarten_42784.pdf, accessed 04-March-2017, 2017

[ADA17c] ADAC - ALLGEMEINER DEUTSCHER AUTOMOBIL-CLUB,
Statistik - Unfälle Ortslage
https://www.adac.de/_mmm/pdf/statistik_4_1_innerorts_42803.pdf, accessed 04-March-2017, 2017

[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

[BA14] BA, J.; CARUANA, R.,
Do deep nets really need to be deep?
Advances in neural information processing systems, 2014, pp. 2654–2662

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

[BAL16] BALLAN, L.; CASTALDO, F.; ALAHI, A.; PALMIERI, F.; SAVARESE, S.,
Knowledge Transfer for Scene-specific Motion Prediction
CoRR abs/1603.06987 (2016)

[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)

[BEN94] BENGIO, Y.; SIMARD, P.; FRASCONI, P.,
Learning long-term dependencies with gradient descent is difficult
IEEE transactions on neural networks 5.2 (1994), pp. 157–166

[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, pp. I-115–I-123

[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

[BOH08] BOHM, A.; JONSSON, M.,
Supporting real-time data traffic in safety-critical vehicle-to-infrastructure communication
Local Computer Networks, 2008. LCN 2008. 33rd IEEE Conference on, IEEE, 2008, pp. 614–621

[BRO16] BROUWER, N.; KLOEDEN, H.; STILLER, C.,
Comparison and evaluation of pedestrian motion models for vehicle safety systems
2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC), 2016, pp. 2207–2212

[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)

[CHO15] CHOLLET, F.,
Keras - Deep Learning library for Theano and TensorFlow
https://github.com/fchollet/keras, accessed 01-December-2016, 2015

[COU15] COURBARIAUX, M.; BENGIO, Y.; DAVID, J.-P.,
BinaryConnect: Training Deep Neural Networks with binary weights during propagations
In: *Advances in Neural Information Processing Systems 28*, ed. by CORTES, C.; LAWRENCE, N. D.; LEE, D. D.; SUGIYAMA, M.; GARNETT, R., Curran Associates, Inc., 2015, pp. 3123–3131

[DEU17] DEUTSCHES FORSCHUNGSZENTRUM FÜR KÜNSTLICHE INTELLIGENZ GMBH,
SADA - Smart Adaptive Data Aggregation
http://robotik.dfki-bremen.de/de/forschung/projekte/sada.html, accessed 2-March-2017, 2017

[DUC11] DUCHI, J.; HAZAN, E.; SINGER, Y.,
Adaptive subgradient methods for online learning and stochastic optimization
Journal of Machine Learning Research 12.Jul (2011), pp. 2121–2159

[ELM90] ELMAN, J. L.,
Finding structure in time
Cognitive science 14.2 (1990), pp. 179–211

[GER00] GERS, F. A.; SCHMIDHUBER, J. A.; CUMMINS, F. A.,
Learning to Forget: Continual Prediction with LSTM
Neural Comput. 12.10 (2000), pp. 2451–2471

[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)

[GMB16] GMBH, R. B.,
Lokale Clouds für mehr Verkehrssicherheit
http://www.bosch-presse.de/pressportal/de/de/lokale-clouds-fuer-mehr-verkehrssicherheit-63296.html, accessed 06-February-2017, 2016

[GRA05] GRAVES, A.; SCHMIDHUBER, J.,
Framewise phoneme classification with bidirectional LSTM and other neural network architectures
Neural Networks 18.5 (2005), pp. 602–610

[GRA09] GRAVES, A.; LIWICKI, M.; FERNÁNDEZ, S.; BERTOLAMI, R.; BUNKE, H.; SCHMIDHUBER, J.,
A novel connectionist system for unconstrained handwriting recognition
IEEE transactions on pattern analysis and machine intelligence 31.5 (2009), pp. 855–868

[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)

[HAS15] HASHIMOTO, Y.; YANLEI, G.; HSU, L.-T.; SHUNSUKE, K.,
A Probabilistic Model for the Estimation of Pedestrian Crossing Behavior at Signalized Intersections
Proceedings of the 2015 IEEE 18th International Conference on Intelligent Transportation Systems, ITSC '15, Washington, DC, USA: IEEE Computer Society, 2015, pp. 1520–1526

[HEL90] HELBING, D.,
Physical Modeling of the Dynamic Behavior of Pedestrians
PhD thesis, Georg-August-Universität zu Göttingen, 1990

[HEL97] HELBING, D.,
Selbstorganisation kollektiver Phänomene
In: *Verkehrsdynamik*, Springer, 1997, pp. 37–44

[HER13] HERMANS, M.; SCHRAUWEN, B.,
Training and Analysing Deep Recurrent Neural Networks
In: *Advances in Neural Information Processing Systems 26*, ed. by BURGES, C. J. C.; BOTTOU, L.; WELLING, M.; GHAHRAMANI, Z.; WEINBERGER, K. Q., Curran Associates, Inc., 2013, pp. 190–198

[HOC01] HOCHREITER, S.; BENGIO, Y.; FRASCONI, P.,
Gradient Flow in Recurrent Nets: the Difficulty of Learning Long-Term Dependencies
(2001)

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

[HOR91] HORNIK, K.,
Approximation capabilities of multilayer feedforward networks
Neural networks 4.2 (1991), pp. 251–257

[I2E17] I2EASE,
Intelligence for efficiently electrified and automated driving through sensor networking
http://www.cerm.rwth-aachen.de/i2ease/, accessed 2-March-2017, 2017

[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

[KÄD16] KÄDING, C.; RODNER, E.; FREYTAG, A.; DENZLER, J.,
Active and Continuous Exploration with Deep Neural Networks and Expected Model Output Changes
(2016)

[KAL15] KALCHBRENNER, N.; DANIHELKA, I.; GRAVES, A.,
Grid Long Short-Term Memory
CoRR abs/1507.01526 (2015)

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

[KEL14] KELLER, C. G.; GAVRILA, D. M.,
Will the pedestrian cross? A study on pedestrian path prediction
IEEE Transactions on Intelligent Transportation Systems 15.2 (2014), pp. 494–506

[KIN14] KINGMA, D. P.; BA, J.,
Adam: A Method for Stochastic Optimization
CoRR abs/1412.6980 (2014)

[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

[LAW97] LAWRENCE, S.; GILES, C. L.; TSOI, A. C.; BACK, A. D.,
Face recognition: A convolutional neural-network approach
IEEE transactions on neural networks 8.1 (1997), pp. 98–113

[LIP15] LIPTON, Z. C.,
A Critical Review of Recurrent Neural Networks for Sequence Learning
CoRR abs/1506.00019 (2015)

[LIV10] LIVNAT, A.; PAPADIMITRIOU, C.; PIPPENGER, N.; FELDMAN, M. W.,
Sex, mixability, and modularity
Proceedings of the National Academy of Sciences 107.4 (2010), pp. 1452–1457

[LUB10] LUBER, M.; STORK, J. A.; TIPALDI, G. D.; ARRAS, K. O.,
People tracking with human motion predictions from social forces
Robotics and Automation (ICRA), 2010 IEEE International Conference on, IEEE, 2010, pp. 464–469

[MEI14] MEISSNER, D.; REUTER, S.; STRIGEL, E.; DIETMAYER, K.,
Intersection-based road user tracking using a classifying multiple-model PHD filter
IEEE Intelligent Transportation Systems Magazine 6.2 (2014), pp. 21–33

[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

[PAS12] PASCANU, R.; MIKOLOV, T.; BENGIO, Y.,
Understanding the exploding gradient problem
CoRR abs/1211.5063 (2012)

[PED11] PEDREGOSA, F.; VAROQUAUX, G.; GRAMFORT, A.; MICHEL, V.; THIRION, B.; GRISEL, O.; BLONDEL, M.; PRETTENHOFER, P.; WEISS, R.; DUBOURG, V.; VANDERPLAS, J.; PASSOS, A.; COURNAPEAU, D.; BRUCHER, M.; PERROT, M.; DUCHESNAY, E.,

Scikit-learn: Machine Learning in Python
Journal of Machine Learning Research 12 (2011), pp. 2825–2830

[PEL09] PELLEGRINI, S.; ESS, A.; SCHINDLER, K.; VAN GOOL, L.,
You'll never walk alone: Modeling social behavior for multi-target tracking
Computer Vision, 2009 IEEE 12th International Conference on, IEEE, 2009, pp. 261–268

[PTV17] PTV PLANUNG TRANSPORT VERKEHR,
PTV Vissim - Verkehrsfluss-Simulationssoftware
http://vision-traffic.ptvgroup.com/de/produkte/ptv-vissim/, accessed 5-February-2017, 2017

[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

[REH15] REHDER, E.; KLOEDEN, H.,
Goal-directed pedestrian prediction
Proceedings of the IEEE International Conference on Computer Vision Workshops, 2015, pp. 50–58

[ROB16] ROBICQUET, A.; SADEGHIAN, A.; ALAHI, A.; SAVARESE, S.,
Learning Social Etiquette: Human Trajectory Understanding In Crowded Scenes
Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part VIII, 2016, pp. 549–565

[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

[SCH97] SCHUSTER, M.; PALIWAL, K.,
Bidirectional Recurrent Neural Networks
Trans. Sig. Proc. 45.11 (1997), pp. 2673–2681

[SEE14] SEELIGER, F.; WEIDL, G.; PETRICH, D.; NAUJOKS, F.; BREUEL, G.; NEUKUM, A.; DIETMAYER, K.,
Advisory warnings based on cooperative perception
Intelligent Vehicles Symposium Proceedings, 2014 IEEE, IEEE, 2014, pp. 246–252

[SER11] SERMANET, P.; LECUN, Y.,
Traffic sign recognition with multi-scale convolutional networks
Neural Networks (IJCNN), The 2011 International Joint Conference on, IEEE, 2011, pp. 2809–2813

[SHA16] SHAH, R.; ROMIJNDERS, R.,
Applying Deep Learning to Basketball Trajectories
CoRR abs/1608.03793 (2016)

[SRI14] SRIVASTAVA, N.; HINTON, G.; KRIZHEVSKY, A.; SUTSKEVER, I.; SALAKHUT-DINOV, R.,
Dropout: A Simple Way to Prevent Neural Networks from Overfitting
Journal of Machine Learning Research 15 (2014), pp. 1929–1958

[SUT13] SUTSKEVER, I.,
Training recurrent neural networks
PhD thesis, University of Toronto, 2013

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

[TAM10] TAMURA, Y.; FUKUZAWA, T.; ASAMA, H.,
Smooth collision avoidance in human-robot coexisting environment
2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2010, pp. 3887–3892

[THE16] THEANO DEVELOPMENT TEAM,
Theano: A Python framework for fast computation of mathematical expressions
arXiv e-prints abs/1605.02688 (2016)

[TIE12] TIELEMAN, T.; HINTON, G.,
RmsProp: Divide the gradient by a running average of its recent magnitude
Neural Networks for Machine Learning, 2012

[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)

[WAL14] WALKER, J.; GUPTA, A.; HEBERT, M.,
Patch to the future: Unsupervised visual prediction
Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2014, pp. 3302–3309

[WER90] WERBOS, P. J.,
Backpropagation through time: what it does and how to do it
Proceedings of the IEEE 78.10 (1990), pp. 1550–1560

[WIL90] WILLIAMS, R. J.; PENG, J.,
An Efficient Gradient-Based Algorithm for On-Line Training of Recurrent Network Trajectories
Neural Computation 2 (1990), pp. 490–501

[WOR17] WORLD HEALTH ORGANIZATION,
Global Status Report on Road Saftey 2013
http://www.who.int/violence_injury_prevention/road_safety_status/2013/en/, accessed 3-March-2017, 2017

[XIA14] XIAO, T.; ZHANG, J.; YANG, K.; PENG, Y.; ZHANG, Z.,
Error-Driven Incremental Learning in Deep Convolutional Neural Network for Large-Scale Image Classification
ACM Multimedia, 2014

[YAM11] YAMAGUCHI, K.; BERG, A. C.; ORTIZ, L. E.; BERG, T. L.,
Who are you with and where are you going?
Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on, IEEE, 2011, pp. 1345–1352

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

[ZAR14] ZAREMBA, W.; SUTSKEVER, I.,
Learning to Execute
CoRR abs/1410.4615 (2014)

[ZEI12] ZEILER, M. D.,
ADADELTA: An Adaptive Learning Rate Method
CoRR abs/1212.5701 (2012)

9 List of Abbreviations

ADAS	Advanced Driver Assistance Systems
ANN	Artificial Neural Network
ATC	Aldenhoven Testing Center
cf.	compare with
CNN	Convolutional Neural Network
DNN	Deep Neural Network
E.g.	For example
e.g.	for example
ETH	Swiss Federal Institute of Technology
Fig.	Figure
FNN	Feedforward Neural Network
GPS	Global Positioning System
GPU	Graphics Processing Unit
I2EASE	Intelligence for efficiently electrified and automated driving through sensor networking
I2V	Infrastructure-to-Vehicle Communication
Ko-FAS	Cooperative driver assistance systems
LSTM	Long-Short Term Memory
MD	Mean Displacement
MFD	Mean Final Displacement
MSD	Mean Squared Displacement
MSE	Mean Squared Error
NLP	Natural language processing
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
SADA	Smart Adaptive Data Aggregation
Seq2Seq	Sequence to Sequence
SF	Social Forces
SGD	Stochastic Gradient Decent
V2X	Vehicle-to-everything Communication
VRU	Vulnerable Road Users

10 List of Symbols

v_{avg}	Average velocity
Hz	Herz
f_s	Sampling rate
Y_i	Trajectory i
m	Meter
σ	Sigmoid function
$tanh$	Tangens Hyperbolicus
Σ	standard deviation of the Gaussian kernel
O	Observation Trajectories
P	True prediction trajectories
$\hat{P}$	Model prediction trajectories
α	Train split factor
$\mathcal{L}$	Loss function
p	Dropout probability
N_{pred}	Number of data points of the prediction
N_{obs}	Number of data points of the observation
h	Hidden dimensions of a neural net
η	Learning rate of an optimiser
b	Training batch-size
e	Number of training epochs
d	Depth of a RNN
$\mathcal{A}$	Event at an intersection scenario
t	Time
$f(\cdot, w)$	Trained model
E	Prediction Error

11 Appendix

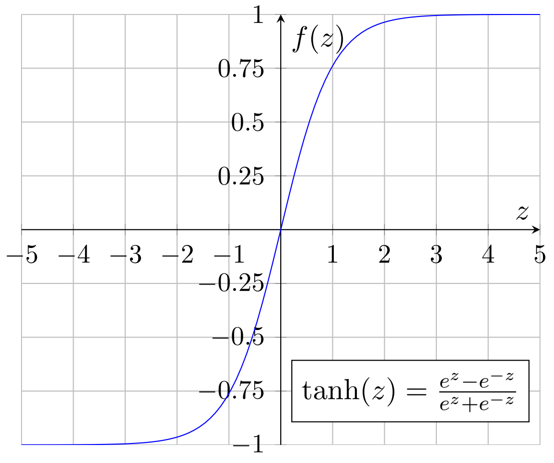
11.1 Attachment A: Default Configuration File

Listing 11.1: Example configuration file

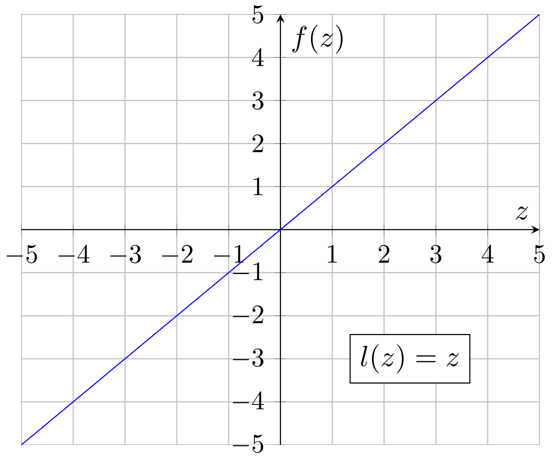
```
[general]
save_dir = eth_output_dir
data = eth
verbose = 2
random_seed = 42
save_weights = True
save_scaled_trajectories = True

[model]
architecture = seq2seq
hidden_dim = 128
nb_epoch = 200
loss = mean_squared_displacement
batch_size = 32
scale_learning_rate = 1.0
clipvalue = 1
optimizer = adam
earlyStopping_patience = 20
depth = 1
dropout = 0.2

[preprocessing]
split = 0.7
shuffle_all_trajectories = False
observationLength = 8
predictionLength = 12
overlapping = 0
smoothing_sigma = 1

[plotting]
n_stability_plots = 100
n_kde_stability_plots = 100
n_heatmap_stability_plots = 100
n_simple_trajectory_plots = 100
n_double_trajectory_plots = 100
dpi = 100

[epoch_analysis_2D]
nb_epoch = 50, 100, 150, 200, 250, 300, 350, 400, 450, 500, 550, 600, 650, 700
depth = 1, 2, 3
```

11.2 Attachment B: Neural network activation functions

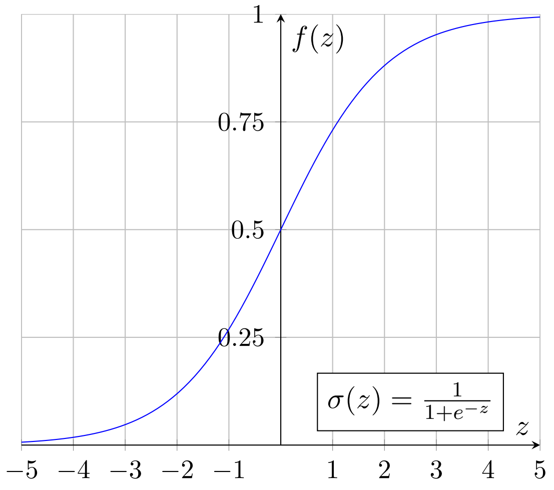
tanh



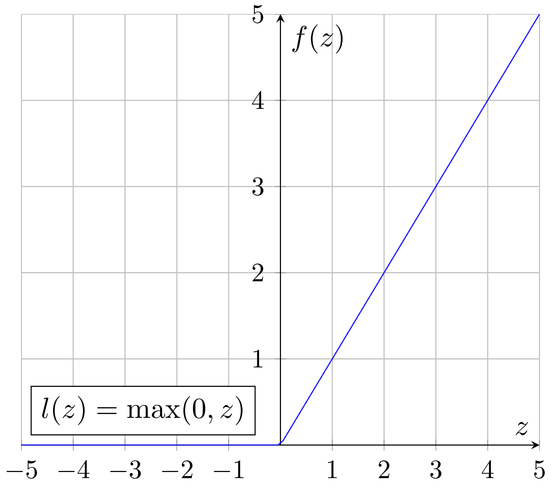
linear



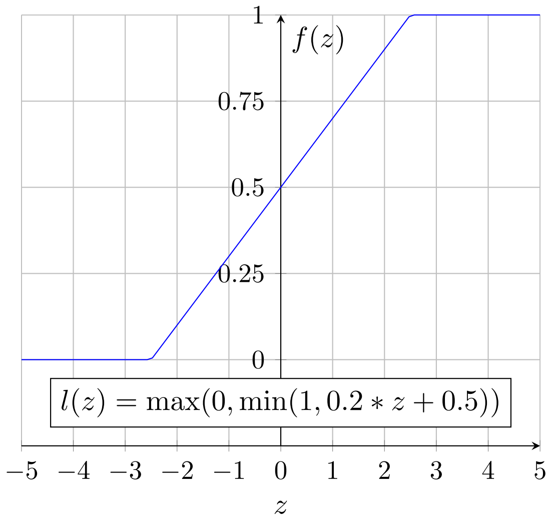
sigmoid



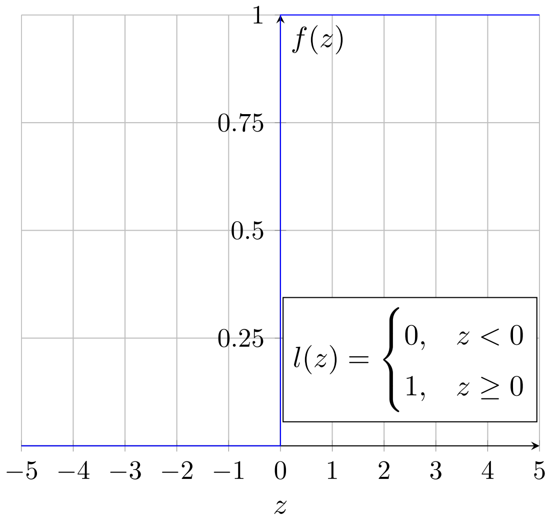
ReLU



Hard sigmoid



Heaviside step function



11.3 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 in section 5 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:

$$Y_i = \begin{pmatrix} x_i^1 & y_i^1 \\ x_i^2 & y_i^2 \\ \vdots & \vdots \\ x_i^{N-1} & y_i^{N-1} \\ x_i^N & y_i^N \end{pmatrix} \in \mathbb{R}^{N\times 2}. \qquad \text{Eq. 11-1}$$

Mean Squared Displacement

$$\mathcal{L}_{MSD}(\hat{Y}, Y) := \frac{1}{S}\sum_{i=1}^{S}\sum_{k=1}^{N}\left(x_i^k - \hat{x}_i^k\right)^2 + \left(y_i^k - \hat{y}_i^k\right)^2 \qquad \text{Eq. 11-2}$$

Mean Squared Error

$$\mathcal{L}_{MSE}(\hat{Y}, Y) := \frac{1}{S}\frac{1}{N}\sum_{i=1}^{S}\sum_{k=1}^{N}\left(x_i^k - \hat{x}_i^k\right)^2 + \left(y_i^k - \hat{y}_i^k\right)^2 \qquad \text{Eq. 11-3}$$

Mean Final Displacement

$$\mathcal{L}_{MFD}(\hat{Y}, Y) := \frac{1}{S}\sum_{i=1}^{S}\sqrt{\left(x_i^N - \hat{x}_i^N\right)^2 + \left(y_i^N - \hat{y}_i^N\right)^2} \qquad \text{Eq. 11-4}$$

Mean Displacement

$$\mathcal{L}_{MD}(\hat{Y}, Y) := \frac{1}{S}\sum_{i=1}^{S}\sum_{k=1}^{N}\sqrt{\left(x_i^k - \hat{x}_i^k\right)^2 + \left(y_i^k - \hat{y}_i^k\right)^2} \qquad \text{Eq. 11-5}$$

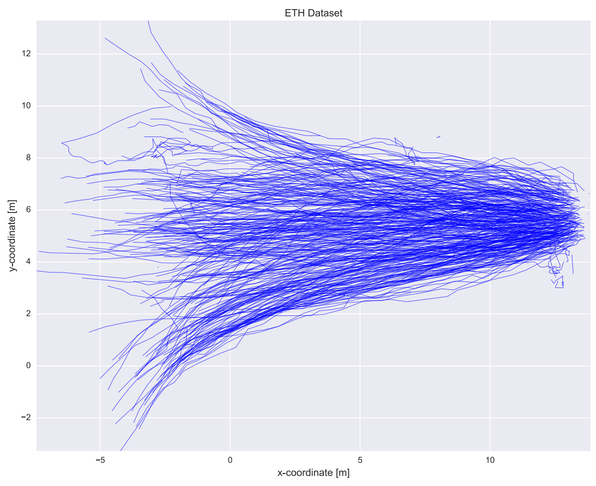
11.4 Attachment D: Dataset details

Dataset	Number of trajectories	Average trajectory length $[m]$	$v_{avg}[\frac{km}{h}]$	Sampling rate $[Hz]$	Data points
ETH	359	13.17	4.9	2.5	8888
Hotel	389	6.58	3.76	2.5	6538
ATC	8	434.00	2.97	2.5	9980
Vissim Case 1	1056	60.67	4.93	2.5[1]	118168
Vissim Case 2	1056	61.68	4.94	2.5[1]	120061
Vissim Case 3	1056	61.12	4.94	2.5[1]	118999
Laser I	2385	157.89	40.71	2.5[2]	83277
Laser II	646	104.20	35.69	2.5[2]	17714
Laser III	1952	116.24	30.07	2.5[2]	67968

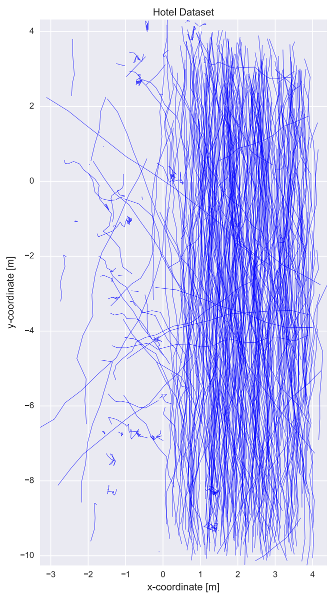
Fig. 11-1: Dataset statistics

11.4.1 ETH Datasets

Vision-based pedestrian tracking performed at public spaces in Zürich. Published by Pellegrini et al. [PEL09].



(i) ETH dataset



(ii) Hotel dataset

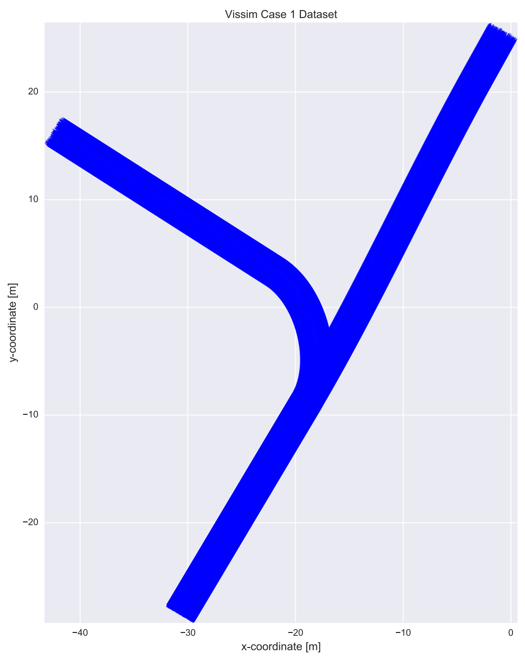
Fig. 11-2: ETH dataset collection consists of the *ETH* dataset and the *Hotel* dataset

[1] Downsampled from originally $10\ [Hz]$ to $2.5\ [Hz]$

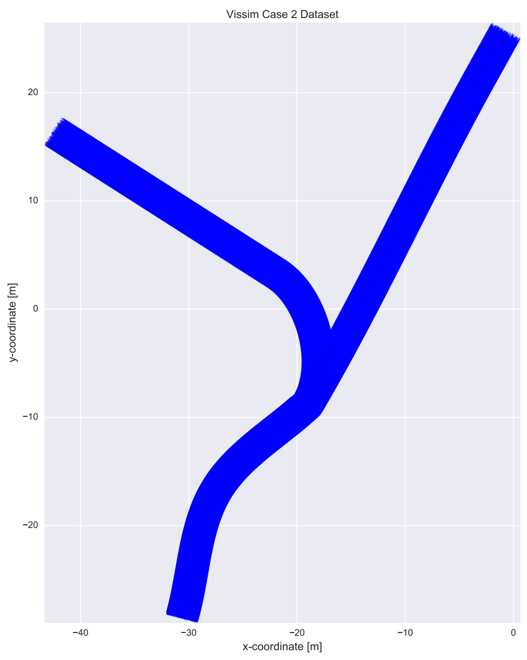
[2] Downsampled from originally $25\ [Hz]$ to $2.5\ [Hz]$

11.4.2 Vissim Datasets

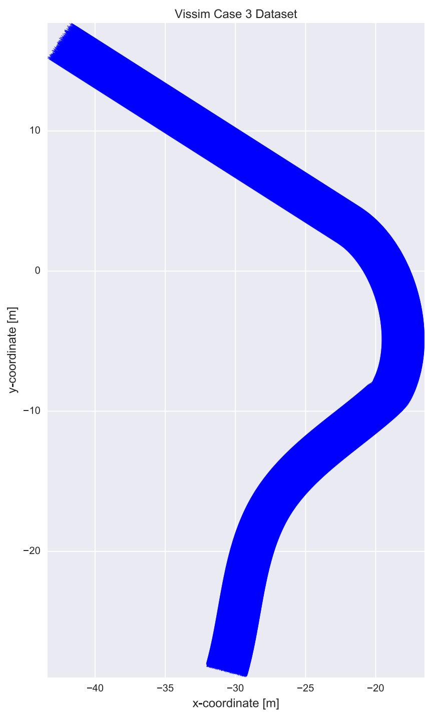
Generated with Vissim that makes use of a social forces model [PTV17].



(i) Vissim Dataset Case 1



(ii) Vissim Dataset Case 2

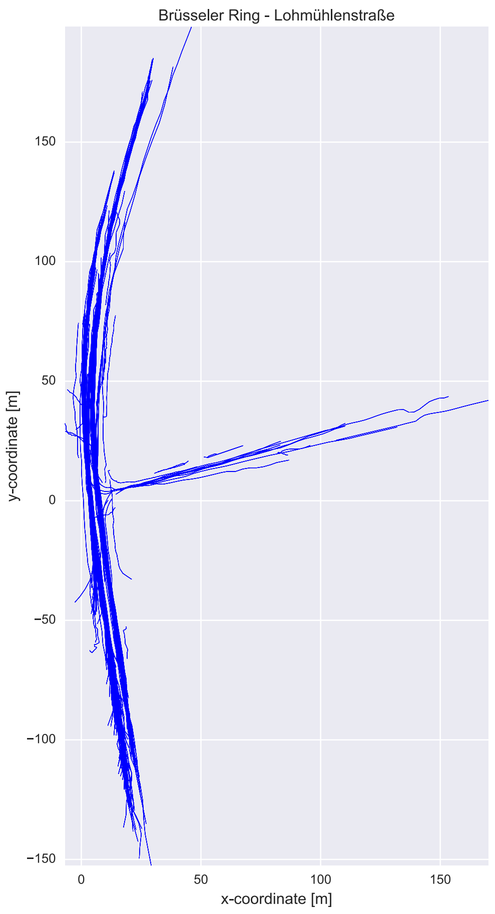


(iii) Vissim Dataset Case 3

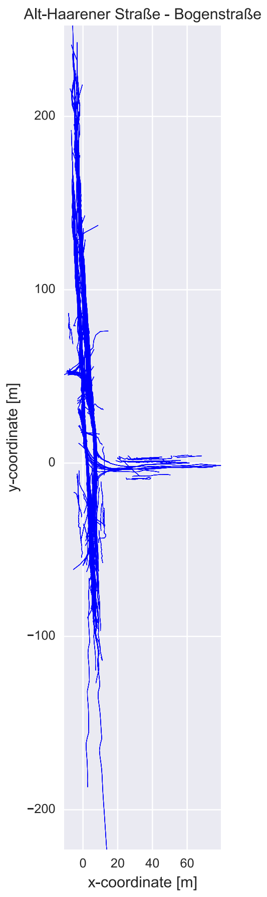
Fig. 11-3: Vissim Datasets

11.4.3 Laser Datasets

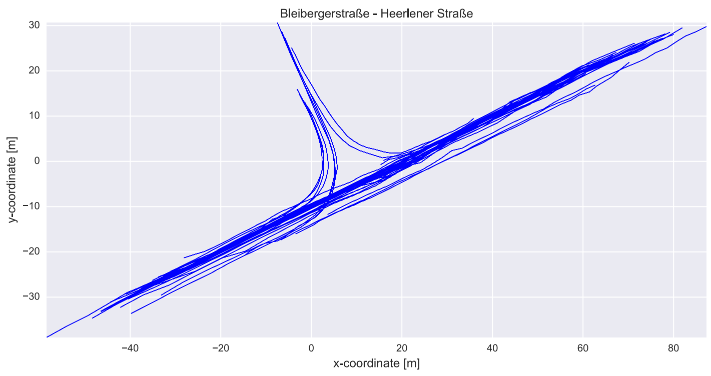
Captured at intersections in Aachen. The majority of the tracked objects are vehicles.



(i) Laser I: Intersection Brüsseler Ring - Lohmühlenstraße



(ii) Laser III: Intersection Alt-Haarener Straße - Bogenstraße



(iii) Laser II: Intersection Bleibergerstraße - Heerlener Straße

Fig. 11-4: Laser datasets

11.4.4 Aldenhoven Testing Center Dataset

Pedestrian movements at an intersection at the ATC. Captured with a high-precision GPS receiver that was carried by pedestrians.

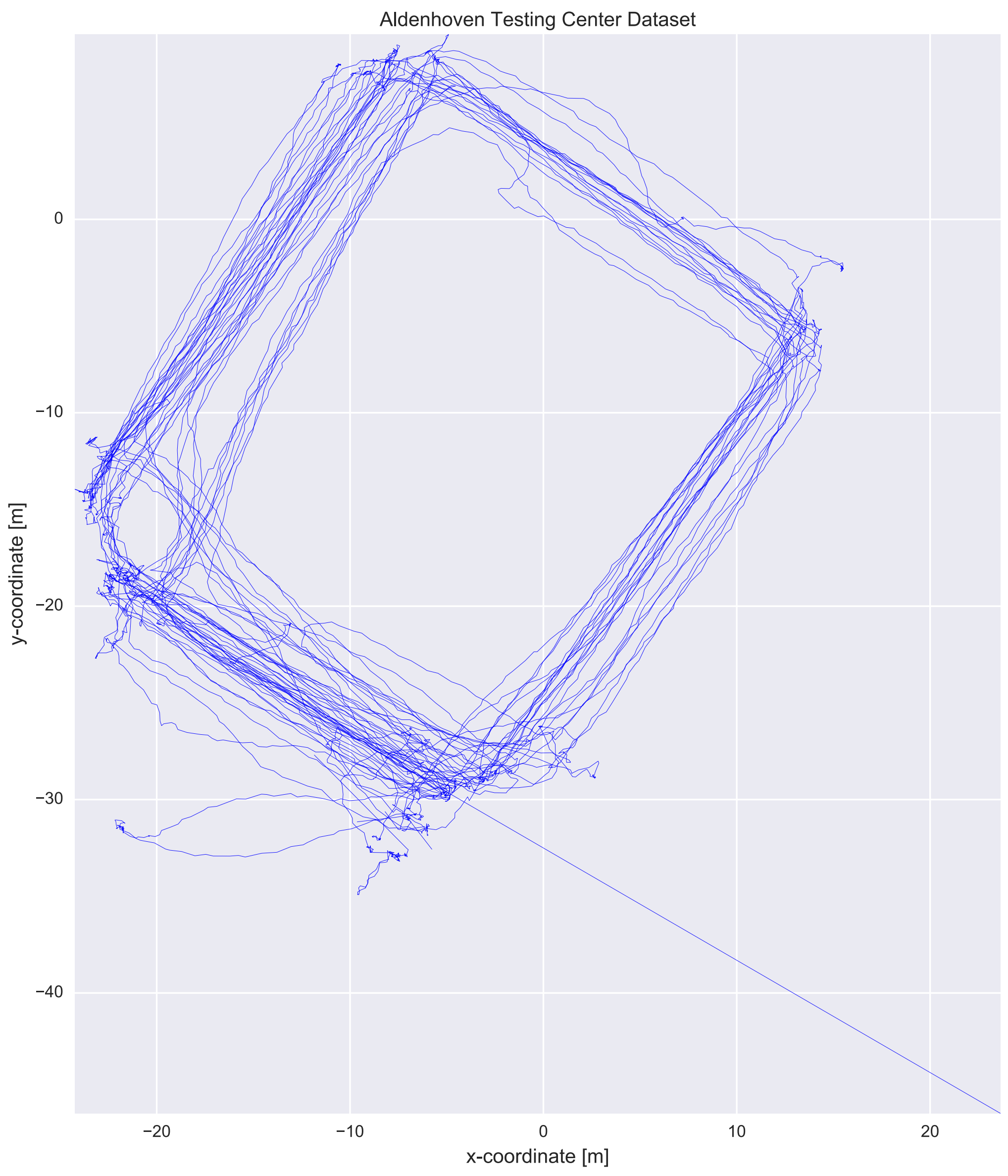


Fig. 11-5: Aldenhoven Testing Center dataset